MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

STATE UNIVERSITY«KYIV AVIATION INSTITUTE»

Faculty of Aeronavigation, Electronics and Telecommunications

Department of aviation computer-integrated complexes

ADMIT TO DEFENSE

Head of the graduating department

_____ Viktor M. Sineglazov

“____”_____2025y.

QUALIFICATION WORK

(EXPLANATORY NOTE)

OF THE GRADUATE OF THE EDUCATIONAL DEGREE

“BACHELOR”

Specialty 151 "Automation and computer-integrated technologies"

Educational and professional program "Computer-integrated technological processes and production"

Theme: Intelligent system for generating artificial thermal images of asynchronous motor faults

Performer: student of Ba-121-21-1-Iz group Koziura V.

Supervisor: D.Sc. (Tech.), Prof. Viktor SYNEGLAZOV

Standard controller: _____ Filyashkin M.K.

(sign)

Kyiv – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра авіаційних комп'ютерно-інтегрованих комплексів

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри

_____ Віктор СИНЄГЛАЗОВ

“____” _____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

“БАКАЛАВР”

Спеціальність 151 "Автоматизація, та комп’ютерно-інтегровані технології"

Освітньо-професійна програма "Комп’ютерно-інтегровані технологічні процеси і виробництва"

Тема: Інтелектуальна система генерації штучних теплових зображень несправностей асинхронного двигуна

Виконавець: студент Ба-121-21-1-ІЗ групи Козюра Владислав Сергійович

Керівник: д.т.н., проф. Синєглазов Віктор Михайлович

Нормоконтролер: _____ Філяшкін М.К.

(підпис)

Київ – 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра авіаційних комп’ютерно-інтегрованих комплексів

Освітній ступінь: бакалавр

Спеціальність 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Інформаційні техногії та інженерія авіаційних комп’ютерних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Віктор СИНЄГЛАЗОВ

“_____”_____2025р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Козюри Владислава Сергійовича

1. Тема роботи : “ Інтелектуальна система генерації штучних теплових зображень несправностей асинхронного двигуна ”.

2. Термін виконання роботи : з 29.04.2024р. до 12.06.2024р.

3. Вихідні дані до роботи :
– Набір реальних теплових зображень індукційних двигунів у різних технічних станах;
– Опис процесу попередньої обробки термограм (CenterCrop, Resize, Normalize, аугментації);
– Архітектурні специфікації та ваги реалізованої моделі cWGAN-GP;
– Натреновані ваги класифікатора (CNN/ResNet-18) для оцінки синтетичних зразків.

4. Перелік обов’язкового графічного матеріалу: графіки, таблиці, зображення, діаграми.

5. Календарний план – графік

Етап роботи	Термін виконання	Підпис
Підготовка та огляд літератури	10.03.2024 – 17.03.2024	Виконано
Розробка методів зменшення вимірності	18.03.2024 – 24.03.2024	Виконано
Аналіз РНМ для обробки даних	25.03.2024 – 31.03.2024	Виконано
Проектування інтелектуальних систем	01.04.2024 – 07.04.2024	Виконано
Огляд програмного забезпечення	08.04.2024 – 14.04.2024	Виконано
Розробка та навчання моделі	15.04.2024 – 30.04.2024	Виконано
Експериментальна оцінка	01.05.2024 – 14.05.2024	Виконано
Аналіз результатів	15.05.2024 – 21.05.2024	Виконано
Підготовка звіту	22.05.2024 – 31.05.2024	Виконано
Остаточна перевірка та подання	01.06.2024- 12.06.2024	Виконано

Дата видачі завдання: 25 березня 2024 р.

Керівник дипломної роботи _____Синєглазов В.М.

(підпис)

Завдання прийняв до виконання _____Козюра В. С.

(підпис)

STATE UNIVERSITY«KYIV AVIATION INSTITUTE»

Faculty of Aeronautics,Electronics and Telecommunications

Department of aviation computer-integrated complexes

Educational degree : Bachelor

Specialty 151 "Automation and computer-integrated technologies"

Educational and professional program " Information technology and engineering of aviation computer systems”

APPROVED

Head of the graduation department

_____ Viktor SINEGLAZOV

“____”_____2025

TASK

For the student’s qualification work

Koziura Vladyslav S.

1. Work topic: " Intelligent system for generating artificial thermal images of asynchronous motor faults ".

2. The term of the work: from 04/29/2024. until 03.06.2024

3. Input data for the work:

– A set of real thermal images of induction motors in various technical states;

– Description of the process of preprocessing thermograms (CenterCrop, Resize, Normalize, augmentation);

– Architectural specifications and weights of the implemented cWGAN-GP model;

– Trained weights of the classifier (CNN/ResNet-18) for evaluating synthetic samples.

4. List of mandatory graphic material: graphs, tables, images, diagrams.

5. Calendar plan – schedule

Stage of work	Термін виконання	**Signature**
Preparation and review literature	10.03.2024 – 17.03.2024	Done
Development of methods dimensionality reduction	18.03.2024 – 24.03.2024	Done
Analysis of RNA for data processing	25.03.2024 – 31.03.2024	Done
Designing intelligent systems	01.04.2024 – 07.04.2024	Done
Software review software	08.04.2024 – 14.04.2024	Done
Development and training models	15.04.2024 – 30.04.2024	Done
Experimental rating	01.05.2024 – 14.05.2024	Done
Analysis of results	15.05.2024 – 21.05.2024	Done
Preparation of the report	22.05.2024 – 31.05.2024	Done
Final check and presentation	01.06.2024- 12.06.2024	Done

Issue data of the task : 25 March 2024

Supervisor : _____ Viktor SINEGLAZOV

(sign)

The task was accepted by : _____ KOZIURA V.S.

(sign)

ABSTRACT

Thesis: 111 pp., 19 figures, 3 tables, 2 appendixes, 16 sources.

Keywords: Machine Learning; Deep Learning; Neural Networks; Wasserstein Generative Adversarial Networks; Conditional GAN; Thermal Image Synthesis; Fréchet Inception Distance; Structural Similarity Index; Thermal Diagnostics; Hyperspectral Image Classification.

The purpose of the work is to develop and research methods for generating thermal images of asynchronous motors in various states of health based on generative-competitive neural networks (WGAN-GP and cWGAN-GP) to increase the efficiency of fault diagnosis.

The object of the research is the processes of diagnosing the technical condition of induction motors using thermal images and generative deep learning methods.

The subject of the research is methods for generating thermal images of induction motors using generative adversarial neural networks (WGAN-GP and cWGAN-GP) to increase the efficiency of diagnosing their technical condition.

Research methods - The work uses methods of analyzing literary sources to study modern approaches to diagnosing asynchronous motors and generative neural networks, machine learning methods for implementing and training WGAN-GP and cWGAN-GP, image processing methods to improve the quality of generated thermal images

The scientific novelty of the results - obtained This work introduces, for the first time in the domain of induction motor thermal diagnostics, a conditionally controlled WGAN-GP architecture that integrates one-hot encoding of fault classes directly into both generator and critic. Unlike prior studies that either focus on vibration spectrograms or generic image domains, our model is specifically tailored to synthesize thermal maps of asynchronous motors under varied operational regimes (load levels, cooling faults, imbalance, rotor blockages), preserving both global statistical distributions (validated by FID ≈ 62) and local structural patterns (SSIM ≈ 0.74). We also demonstrate stable training dynamics - avoiding mode collapse - through a carefully balanced gradient-penalty and update ratio (G:D = 1:5), and show that injecting conditional information at intermediate layers improves class-

specific feature fidelity.

The practical significance of the results - obtained By enriching limited real-world datasets with high-quality synthetic thermograms, our approach boosts fault-class representation - especially for rare stator and rotor defects - and raises classification accuracy from 67 % to 92 %. This enables more reliable, data-efficient thermal diagnostics without expensive physical tests and integrates smoothly into existing maintenance workflows on standard GPU platforms.

Content

LIST OF SYMBOLS, ABBREVIATIONS, TERMS

CPU - central processing unit

DL - deep learning

GAN – Generative Adversarial Network

WGAN – Wasserstein Generative Adversarial Network

WGAN-GP – Wasserstein GAN with Gradient Penalty

cWGAN-GP – Conditional WGAN-GP

CNN – Convolutional Neural Network

IR – Infrared

ROI – Region of Interest

FID – Fréchet Inception Distance

SSIM – Structural Similarity Index

MMD – Maximum Mean Discrepancy

EMD – Earth Mover's Distance

STFT – Short-Time Fourier Transform

MSE – Mean Squared Error

PSNR – Peak Signal-to-Noise Ratio

LR – Learning Rate

GPU – Graphics Processing Unit

ReLU - rectified linear unit

SIFT - scale-invariant feature transformation

STL - self-study

SURF - Accelerated reliable functions

INTRODUCTION

Topic relevance - In modern industry, asynchronous motors are one of the most common elements of electromechanical systems used in various industries. However, their reliability largely depends on the condition of bearings, which account for 40–50% of the total number of motor failures. Timely detection of faults is critically important for ensuring uninterrupted operation of equipment, reducing operating costs and preventing accidents.

Traditional diagnostic methods, in particular the analysis of vibration signals, have a number of limitations, including high sensitivity to speed fluctuations, the complexity of data pre-processing and the need for high-tech equipment. The use of thermal imaging as an alternative approach to monitoring the condition of motors allows for more stable results that do not depend on the rotation speed.

However, the effective use of thermal diagnostics requires a large volume of high-quality images of different engine states, which is often a problem due to limited data availability. Generative adversarial neural networks (GAN), in particular WGAN-GP, open up new possibilities for creating realistic thermal images that can be used to train fault diagnosis models. The use of conditional WGAN (cWGAN-GP) allows improving the quality of synthesized images by adapting them to specific fault types.

By integrating one-hot condition vectors with the WGAN-GP framework, our work establishes an end-to-end pipeline for thermal image augmentation: the generator learns to produce high-fidelity infrared maps under varying load and cooling scenarios, while the critic enforces Lipschitz continuity via gradient penalty. We evaluate the synthetic data not only with perceptual metrics such as Fréchet Inception Distance and Structural Similarity Index, but also by measuring the improvement in classification accuracy when these images augment a CNN diagnostic model. This dual validation—quantitative and task-driven—demonstrates that conditionally controlled WGAN-GP can effectively bridge gaps in real-world thermal datasets, leading to more robust, data-efficient motor fault detection systems suitable for deployment in modern industrial environments.

CHAPTER I

ASYNCHRONOUS MOTORS: OPERATING PRINCIPLES, COMMON FAULTS AND MODERN METHODS OF INTELLIGENT DIAGNOSTICS

1.1 Principle of operation and design of an induction motor

An induction motor, or induction motor, is one of the most common types of alternating current electric machines, playing a key role in providing mechanical motion in industry, energy, transport and domestic applications. Its functioning is based on the physical principle of electromagnetic induction, according to which an electromotive force is induced in a conductor placed in a changing magnetic field, which causes the appearance of a current. In the context of an induction motor, the current in the rotor is not supplied directly, but is induced by a magnetic field created by the stator windings when an alternating voltage is applied.

Structurally, an induction motor consists of two main parts: a stator and a rotor.

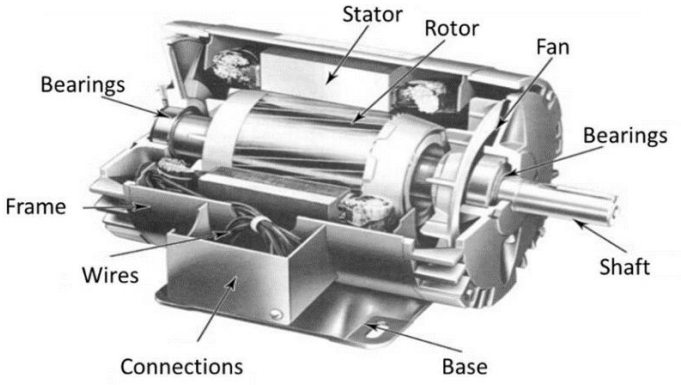


Fig.1.1 General structure of an induction motor

The stator is the stationary part of the motor, in which a three-phase or single-phase winding is placed, which, when connected to an alternating current source, forms a rotating magnetic field, which structure shown in fig.1.1. This magnetic field penetrates the air gap between the stator and the rotor, causing a current to flow through the rotor conductors. In turn, the rotor, which is the moving part of the machine, is placed inside the stator and consists either of short-circuited conductors (the so-called "drum" or "squirrel-cage" type),

or of a winding connected through slip rings to an external adjusting resistor (phase rotor)[1].

The principle of operation of an asynchronous motor is the interaction between the rotating magnetic field of the stator and the current induced in the rotor conductors. According to Lenz's law, the induced current creates its own magnetic field, which opposes the change that caused it, that is, the rotation of the stator magnetic field. As a result of this interaction, an electromagnetic torque arises, which forces the rotor to rotate in the direction of the field. In this case, the rotor cannot reach the synchronous speed of the field, because when it is reached, the speed difference between the field and the rotor disappears (the so-called "slip"), and therefore current induction ceases, and the motor loses its ability to create torque.

There are several types of induction motors. Depending on the design of the rotor, they are divided into motors with a short-circuited rotor and phase motors shown in fig 1.2.

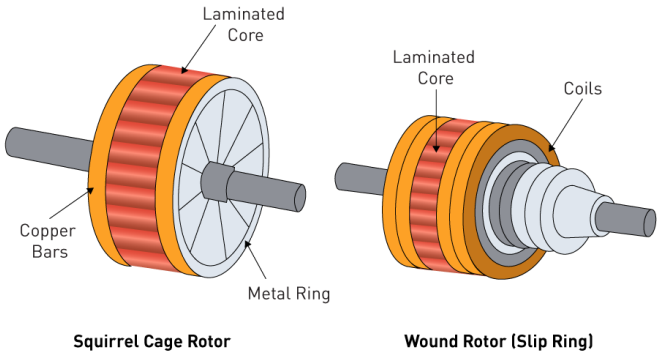


Fig. 1.2 Types of rotors: squirrel-cage and phase

The former are the most common due to their simplicity, reliability and low cost of production, although they have limited ability to regulate speed and starting torque. Motors with a phase rotor, on the contrary, allow flexible adjustment of start-up and speed parameters by connecting external resistors, but they are more expensive and more difficult to maintain. Depending on the number of supply phases, motors are divided into single-phase and three-phase. Single-phase asynchronous motors are mostly used in household appliances, where three-phase power is not available. To start such motors, additional

components are required, such as a starting capacitor. Three-phase motors, in turn, are the main working tool in industry due to their high efficiency, self-starting and ability to work with heavy loads.

Asynchronous motors have a number of undeniable advantages, including simplicity of design, mechanical reliability, low manufacturing cost, high energy efficiency in nominal mode and a wide range of applications. However, they also have certain disadvantages. These include low starting torque (especially in squirrel-cage motors), limited ability to adjust the speed of rotation without the use of additional electronic devices (for example, frequency converters), high starting currents, which can cause voltage drops in the network, as well as a decrease in the power factor at partial loads. In addition, the dependence of the speed of rotation on slip makes it difficult to accurately regulate the speed without the introduction of complex control systems[1]. Despite these limitations, asynchronous motors remain a key component in many industries. They are used to drive pumps, fans, compressors, conveyor lines, machine tools, in ventilation and air conditioning systems, household appliances (washing machines, refrigerators, vacuum cleaners), as well as in transport, including modern electric vehicles, where their efficiency and reliability are crucial.

Thus, understanding the principles of operation, design features, advantages and disadvantages of asynchronous motors is the foundation for further analysis of typical malfunctions that occur during their operation. At the same time, as modern digital technologies develop, methods of intelligent fault diagnosis, in particular using thermal imaging, are becoming increasingly relevant. This creates a bridge to the next chapter of the study, where an innovative approach will be considered - the application of a Wasserstein-Generative Adversarial Network with a gradient penaltyer (WGAN-GP) to generate thermal images that allow for the effective detection of anomalies in the operation of electric motors at the early stages[2].

1.2 The role of induction motors in industry and applications

Induction motors have become the cornerstone of modern industrial automation due to their reliability, energy efficiency and versatility in application in various industries.

Their wide implementation covers not only traditional production processes, but also modern high-tech systems - from automated lines to electric transport. In light of the global transition to smart, sustainable and automated production, the role of induction motors is constantly growing, becoming even more complex and critically important.

In the production environment, induction motors are key drive elements for a variety of equipment - conveyor lines, pumps, compressors, fans, etc. They provide the ability to operate equipment continuously even under variable load conditions without losing efficiency. Due to their simple design, durability and minimal maintenance requirements, such motors are ideal for mass production, where it is critical to ensure process stability and minimize downtime.

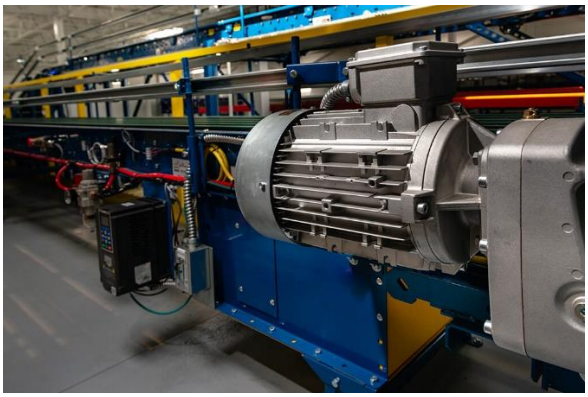


Fig. 1.3 The use of asynchronous motors in drives of conveyor systems in automated production.

In the context of modern "smart" production, where productivity and energy efficiency are of primary importance, asynchronous motors are a reliable support for all automated mechanical processes. In addition to classic manufacturing, asynchronous motors play a leading role in the development of the robotics and automation industry. Modern factories are increasingly implementing robotic manipulators, automated assembly lines and high-precision positioning devices, which require precise and controlled rotation. By using control technologies such as frequency converters (VFD) or vector control (FOC), asynchronous motors are able to provide high torque and precise speed control - especially valuable in industries where both speed and positioning accuracy are important. This

applies, in particular, to electronics manufacturing, automated packaging, sorting lines and pick-and-place systems[2].

Asynchronous motors play an equally important role in heating, ventilation and air conditioning (HVAC) systems. They provide the operation of fans, pumps and air handling units responsible for maintaining a comfortable microclimate in industrial and commercial premises. Modern HVAC systems are integrated with building management systems (BMS) and, thanks to this, allow you to dynamically adjust the speed of rotation of motors depending on the temperature, humidity and number of people in the room. This not only improves energy efficiency, but also contributes to more sustainable energy systems in large buildings. In the transport sector, in particular in electric transport, asynchronous motors occupy a leading position. They are actively used in electric vehicles, trams, trains, hybrid systems and even in modern air transport. Among their main advantages are high torque at low speeds, compact dimensions, and the ability to recover energy during braking. These characteristics have made induction motors the choice of companies like Tesla for their electric cars. In rail transport, they provide smooth acceleration and deceleration of trains, and are often integrated with energy-saving systems to feed electricity back into the grid.

One of the most important advantages of induction motors is their high reliability. The simplicity of the design, the absence of brushes and switches minimizes wear and tear and the need for maintenance. This makes them ideal for long continuous cycles of operation, especially in industries where even a short downtime can lead to significant financial losses. Their endurance to difficult operating conditions further enhances their value in heavy industry. Another important characteristic is energy efficiency. The efficiency of induction motors can reach 80–95% depending on the type and load mode. This is especially relevant in conditions of increased requirements for energy conservation and reduction of the carbon footprint. In large manufacturing enterprises, even a small increase in efficiency leads to a significant reduction in operating costs and energy consumption.

Economy is another important advantage. Asynchronous motors are cheaper to manufacture than other types of electric motors, have a wide range of power and size options, and have a long service life. This makes them particularly attractive to both large businesses and small businesses looking to automate production with minimal costs[3].

Thanks to modern control technologies, such as variable frequency drives (VFD), asynchronous motors can operate in variable speed mode. This allows the motor's operating parameters to be precisely tuned to specific process requirements, which increases accuracy, reduces wear, and optimizes energy consumption. For example, in pumping systems, this allows for real-time fluid flow control based on consumer needs.

In addition, integration with digital control systems - such as programmable logic controllers (PLC), human-machine interfaces (HMI), and SCADA platforms - make induction motors key elements of today's automated manufacturing systems. These technologies enable remote monitoring, diagnostics, fault prediction and real-time adaptive control. The development of artificial intelligence and machine learning opens up new opportunities - in particular in the field of predictive maintenance and optimization of operating parameters based on accumulated data[3].

1.3 Typical faults of induction motors and their consequences

Asynchronous motors are an integral part of industrial and commercial systems due to their simplicity, reliability, energy efficiency, and cost-effectiveness. They are widely used in drives for conveyor systems, pumps, fans, compressors, heating, ventilation, air conditioning (HVAC) systems, as well as in rail transport and electric vehicles. However, despite their strength and wear resistance, asynchronous motors are subject to various failures during operation.

These failures can be caused by mechanical wear, thermal overloads, electrical asymmetries, adverse environmental conditions, violations of maintenance regimes or design defects. If such problems are not detected in the early stages, they can lead to serious damage to the equipment, unexpected stoppages of the technological process,

significant economic losses or even emergency situations that threaten the safety of personnel.

The most common types of induction motor faults include:

• Worn or damaged bearings;
• Shaft misalignment or warping;
• Unbalance or rotor vibrations;
• Damage to the stator windings (short circuits, insulation breakdowns);
• Power supply problems (phase loss, voltage imbalance);
• Overheating due to overload or poor ventilation;
• Broken rods of the rotor;
• Insufficient or dirty lubricant in the bearing system.

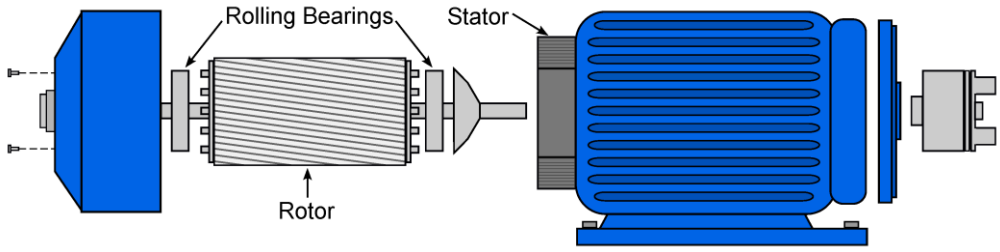


Fig. 1.4 Common fault locations in a standard induction motor.

Thermal imaging diagnostics: an effective and non-contact approach

Infrared thermography, or thermal imaging, is a promising diagnostic method that allows non-contact and rapid detection of temperature anomalies associated with potential faults. Thermal images provide the ability to visualize the temperature distribution over the surface of the motor, which allows identifying overheating zones, which are usually the result of internal defects[4].

Thus, thermal imaging diagnostics allows not only to detect existing problems, but also to carry out preventive maintenance, reducing the risk of critical breakdowns and downtime.

The problem of insufficient data for intelligent analysis

Although thermal images,which shown in fig 1.5, are extremely informative, their use in automated diagnostics systems based on artificial intelligence (for example, deep convolutional neural networks) faces a serious problem: the lack of high-quality annotated data.

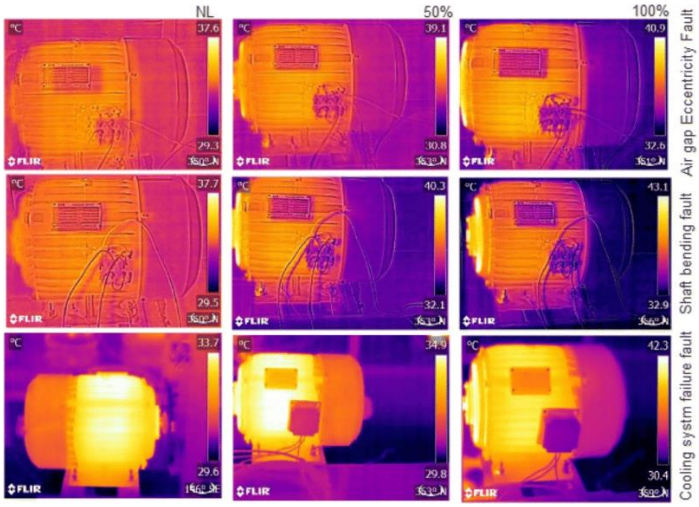


Fig. 1.5 Thermal image showing overheating in a faulty motor bearing.

As a result, a critical barrier arises: the insufficient amount of data makes it difficult to train neural networks for the tasks of classification and detection of defects using thermal images.

One of the modern solutions to this problem is the generation of artificial thermal images using generative models. This approach allows:

- Create new examples for rare classes of faults;
- Balance data sets;
- Improve the generalization ability of the model;
- Avoid the need for physical simulation of faults;
- Reduce the cost of lengthy experimental testing.

Thus, the use of generative models - in particular, generative adversarial networks (GAN) - becomes an effective tool for the synthesis of realistic thermal images of asynchronous motor faults, which opens new horizons in the construction of intelligent diagnostic systems.

1.4 Using thermal imaging to detect faults

Infrared cameras, also known as thermal imagers, are an extremely useful means of detecting faults in electric motors, as well as a tool for continuous monitoring of their condition in the context of preventive maintenance. They are widely used in the manufacturing, energy and commercial service sectors[1]. Thermal images obtained with the help of such cameras reflect the real state of engine operation through the temperature distribution on its surface. Such monitoring helps to prevent many sudden failures in critical systems for production. The onset of faults in an electric motor can be detected using various diagnostic methods, the most common of which are vibration analysis, ultrasonic inspection and thermal imaging scanning.

Electric motors are the basis of modern industry, as they provide the majority of mechanical processes. Thermal imaging cameras play a key role in both identifying specific problems and systematically monitoring the overall condition of equipment. They are particularly useful for long-term preventive maintenance. A handheld thermal imaging camera can capture infrared temperature measurements of the motor profile in a two-dimensional image (fig.1.4.1), allowing you to assess the condition of all critical components without having to disassemble the equipment.

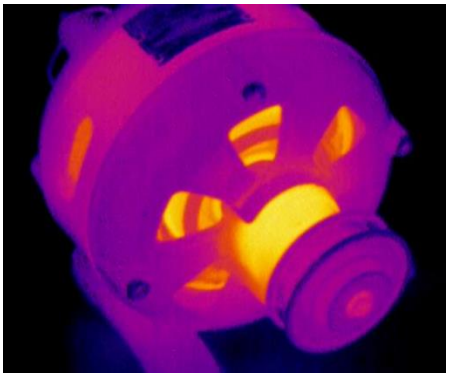


Fig. 1.6 Infrared image of an electric motor with a distinct temperature distribution in the shaft area.

Thermal imaging diagnostics have a number of advantages that make it an effective tool for monitoring the condition of electric motors[2]. It allows you to perform an inspection directly during the operation of the equipment, confirm the quality of the

performed repairs, perform an inspection from a safe distance, as well as significantly increase the efficiency of maintenance and reduce its cost. To create high-quality thermal profiles of engines, it is recommended to record infrared images under conditions of their normal operation. This allows you to form a basic (reference) temperature template, with which other measurements can be compared in the future. An infrared camera is able to determine the temperature of all key components of the motor: the housing, shaft coupling, motor and shaft bearings, and gearbox. In cases where the motor operates with a low electrical load, signs of problems may be insignificant. Therefore, the National Fire Protection Association (NFPA 70B) recommends performing a thermal inspection at a load of at least 40% of the nominal. The higher the load, the more pronounced the temperature differences will be, and the easier it will be to identify potential faults[4].

There are several key features to look for when conducting a thermal analysis. All electric motors have a normal operating temperature listed on the nameplate. Deviations from this temperature, as seen in the thermal image, can indicate a problem. For example, overheating of the motor housing can be the result of insufficient ventilation, which can sometimes be remedied by simply cleaning the ventilation grilles. Power quality problems, such as phase imbalance, overload or the presence of harmonics, can also lead to increased temperatures in certain areas of the motor. Excessive heating of bearings usually indicates that they are worn out or are approaching failure (fig 1.4.2).

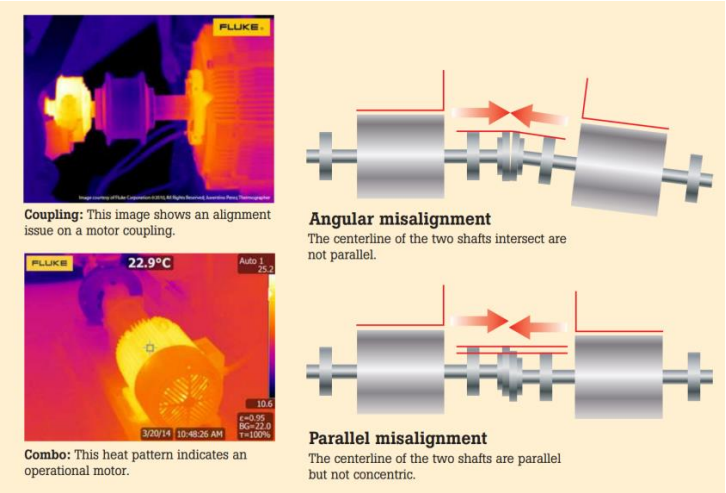


Fig. 1.7 Example of detecting shaft alignment problems based on thermal imaging: angular and parallel misalignment.

This situation occurs due to overloading, insufficient or incorrect lubrication, inefficient sealing, misalignment of the shaft or incorrect mounting of bearings.

In case of overheating, it is important to clearly establish its cause. If the problem is caused by poor ventilation, it is advisable to stop the motor to clean the grilles, and during the next scheduled maintenance, to perform a deep cleaning. If the source of overheating is overload or voltage imbalance, thermal imaging analysis will help to identify an overheated contact or high resistance in the switching equipment. In this case, a check with a multimeter, current clamps or power quality analyzer is required.

If the overheating is related to the bearing, you should create a request for its replacement or maintenance (lubrication). It is also advisable to perform a vibration analysis to clarify the extent of the damage. In case of insulation damage, it is recommended to reduce the load in accordance with NEMA standards and replace the motor at the earliest opportunity. If a misalignment of the shaft is suspected, the diagnosis should be confirmed by vibration analysis, and in the event of a possible stoppage, laser or indicator alignment should be performed.

Thus, thermal imaging diagnostics is an effective, non-contact and visual method of assessing the technical condition of asynchronous motors. It allows you to detect a wide range of faults at the early stages of their development, increasing the reliability of equipment operation and reducing repair costs. However, despite its high informativeness, the use of thermal images in more complex, automated diagnostic systems is faced with a number of limitations. The most important of them is the shortage of high-quality, annotated thermal data necessary for training modern artificial intelligence models. In real conditions, it is difficult to collect a large and diverse set of images covering all possible types of faults. It is this problem - the limitation of thermal data - that has become the basis for finding new approaches to solving it. One of such promising areas is the creation of synthetic thermal images using generative models. The next section will consider in more detail why this problem is critical for the implementation of intelligent diagnostics, as well as what approaches are used to overcome it.

1.5 Modern methods for diagnosing faults in induction motors based on images.

In modern conditions, industrial enterprises are increasingly focused not only on the stable operation of electric motors, but also on the implementation of intelligent diagnostic systems capable of quickly detecting potential malfunctions. One of the promising areas of such diagnostics is the analysis of visual information, in particular thermal images obtained using infrared cameras. This approach allows you to assess the condition of the induction motor without stopping the process, as well as to automate the detection of defects using computer vision algorithms. In response to the need for rapid detection of technical malfunctions in industrial asynchronous motors, numerous approaches to diagnostics have been proposed in the scientific literature, which are conventionally divided into invasive and non-invasive methods. Invasive methods are usually based on the analysis of electromagnetic parameters of the motor, such as current or torque. However, these approaches are difficult to implement in real production conditions, especially in the early stages of fault development, when signal fluctuations are small and difficult to interpret. In addition, invasive methods require the installation of specialized equipment, which increases the risk to personnel and the overall cost of implementation[4].

On the other hand, non-invasive methods of diagnosis, in particular thermographic analysis, show significant potential. Since most faults in induction motors are accompanied by increased heat generation, thermal images are a reliable indicator of the technical condition of the equipment. The main goal of such analysis is to record the heat distribution on the surface of the motor and detect anomalous patterns that indicate the presence of faults. The diagnostic process includes image pre-processing, region of interest (ROI) selection, feature extraction and classification. In classical approaches, the region of interest is defined manually - the operator selects the most "hot" points, and then the image is analyzed based on first and second order statistical characteristics (mean, variance, contrast, entropy, etc.). These features are fed to the input of traditional machine learning models, in particular multilayer perceptron (MLP), decision trees, or fuzzy logic systems.

However, such methods have several critical limitations:

• similar features may occur for different types of faults due to variability in external conditions;

• low robustness to noise or image artifacts;

• dependence on manual processing, which limits automation.

To overcome these problems, deep learning methods, such as convolutional neural networks (CNN) (fig.1.8), have been proposed in scientific research. These models have the ability to automatically detect relevant features from an image without the need for manual ROI extraction or feature construction. The CNN receives an image as input, and during training, it independently learns to distinguish normal engine states from faulty ones, taking into account spatial patterns of temperature distribution.

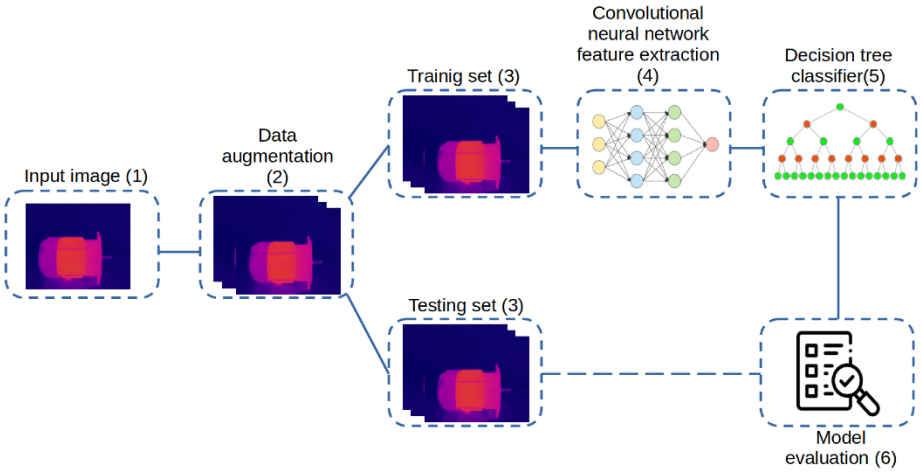


Fig. 1.8 General scheme of using a convolutional neural network for thermal image classification.

In some works, CNN are combined with additional classifiers. For example, one model involves using CNN to pre-extract features from a thermal image, after which these features are fed to the input of a decision tree or SVM (support vector machine). This combination provides higher accuracy by combining the flexibility of CNN with the structure of traditional models. In addition, input image enhancement methods such as CLAHE (Contrast Constrained Adaptive Histogram Equalization) are used to improve the quality of the features in the early stages of processing.

Серед успішно протестованих архітектур можна виділити:

- ResNet50 - глибока мережа з резидентними з’єднаннями;
- InceptionV3 з SE-модулями - для фокусування уваги моделі на найважливіших ділянках зображення;
- EfficientNetB0 - оптимізована модель з високою продуктивністю та низькою кількістю параметрів;
- GLCM, wavelet, PCA - як доповнення для формування векторів ознак у гібридних підходах;
- KNN, Random Forest, нейро-нечіткі системи - як додаткові блоки класифікації.

As a result of many studies, it has been found that the accuracy of engine state classification using CNN exceeds 95%, and the robustness of models to noise and environmental changes increases by 1.5–2 times compared to traditional methods[5].

However, despite these advantages, CNN and similar models have a common critical dependency - the need for large and balanced sets of thermal images. In practice, this is a significant challenge: annotated thermal images are difficult to collect, failures occur irregularly, and artificially creating defects can be expensive or dangerous.

That is why generative models, in particular Generative Adversarial Networks (GAN) (fig.1.9), are gaining increasing popularity. They allow generating synthetic thermal images that retain the visual characteristics of real defects, and thereby provide expansion of training datasets without risk to equipment or personnel.

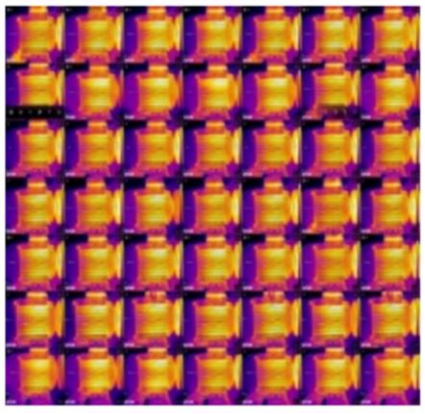


Fig. 1.9 Example of synthetic thermal images of an induction motor generated using a GAN-type model to augment the training dataset.

Modern methods for diagnosing faults in induction motors are increasingly based on the analysis of visual data, in particular thermal images. Thanks to the use of computer vision algorithms, and especially deep convolutional neural networks, it has become possible to automate the detection of critical engine defects in the early stages without the need to stop the equipment.

The main advantages of such methods are high accuracy, noise immunity, self-learning ability and minimization of the human factor. However, the effectiveness of these technologies largely depends on the quality and quantity of training data, which is a significant problem in real production conditions.

In this regard, an important innovation in the field of technical diagnostics has been the use of generative neural networks. In particular, models such as WGAN-GP (Wasserstein GAN with Gradient Penalty) allow generating realistic thermal images that can be used as additional data for training fault detection systems. The following section will consider the features of the operation of GAN architectures, as well as an example of using WGAN-GP to generate synthetic thermal maps of induction motors[5].

Conclusions to Chapter I

This section reviewed the main technical and operational aspects of induction motors, which are the main drives in modern industry. Their operating principle, design features, and advantages that ensure their reliability, energy efficiency, and a wide range of applications, from automated production lines to electric vehicles, were analyzed in detail.

Particular attention was paid to typical faults that occur during the operation of induction motors, including bearing wear, shaft misalignment, winding damage, overload, and overheating.

Modern approaches to image-based motor condition monitoring were analyzed, including both traditional (with manual feature extraction) and deep learning methods, including convolutional neural networks. Separately, a key problem is emphasized - the shortage of high-quality and balanced thermal images for training neural networks.

In response to this problem, the feasibility of using generative adversarial networks is substantiated. (Generative Adversarial Networks, GAN), in particular, architectures such as WGAN-GP and cWGAN-GP. They are capable of synthesizing realistic thermal images for various types of faults, which allows expanding training samples without physically creating defects, minimizing risks and costs.

Thus, this section has formed a theoretical basis for further research into the application of generative neural networks in technical diagnostics tasks. The next section will discuss in detail the principles of GAN operation, as well as the implementation of the WGAN-GP approach for generating thermal images of induction motors in different states.

CHAPTER II
OVERVIEW OF GENERATIVE ADVERTISEMENT NETWORKS AND PROBLEM STATEMENT

2.1. Generative Adversarial Neural Networks (GAN): Working Principle and Basic Concepts

Generative adversarial networks are one of the varieties of generative models. The term "generative model" is used in different contexts in different ways. In the framework of this work, a generative model is understood as a model that, based on a training data set consisting of samples drawn from some real distribution $p_{data}(x)$, learns to represent the estimate of this distribution in a certain way. As a result of this process, a model probability distribution is formed $p_{model}(x)$. In some cases, the model explicitly estimates $p_{model}(x)$, while in other cases the model is only capable of generating samples from this distribution. Some models can perform both tasks. Generative adversarial networks (GAN) are mostly focused on generating samples, although there are modifications of GAN that also allow explicit estimation of the model distribution. At this point, it is appropriate to ask the question: how do generative models work in general and how do GAN compare to other approaches to generative modeling? In particular, it is important to consider one of the basic principles underlying most generative models - the principle of maximum likelihood estimation.

To simplify the discussion, we will first focus on generative models that are directly based on this principle. It should be noted that not all generative models use maximum likelihood estimation by default. Some of them (like GAN) do not use this principle directly, but can be adapted to implement it. The use of maximum likelihood allows us to clearly formulate the problem of building a generative model, as well as to compare GAN with other models that are directly based on this principle[6].

The basic idea of maximum likelihood is to choose the following model parameters θ, which maximize the probability that the model will generate exactly the training data that was actually observed. Formally, if given a set of training examples $\{ x^{(1)},x^{(2)},...,x^{(m)}\}$, obtained from the distribution $p_{data}(x)$, then the likelihood of the model is defined as:

$$L(\theta) = \prod_{i=1}^{m} p_{model}(x^{(i)}; \theta) \tag{2.1}$$

To simplify computational procedures, log-likelihood is usually used, which converts the product of probabilities into a sum of logarithms:

$$\log L(\theta) = \sum_{i=1}^{m} \log p_{model}(x^{(i)}; \theta) \tag{2.2}$$

Using log-likelihood avoids the numerical problems associated with multiplying a large number of very small numbers, which often arise when working with real data sets. Thus, the task of training the model is reduced to finding parameters that maximize the log-likelihood[6]:

Another view of the maximum likelihood principle allows us to interpret it as minimizing the Kullback–Leibler divergence (KL-divergence) between the real data distribution $p_{data}(x)$ and the model distribution $p_{model}(x;\theta)$:

$$KL(p_{data}||p_{model}) = \int p_{data}(x) \log \frac{p_{data}(x)}{p_{model}(x;\theta)} dx \tag{2.3}$$

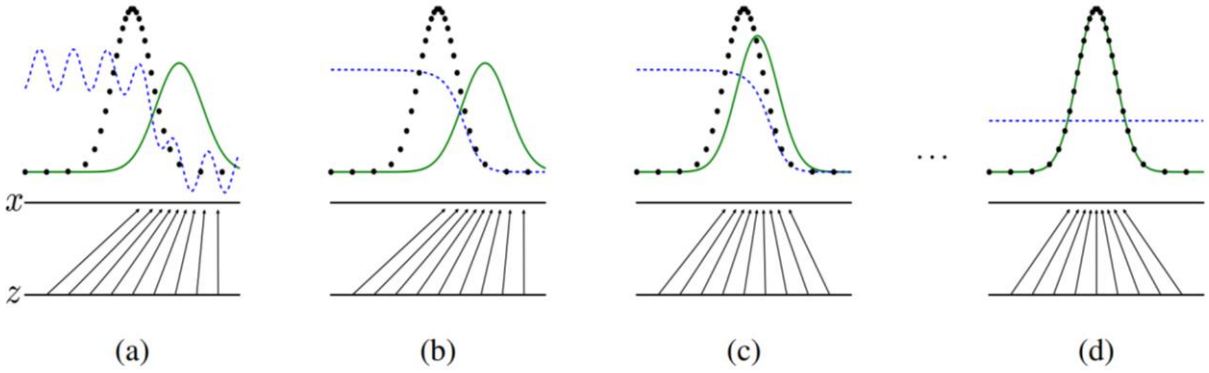


Fig 2.1 Schematic illustration of the process of approximating a model distribution to the real data distribution by training a generative model.

Thus, by minimizing this discrepancy, the model approximates its distribution to the true distribution of the data, which is one of the fundamental goals of generative modeling.

Although the principle of maximum likelihood is a powerful tool, it cannot always be used directly. In certain cases, generative models, which are based on maximum likelihood,

face serious computational and analytical difficulties. To avoid the problems associated with the need to explicitly define and calculate probability densities as precisely as possible, alternative approaches have been developed that use either deterministic approximations or stochastic methods. Deterministic approaches mainly include variational methods, while stochastic approaches include Markov chain Monte Carlo methods.

One of the most well-known deterministic variational methods is the variational autoencoder (VAE). The variational autoencoder creates a lower bound for the log-likelihood that is guaranteed to be less than or equal to the true log-likelihood. In the process of training the model, this lower limit is maximized, which allows avoiding complex calculations associated with explicit estimation of the density. The main advantage of this approach is that the lower bound can be computed analytically or effectively approximated. However, the main disadvantage of variational autoencoders is that if the approximating posterior or prior distribution is chosen to be too weak, then even with perfect optimization and an infinite amount of training data, the gap between the lower bound and the true log-likelihood can be significant. This can lead to the model distribution $p_{model}(x)$ will differ from the real distribution $p_{data}(x)$, which will result in lower quality data being generated.

It is in response to these shortcomings that generative adversarial networks (GAN) were proposed. GAN were specifically designed to be unbiased in approximating the true data distribution. That is, theoretically, given sufficient model capacity and an infinite set of training data, the Nash equilibrium in the game between the GAN generator and discriminator corresponds to an exact reproduction of the true data distribution $p_{data}(x)$.

In general, GAN demonstrate a greater ability to generate high-quality samples compared to variational autoencoders. Despite the lack of unambiguous methods for quantitatively measuring the quality of the generated samples, general empirical experience suggests that GAN generate more realistic samples that are difficult to distinguish from real data. However, from a practical point of view, GAN do not simplify optimization compared to VAE, since training GAN can be quite difficult due to the instability of the game

between the generator and the discriminator, which is a well-known problem of GAN that is actively researched.

How do generative adversarial networks work? The main idea of a GAN is to organize the interaction of two networks in the form of a game: one network (the generator, G) generates samples that are as similar as possible to real data, and the other (the discriminator, D) tries to determine whether the obtained samples are real or artificially generated[7].

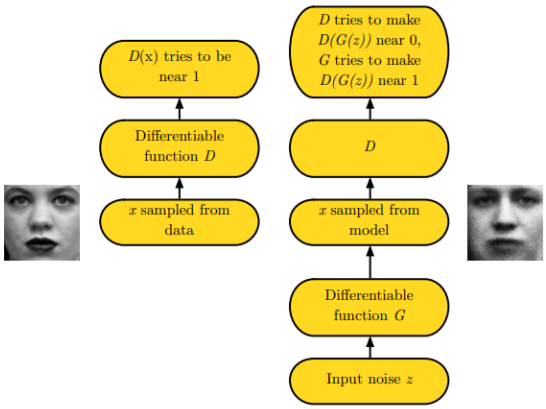


Fig. 2.2 General diagram of the interaction between the generator and the discriminator in a generative adversarial network (GAN).

The discriminator is trained in a standard supervised learning paradigm, dividing the input data into two classes: genuine (real) and spurious (generated). The generator, on the other hand, is trained in such a way as to "fool" the discriminator as much as possible by generating data that it will perceive as genuine. This process can be intuitively represented as a competition between a counterfeiter (generator), who tries to create perfect counterfeits, and the police (discriminator), who try to distinguish genuine from counterfeit money. To successfully achieve Nash equilibrium, the generator must learn to create samples that are indistinguishable from the real ones, and the discriminator must become an ideal classifier, which, however, when the ideal equilibrium is reached, is no longer able to distinguish the real samples from the generated ones, because they become exactly the same in distribution.

Formally, GAN are structured probabilistic models that include hidden variables Z (the generator input noise) and observed variables X (the data). Each of the two networks in the game (the generator G and the discriminator D) is described by its own differentiable function with corresponding parameters θ_G θ_D. They have separate cost functions that depend on the parameters of both networks, but each network controls only its own parameters. The discriminator's task is to minimize its loss function. $J^{(D)}(\theta_D,\theta_G)$, changing only the parameters θ_D , and the generator - minimize its own loss function $J^{(G)}(\theta_D,\theta_G)$, changing only the parameters θ_G . Since the parameters of each network affect the other, this competition is better viewed as a game than as a simple optimization problem. The optimal solution to this game is a Nash equilibrium - a state in which neither party can improve its position without changing the parameters of the other party.

It is worth dwelling in more detail on the description of each of the components of generative competitive networks, namely the generator and the discriminator. The generator G is a differentiable function, usually implemented as a deep neural network, which takes as input a random noise z, selected from some simple prior distribution, for example, uniform or Gaussian. This random vector z is transformed by a generator function into a data sample x, which must belong to the model distribution pmodel(x). Formally, this can be written as $x = G(z; \theta_G)$, де θ_G - are the generator parameters. There are no strict restrictions on the generator architecture, except for the need for the function to be differentiable and for the dimensionality of the output data to match the desired format. Different approaches to integrating noise into a neural network are possible: noise can be fed to the input of the first layer of the neural network or added at intermediate levels, which allows for different levels of detail and realism of the generated samples. On the other hand, the discriminator D is also a differentiable function that takes as input samples x, which can be either real or generated, and estimates the probability of their authenticity. The discriminator tries to classify the received samples as accurately as possible, assigning them a probability of belonging to the real data distribution. Formally, this is written as $D(x; \theta_D)$, де θ_D are the discriminator parameters.

The GAN training process consists of simultaneous steps of stochastic gradient descent for the generator and the discriminator. At each training iteration, two mini-groups (minibatches) are selected: one of them consists of real data samples x, and the second of artificially created samples G(z), generated from the input noise z. The discriminator parameters are updated in such a way as to minimize its loss function, which, in particular, reflects the ability of the discriminator to distinguish real and artificial samples. The generator parameters are updated in order to minimize its own loss function, which depends on the ability to “fool” the discriminator - that is, to make it classify the generated samples as real. In this case, the training process is carried out alternately: the discriminator and the generator perform gradient descent steps simultaneously or alternately, depending on the chosen training protocol. As a rule, gradient-based optimization algorithms are used, for example, Adam.

Thus, the game that takes place between the generator and the discriminator can be described using the following minimax function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.4)$$

This equation expresses the fact that the discriminator seeks to maximize the probability of correctly classifying real and artificial samples, while the generator seeks to minimize the probability that the discriminator correctly recognizes its samples as fake. In practice, the Nash equilibrium, which is the theoretical ideal for GANs, is only approximately achieved due to the complexity of optimization and the possibility of unstable model behavior. One common problem is the situation when the discriminator becomes "too good" and easily recognizes artificial samples in the early stages of training, which causes the gradients obtained by the generator to become very small, which leads to a slowdown or complete stop of the generator's training. To avoid this phenomenon, modifications of the loss functions are used in practice, such as replacing the term $\log(1-D(G(z)))$ на $-\log(D(G(z)))$, which provides a stronger gradient in the initial stages of training.

In addition, the implementation of GAN requires careful selection and balancing of the capacities (number of parameters) of the generator and discriminator, as well as taking into account the difficulties of synchronizing their training. If the generator is trained much faster or slower than the discriminator, this can lead to model degradation and poor sample quality. Therefore, it is usually recommended to use approximately equal model complexity for both networks, although in practice it is sometimes recommended to do a few additional steps of discriminator training for each generator training step to maintain some balance between the networks. Despite these difficulties, GAN have significant advantages over many other generative modeling methods. The main advantage of GAN is that gradient extraction and model training do not require approximate inference procedures or Markov chains, which are often used in other generative approaches (such as deep Boltzmann machines). This allows GAN to generate highly realistic samples more efficiently and achieve better results with fewer resources and simpler algorithmic solutions[7].

It is important to emphasize a few more features of generative-competitive networks that favorably distinguish them from other popular approaches in the field of generative modeling. In particular, GAN do not require an explicit representation of the density of the model distribution. $p_{model}(x)$. This is a significant advantage, as it allows for more flexible models that can model more complex and even degenerate (clear, concentrated) distributions that are typical in real-world applications. In other methods, such as deep Boltzmann machines, the distribution must be sufficiently "fuzzy" or regularized to ensure the efficient operation of Markov chain algorithms. GAN do not have this limitation, so they are often used to generate data that requires a high degree of detail and realism, for example in computer vision, medical imaging, voice synthesis and other artificial intelligence tasks. Another significant advantage of GAN is that during training, the generator does not receive direct access to real data, but only the gradients obtained through the discriminator. This prevents direct copying of training examples into the generator parameters, which avoids overfitting and provides a greater generalization ability of the model. Due to this, GAN have the ability to generate new samples that, although

similar to real ones, are not exact copies of them, which is an important advantage in many practical applications.

However, the disadvantages of GAN should also be taken into account. The main difficulty in using generative adversarial networks is the instability of learning, which arises due to the complex interaction of the generator and the discriminator. Often there are problems such as mode collapse (mode collapse), when the generator starts to create a limited set of very similar samples, ignoring the full diversity of the distribution of real data. To overcome this problem, special modifications of GAN have been developed, such as Wasserstein GAN (WGAN) and other variants that stabilize learning and help avoid mode collapse. These methods will be discussed in detail in the following sections of this work. In addition, another drawback is the need to carefully tune the hyperparameters of the model, including the learning speed, network architecture, number of gradient descent steps for the generator and discriminator, which makes the practical application of GAN difficult, especially for users who do not have deep experience in neural network setup.

Thus, generative adversarial networks are a powerful and promising class of generative models that combine a number of important advantages, such as high realism of generated samples, flexibility of architecture, and the absence of the need for an explicit representation of the probability density. Despite the difficulties in training and instability arising from the interaction of the generator and discriminator, GAN remain one of the most effective and popular tools in modern research and practical tasks of artificial intelligence. The following sections of this work are devoted to a detailed review of modern modifications of GAN that allow stabilizing the learning process and improving the quality of generation, as well as an analysis of the possibilities of their application in the tasks of diagnosing the technical condition of equipment based on thermal images.

2.2 Improving the Stability of GAN Learning: Wasserstein GAN and Overcoming Deficiencies Techniques

Generative adversarial networks (GANs) are a powerful class of generative models that transform the problem of generative modeling into a game between two networks: a generator network creates synthetic data from a noise source, while a discriminator network

distinguishes between the data generated by the generator and real data from a training set. GANs are capable of generating visually appealing samples, but they are often difficult to train due to the instability of the optimization process.

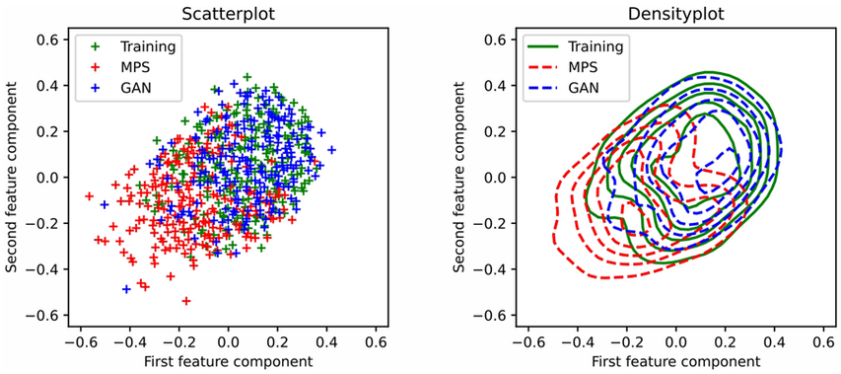


Fig. 2.3 Comparison of the distribution of real and generated data in classical GANs.

Much of the current research has been devoted to the development of methods for stabilizing learning, but ensuring stable learning of GAN remains an open problem.

In particular, some papers analyze the convergence properties of the cost function that is optimized by a classical GAN. As an alternative, Wasserstein GAN (WGAN) has been proposed, which uses the Wasserstein distance to construct a cost function with better theoretical properties than that of a conventional GAN. In WGAN, the discriminator (called the critic in this model) must be restricted to the space of 1-Lipschitz functions, and this restriction is implemented through weight clipping.

In the proposed Wasserstein GAN model, the main contributions are:

1. Demonstration on simple datasets how critical weight constraints can cause undesirable behavior.

2. Proposal of a Gradient Penalty (WGAN-GP) method that eliminates the problems associated with weight clipping.

3. Empirical confirmation of stable learning of different GAN architectures, performance improvement compared to weight pruning, generation of high-quality images, and construction of a speech GAN model at the symbol level without a discrete sampling process.

It is known that the divergences that are usually minimized in GAN may not be continuous everywhere with respect to the generator parameters. This leads to difficulties during training. Instead, Wasserstein GAN propose to use the distance W(p,q) - the so-called "earth-mover distance", also known as Wasserstein-1 distance. Its informal definition is to minimize the cost of transporting mass to transform one distribution into another, where the cost is calculated as the product of mass and transportation distance. Under mild assumptions, the distance W(p,q) is continuous and almost everywhere differentiable.

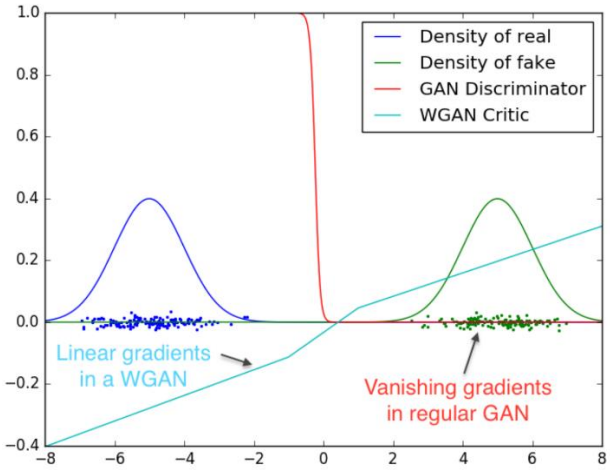


Fig 2.4 Comparison of gradients in classical GAN and Wasserstein GAN.

The WGAN cost function is based on the Kantorovich–Rubinstein duality and has the form:

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p(z)}[D(G(z))] \tag{2.5}$$

where:

• pr is the distribution of the true data,

• p(z) is the distribution of random noise,

• G(z) is the generator,

• D(x) is the critic,

• D is the set of 1-Lipschitz functions.

With an optimal critic (discriminator), the minimization of this cost function with respect to the parameters of the generator is equivalent to the minimization of the Wasserstein distance between the distribution of real data and the model distribution of the generator.

Although the concept of Wasserstein GAN, which is based on the use of Wasserstein distance as a loss functional, theoretically significantly improves the stability of generative learning, in practice the implementation of the Lipschitz continuity constraint turned out to be a difficult task. In particular, there was a need to ensure that the critic (discriminator) belongs to the class of 1-Lipschitz functions, that is, such functions for which the norm of the gradient is bounded by unity everywhere in the input space[10].

In the original version of WGAN, it was suggested to apply a simple technique - weight clipping. The essence of the method was that after each update of the parameters of the critic neural network using the gradient descent algorithm, all weight coefficients were limited to a fixed interval [−c, c]. Formally it looks like this:

$$w \leftarrow \text{clip}(w, -c, c)$$

where w is the network weight,

c is a small predetermined constant, for example c =0.01.

The purpose of this restriction is to prevent the network parameter values from growing too large, which provides control over the Lipschitz constant of the critic function. The use of weight pruning allows us to formally satisfy the requirements of Kantorovich-Rubinstein duality, which is necessary for the correct construction of the Wasserstein GAN functional.

However, as further theoretical and empirical studies have shown, the weight trimming technique has a number of serious drawbacks that limit its application in practical conditions.

First, **weight pruning leads to a limitation of the power of the neural network.** By reducing the allowable space of weight coefficients, the restriction [−c, c] reduces the critic's ability to approximate complex functional dependencies. As a result, even under

ideal optimization conditions, the critic may not approximate the optimal discriminant function accurately enough, which worsens the overall quality of generator training.

Second, if the value of the constant ccc is chosen incorrectly, for example, if it is too small, the critic neural network becomes too weak. It ceases to reliably distinguish between real and generated data, which leads to gradient decay. In this case, the generator receives a signal that is too weak or even useless for updating its parameters.

Thirdly, **an excessively large value of the truncation parameter leads** to the fact that the model loses its Lipschitz boundedness. In this case, the loss function no longer guarantees the theoretical convergence properties, and the entire concept of Wasserstein GAN loses its advantage over the classical GAN.

Moreover, the application of weight pruning becomes difficult when moving to more complex network architectures. For example, in deep convolutional neural networks or recurrent networks, simply pruning the weights after each optimization step has been shown to be insufficient to ensure the required smoothness of the critic function. Under these conditions, even small errors in weight constraints lead to a significant deterioration in the stability of training[8].

In addition, empirical observations have shown that weight pruning can negatively affect the behavior of the optimizer. For example, using optimizers such as RMSProp or Adam in conditions of constant weight trimming leads to the fact that the optimization process becomes less predictable, which further complicates the achievement of game balance between the generator and the critic.

Taken together, these factors have forced researchers to look for more efficient methods for ensuring Lipschitz boundedness without interfering with the weight space, which led to the development of a new technique known as **WGAN with gradient penalty (WGAN-GP),** which will be discussed in the next section.

Як було зазначено раніше, техніка обрізання ваг, застосована в класичному WGAN, має численні недоліки. Вона обмежує апроксимуючу здатність критика і негативно впливає на якість градієнтів для оновлення генератора. Для подолання цих проблем було запропоновано новий підхід - **WGAN з градієнтним штрафом**

(WGAN-GP), which retains the main advantages of the Wasserstein GAN but eliminates the need for weight pruning.

The main idea of WGAN-GP is that the 1-Lippschitzity of a function can be ensured not by limiting the parameters of the model, but by controlling the gradient norms of the output function with respect to its input data. In this approach, it is not necessary to change the weights of the neural network after each optimization step. Instead, an additional penalty term is introduced into the cost function, which punishes the critic for deviations of its gradient norm from unity[9].

The loss function for WGAN-GP has the following form:

$$L = \mathbb{E}_{\tilde{x}\sim p_g}[D(\tilde{x})] - \mathbb{E}_{x\sim p_r}[D(x)] + \lambda \mathbb{E}_{\hat{x}\sim p_{\hat{x}}} \left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2 \quad (2.6)$$

where:

• x∼∼pg are the samples generated by the generator;

• x∼pr are the real data samples;

• x^∼px are the points formed by uniformly selecting points along straight lines between pairs of real and generated samples;

• λ is the regularization coefficient (set to 10 in the basic experiments).

Penalty term:

$$\lambda \mathbb{E}_{\hat{x}\sim p_{\hat{x}}} \left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2 \quad (2.7)$$

provides a "soft" compliance with the 1-Lipschitz condition: instead of a strict requirement that the gradient be bounded strictly from below, the model is only encouraged to maintain a gradient norm approximately equal to unity.

In the sample generation procedure, the xdistribution px is defined as a uniform distribution along the segments between the points $x\sim p_r$ i $x\sim p_g$. Intuitively, this means that we generate new points along trajectories that directly connect the real and generated samples, and control the behavior of the critic precisely in these intervals.

This choice is justified by the fact that the optimal critic has gradients of a norm close to unity along the straight lines connecting the corresponding points of the real and model distributions (confirmed in a separate theoretical proposition in the source).

It is important to note that in WGAN-GP:

• a two-sided penalty function is used, i.e. deviations both in the larger and smaller direction from the norm 1 are punished equally;

• the selected value of the penalty coefficient $\lambda=10$ has shown good results for a wide range of tasks, both in synthetic datasets and on large sets such as ImageNet.

Another important aspect is the interaction with normalization layers in neural networks. Since in WGAN-GP the penalty is imposed on the gradient for each individual sample, the use of batch normalization (batch normalization) has been criticized as contradicting the principle of the model, as it changes the output calculation depending on the composition of the batch. Therefore, when working with WGAN-GP, it is recommended to either abandon normalization in criticism altogether, or use other normalization methods, such as layer normalization, which does not change the dependence of the result on other examples of the batch. The proposed gradient penalty experimentally demonstrated:

• high stability of training,

• improved quality of generated samples,

• complete absence of gradient decay problem.

Thus, the WGAN-GP method is a reliable and effective improvement of the classical Wasserstein GAN and is widely used today in practical problems of deep generative modeling.

The Wasserstein GAN training algorithm involves the construction of an iterative process in which several steps of optimization of the critic (discriminator) and one step of optimization of the generator alternate. The goal of such a training organization is to achieve an approximation of the critic to the optimal discriminant function for the current state of the generator.

At each iteration of the algorithm, initially n_{critic} updates of the critic parameters. For each update, two mini-batches are generated: the first consists of real data samples selected from the distribution p_r, the second one is from random noise vectors obtained from the distribution p_z. Implementing the generator on these random inputs provides synthetic samples that are input to the critic alongside the real samples.

The critic's cost function is defined as the difference between the average values of its outputs on real and synthetic data. To minimize it, the stochastic gradient is calculated over the critic's parameters w according to the following formula:

$$g_w = \nabla_w \left(\frac{1}{m} \sum_{i=1}^{m} \left[D_w(x^{(i)}) - D_w(G_\theta(z^{(i)})) \right] \right) \tag{2.8}$$

The critic parameters are updated using the RMSProp optimizer with a learning rate α:

$$w \leftarrow w + \alpha \cdot \mathrm{RMSProp}(g_w) \tag{2.9}$$

After each update, the operation of trimming the weights of the critic to the interval [−c, c] is performed, which allows ensuring the Lipschitz boundedness of the critic function:

After completion n_{critic}critic updates, one step of updating the generator parameters is performed. To do this, a new mini-batch of random vectors is selected $\{z^{(i)}\}$, which are fed to the generator input to form synthetic samples.

The loss function for the generator is defined as the negative average value of the critic output on the generated samples, which is formalized by the expression:

$$g_\theta = \nabla_\theta \left(-\frac{1}{m} \sum_{i=1}^{m} D_w(G_\theta(z^{(i)})) \right) \tag{2.10}$$

The generator parameters are updated using the same stochastic gradient descent scheme using the RMSProp optimizer:

$$\theta \leftarrow \theta - \alpha \cdot \mathrm{RMSProp}(g_\theta) \tag{2.11}$$

The generator and critic update process alternates until convergence is achieved. All hyperparameters of the algorithm, including the learning rate α, the batch size m, the weight trimming parameter ccc, and the number of critic updates ncritic, must be chosen in advance. In the initial experiments with the standard WGAN, the following parameter values were used: α=0,00005, c=0,01, m=64, n_{critic}=5[12].

It is important to note that the approach with multiple critic update steps before each generator update step is essential to balance the game between the two networks and ensure a correct approximation of the Wasserstein distance between the real data distribution and the distribution generated by the generator.

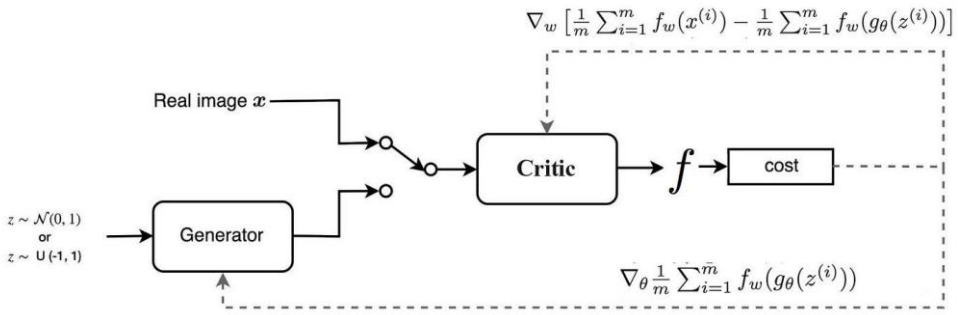


Fig. 2.5 Schematic of the Wasserstein GAN learning process.

Problems associated with the application of the technique of truncation of scales to ensure 1-Lipschitzness led to the need to improve the approach. The development of the WGAN-GP model using a gradient penalty allowed us to preserve the Lipschitz boundedness of the critic without interfering with the weight space, which significantly improved the stability of the optimization and the quality of the generated data.

The use of WGAN-GP has shown high efficiency for complex architectures of neural networks and large data sets, which makes this approach one of the most promising for the further development of generative modeling. Thus, the transition to Wasserstein GAN and WGAN-GP allows to significantly increase the stability of the learning process of generative models and ensure the generation of high-quality data, which is critically important for the practical application of generative neural networks in modern tasks of artificial intelligence.

This section considered the main problems that arise when training generative adversarial networks, as well as modern methods for overcoming them based on the use of Wasserstein GAN and its modifications. The use of an alternative loss functional and the introduction of a gradient penalty allowed us to significantly improve the stability of generator optimization, ensure higher-quality data generation, and expand the possibilities of using GANs in complex practical tasks.

Further improvement of the efficiency of generative models largely depends on the features of their architecture. In the next section, the topology of generative adversarial networks will be considered in detail, which allows us to better understand how the structure of models affects the quality and stability of learning[13].

2.3 Topology of generative and competitive networks

Generative-competitive networks (GANs) consist of two main components: a generator and a critic (in classical models, a discriminator), which interact in the process of two-way learning. A generator aims to create synthetic data that is difficult to distinguish from real data, while a critic tries to accurately classify data as real or artificially generated.

In the WGAN-GP model, the architecture of both networks is specifically adapted to improve the stability of learning by minimizing the Wasserstein distance instead of the classical divergences and introducing a penalty on the gradient norm.

An important feature of the WGAN-GP construction is the use of a loss functional that includes a gradient penalty that controls the Lipschitz boundedness of the critic without the need for hard weight pruning. This avoids problems with exploding or vanishing gradients during training.

The general scheme of such a model involves a competitive interaction: the generator transforms random vectors into data of a certain structure, while the critic evaluates both real samples from the training sample and generated examples. Both networks are trained simultaneously using specially defined loss functions. The structural scheme of the interaction of the generator and the critic in WGAN-GP is presented in Figure 2.3.1.

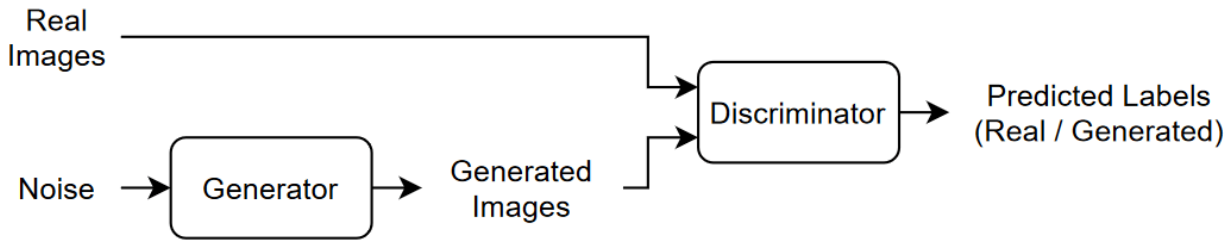


Fig. 2.6 Block diagram of the interaction between the generator and the critic in WGAN-GP.

The architecture of the generator in WGAN-GP is designed to provide a gradual increase in the spatial resolution of data samples from a random input vector to a full-fledged image. Initially, the generator accepts as input a random vector of dimension 1×1×100, which goes through the operation of projection and transformation into a tensor of size 4×4×512.

Further, the data dimensionality is gradually scaled through a series of transposed convolution layers, which provide an increase in spatial dimensions and a decrease in the number of channels. Each transposed convolution layer is accompanied by an activation of the ReLU type, which contributes to the stable transmission of positive gradients through the network[8].

At the output of the generator, a transposed convolution layer is used, which forms the final image of size 64×64×3 (three channels for an RGB color image), after which the tanh activation function is applied. The choice of tanh is due to the need to normalize the initial data in the range [−1,1], which corresponds to the previous normalization of the real images in the training sample [13].

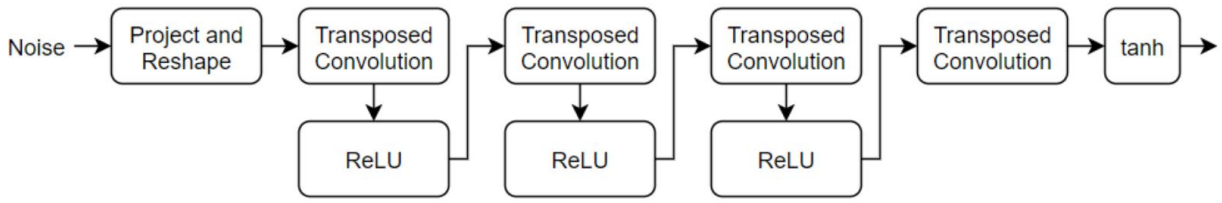


Fig. 2.7 Generator architecture in the WGAN-GP model.

The structure of the layer-by-layer architecture of the generator is given in Table 2.1.

Table 2.1

WGAN-GP generator architecture

№	Layer	Parameters	Output data size
1	Input Noise Vector	100 elements	(1 × 1 × 100)
2	Projection and Transformation (Fully Connected)	4×4×512 projection	(4 × 4 × 512)
3	Transposed Convolution	5×5 filters, 256 channels, stride=2	(8 × 8 × 256)
4	ReLU	-	(8 × 8 × 256)
5	Transposed Convolution	5×5 filters, 128 channels, stride=2	(16 × 16 × 128)
6	ReLU	-	(16 × 16 × 128)
7	Transposed Convolution	5×5 filters, 64 channels, stride=2	(32 × 32 × 64)
8	ReLU	-	(64 × 64 × 3)
9	Transposed Convolution	5×5 filters, 3 channels, stride=2	(64 × 64 × 3)
10	tanh	Output normalization [-1, 1]	Output data size

The use of an architecture that gradually expands the spatial resolution of the data ensures high-quality reproduction of details in the generated images, which is especially important for generative modeling tasks.

Unlike the generator, the critic in WGAN-GP has an architecture that aims to gradually reduce the spatial resolution of the input samples and calculate their fit to the true data distribution. The input data for the critic are either real images from the training set or synthetic samples generated by the generator, both of 64×64×3 format.

The main goal of the critic is to maximize the difference between the mean values of its estimates for real and generated data, which accordingly approximates the calculation of the Wasserstein distance. For this, it is necessary to ensure the correct behavior of network gradients, which directly depends on the chosen topology of the critic.

The WGAN-GP critic uses a series of ordinary convolutional layers with stride=2 to gradually reduce the dimensionality of the input data. After each convolutional layer, the LeakyReLU activation function with a negative slope parameter of 0.2 is applied. This activation function is chosen instead of the standard ReLU to avoid the effect of "dead neurons" and to ensure a constant flow of gradients even in cases of weak activation [3].

It is known that the problem of "dead neurons" is especially critical in generative modeling tasks, where the loss of activity of even a part of the neurons can significantly degrade the quality of model training.

After some convolutional layers, the critic applies Layer Normalization, which replaces the classical Batch Normalization. As stated in the work of Gulrajani et al. [12], the use of BatchNorm can lead to incorrect calculations of the gradient norm penalty, as it creates dependencies between the processing of different elements in the batch. LayerNorm or the complete absence of normalization avoids these problems and ensures that individual gradients are calculated for each example independently. In the last step, the critic has a convolutional layer with a 4×4 filter that produces a single scalar output for each input sample. This estimator does not pass through any nonlinear activation function, other than applying a sigmoid function in some implementations to restrict the output value to the range [0,1]. The detailed architecture of the WGAN-GP critic is presented in Figure 2.3.2.

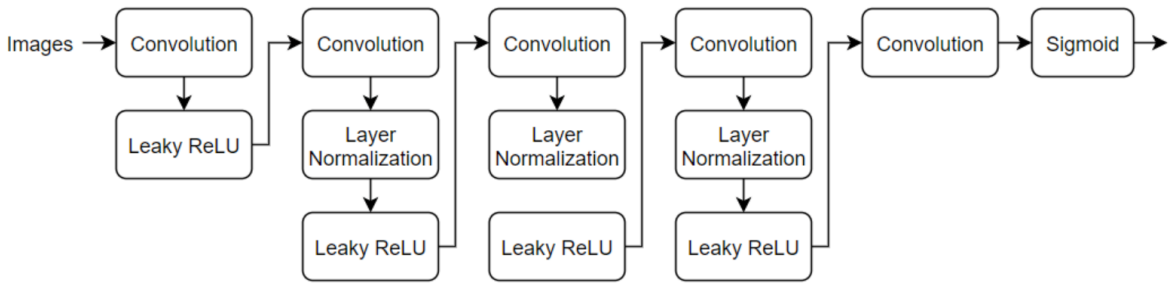


Fig. 2.8 Discriminator architecture in the WGAN-GP model.

The general layered structure of the critic is shown in Table 2.2.

Table 2.2

WGAN-GP Criticism Architecture

№	Layer	Parameters	Output data size
1	Input image	64×64×3	(64 × 64 × 3)
2	Convolution	5×5 filters, 64 channels, stride=2	(32 × 32 × 64)
3	LeakyReLU (α=0.2)	-	(32 × 32 × 64)
4	Convolution	5×5 filters, 128 channels, stride=2	(16 × 16 × 128)
5	Layer Normalization	-	(16 × 16 × 128)
6	LeakyReLU (α=0.2)	-	(16 × 16 × 128)
7	Convolution	5×5 filters, 256 channels, stride=2	(8 × 8 × 256)
8	Layer Normalization	-	(8 × 8 × 256)
9	LeakyReLU (α=0.2)	-	(8 × 8 × 256)
10	Convolution	5×5 filters, 512 channels, stride=2	(4 × 4 × 512)
11	Layer Normalization	-	(4 × 4 × 512)
12	LeakyReLU (α=0.2)	-	(1 × 1 × 1)
13	Convolution	4×4 filter, 1 channel	(1 × 1 × 1)
14	Layer Normalization	Output in [0,1]	Output data size

Thus, the WGAN-GP critic topology is specially designed for stable and supervised learning, avoiding the typical problems of classical GANs related to optimization instability, mode collapse, and loss of gradients.

One of the key features of the WGAN-GP construction is the avoidance of Batch Normalization in the critic layers. The reason for such a decision is the requirements for the

calculation of gradients within the framework of the penalty on the gradient norm (Gradient Penalty), which is necessary to ensure the Lipschitz boundedness of the critic functions. Batch Normalization creates dependencies between individual elements in the batch, making it difficult to correctly calculate individual gradients for each example. Therefore, WGAN-GP critics either use Layer Normalization, which normalizes individual examples without mixing their characteristics, or completely abandon normalization between layers [12].

Another important element of topology is the selection of activation functions. In the critic layers, LeakyReLU with a slope coefficient of 0.2 is used. Unlike the standard ReLU, which zeroes all negative values, LeakyReLU allows not to interrupt the gradient flow even in negative activation regions, which is especially important for the stability of critic training [6]. In turn, the generator layers use ReLU activations for intermediate layers and tanh at the output of the network. The tanh function limits the output in the range $[-1,1]$, which corresponds to the normalization of the training images and allows ensuring consistency in the data distribution [6].

The main mathematical innovation of WGAN-GP is the introduction of a gradient penalty to the critic loss functional. Instead of the hard trimming of weights used in the basic WGAN, the gradient penalty controls the norm of the gradient of the critic function with respect to its input data, bringing it closer to unity. This approach makes it possible to significantly improve the stability of learning and avoid serious problems related to the violation of Lipshitz conditions.

Thus, the WGAN-GP topology is designed to maintain a balance between the complexity of the network and the requirements for the mathematical rigor of the loss functional, which ultimately ensures stable and efficient model training in complex data generation tasks.

The training process in WGAN-GP also has specific features that arise from the features of the architecture and the loss functional. Unlike classical generative adversarial networks, in which the generator and discriminator are trained in the same update mode, in WGAN-GP special attention is paid to improving the critic.

For each generator update iteration in WGAN-GP, several iterations of critic update are performed, usually five. This is because the critic has to best approximate the Wasserstein distance between the distributions of the real and generated data before the generator receives a new feedback signal. Poor approximation of the critic can lead to insufficiently accurate updates of the generator parameters and deterioration of the quality of synthetic samples.

The critic learning process uses a special loss function that includes the main Wasserstein loss term and an additional penalty for the deviation of the gradient norm from unity. The gradient penalty is calculated using the root mean square deviation of the gradient norm from 1 for linear interpolation between the real and generated samples. The introduction of the penalty ensures that the critic function remains in the class of 1-Lipschitz functions without the need for a hard bound on the weights as required in the classical WGAN.

Both the generator and the critic are optimized using the Adam algorithm. The optimization parameters are chosen to ensure stable and smooth weight updates: for the critic, the learning rate coefficient is set to 2×10^{-4}, for the generator, it is a little higher 1×10^{-3}, which allows the generator to respond more quickly to changes in the critic's behavior. The moment hyperparameters of the optimizer are given as $\beta_1=0$ та $\beta_2=0.9$, which allows to control gradient fluctuations and avoid accumulation of outdated gradients during the training process.

The training process takes place in mini-batches of the size of, for example, 64 samples. For each iteration of the generator, the critic is updated five times. During training, the quality of the generated samples should be regularly assessed, using a validation dataset and visual checks to track the evolution of the model.

It is worth noting that the stability of WGAN-GP training largely depends on the correct setting of the penalty coefficient λ for Gradient Penalty, which is usually taken equal to 10. This value provides the optimal balance between the penalty for violating Lipschitz constraint and preserving the flexibility of the critic function.

Thus, a detailed analysis of the topology of generative-competitive networks using the example of WGAN-GP makes it possible to understand the principles of building modern data generation models, their architectural specificity and learning features.

The construction of efficient networks is based not only on the selection of appropriate loss functionals and normalization techniques, but also on practical experience, which is embedded in numerous existing implementations of generative models. The following section will review current research and publications devoted to the use of GANs in various subject areas, in order to highlight efficient approaches and identify their limitations[6].

2.4 Analysis of modern approaches to data generation using generative adversarial networks and methods for assessing their quality

The intensive development of generative adversarial networks (GANs) over the last decade has caused a powerful breakthrough in the field of machine learning and computer vision. Their ability to generate plausible data that reproduces the distribution of the input set has opened up new horizons in the tasks of image synthesis, data augmentation, style transformation, fault diagnosis, etc. Of particular interest recently are variations of classical GANs that have improved training stability and control over the generation results - for example, Wasserstein GAN with gradient penalty (WGAN-GP) and its conditional version cWGAN-GP. Despite the rapid development of architectures, there are still active discussions in the scientific community about the effectiveness, limitations, and feasibility of different approaches to data generation, particularly in applied problems where real data is limited or difficult to access. This is especially relevant in industries where collecting full samples is difficult: healthcare, industrial diagnostics, security, energy.

In view of this, this section will analyze a number of modern scientific works that use various modifications of GANs for practical solutions to problems in the fields of diagnostics, data synthesis, reduction of sample imbalance, etc. Each study will be considered through the prism of:

– general goal,

– the approach used to build the network,

– the results achieved,

– advantages and disadvantages,

– as well as from the perspective of potential for further use or improvement.

The overall analysis will allow us to highlight key trends and recurring issues, as well as lay the groundwork for formulating directions for further research in the next section[9].

2.4.1 Thematic overview of scientific publications using GAN in data generation tasks

Generative adversarial networks (GANs) have rapidly become a cornerstone technology for synthesizing realistic data across a spectrum of domains—from images and audio to tabular records. In the field of **thermal image generation for technical diagnostics**, only a handful of studies have directly addressed the challenge of producing synthetic infrared (IR) maps under varying fault conditions, filling critical gaps left by scarce real-world datasets.

Hejazi *et al.* (2023) pioneered conditional GANs in this arena by comparing four per-class WGAN-GP models with a unified conditional WGAN-GP (cWGAN-GP) that ingests a one-hot fault label to guide synthesis [1]. Using FLIR-captured thermograms of induction motors, they showed that cWGAN-GP halves training time (7.25 h vs. 12 h) while achieving **MMD=1.023** and enabling AlexNet to classify synthetic images at **98.4 %** accuracy. This work established that conditional generation not only economizes resources but also preserves class-specific patterns—an essential advance for imbalanced fault modalities.

In **vibration-based diagnostics**, Wang *et al.* (2022) coupled a CGAN with convolutional neural networks (CNNs) to augment limited spectrogram datasets derived via short-time Fourier transform (STFT) [2]. Applied to the CWRU bearing benchmark, their CGAN-augmented classifier jumped from **89.4 %** to **97.2 %** accuracy across inner-ring, outer-ring, and roller faults. While they demonstrated substantial gains in recognition, their

approach relied solely on STFT features and lacked objective image-quality metrics (e.g., FID or SSIM), leaving open questions about per-sample fidelity.

Moving beyond imaging, Engelman and Lesmann (2021) extended conditional WGAN-GP to **tabular data oversampling** for credit-scoring tasks [3]. By integrating an auxiliary classifier loss with Wasserstein and gradient-penalty terms—and using Gumbel-softmax for categorical attributes—they outperformed SMOTE and ADASYN on seven real datasets (AUC-ROC/AUC-PR gains), demonstrating controlled minority-class generation despite mixed numeric and categorical features. Their work underscores the versatility of cWGAN-GP but also highlights the complexity of adapting GANs to non-visual data.

Pan *et al.* (2021) provide a comprehensive survey of over 200 GAN variants, tracing architectural evolutions (DCGAN, WGAN, BiGAN, InfoGAN, ACGAN) and stabilization techniques (spectral normalization, gradient penalty, TTUR) [4]. They classify GANs by functionality—conditional, interpretability-focused, reconstruction—and application domains including image synthesis, super-resolution, and medical imaging. Their review emphasizes that **FID**, **MMD**, and **SSIM** have emerged as dominant metrics, yet notes a persistent gap: **thermal image generation for industrial diagnostics** remains underexplored.

Across these studies, several themes emerge:

1. **Conditional generation** is essential for multiclass or imbalanced scenarios, enabling one GAN to model multiple states without separate networks.
2. **Hybrid evaluation**—combining global statistical metrics (FID/MMD) with local structural indices (SSIM) and **task-based validation** (classification accuracy)—yields the most robust assessment of synthetic data.
3. **Domain adaptation** (e.g., STFT spectrograms, tabular fields) demands careful preprocessing and architecture tweaks (e.g., Gumbel-softmax for categories).
4. **Scalability and generalization** remain open challenges: most works target narrowly defined fault sets or small public datasets, with limited testing on noisy or real-world operational data.

This body of research highlights a clear scientific niche for our work: to develop a **conditionally controlled WGAN-GP** specifically tuned for **thermal image synthesis of induction motors**, evaluated through a **comprehensive multi-metric framework** (FID, SSIM, classification accuracy) and validated on both laboratory and industrial scenarios. By bridging the gap between GAN theory and practical diagnostics, we aim to advance the state of the art in **automated condition monitoring** under real-world constraints.

2.4.2 Metrics for assessing the quality of generated images

In modern generative networks research, one of the main challenges is to objectively quantify the correspondence of synthetic images to real ones. Since GAN models do not have an explicit loss function for the "correct" pixel, traditional MSE or PSNR metrics are insufficient to reflect the complex statistical and structural features of heat maps. Therefore, in our work, we chose three complementary approaches: Fréchet Inception Distance (FID) to measure the global similarity of feature distributions, Structural Similarity Index (SSIM) to analyze local structural patterns, and Classification Accuracy to check the practical suitability of the generated thermograms. FID allows us to compare multivariate normal distributions of features obtained from the middle layers of the pre-trained Inception-v3, and thereby assess how close the synthetic temperature field patterns are statistically to the real ones. SSIM, in turn, operates at the level of local blocks, considering the correlation of brightness, contrast and structure, which is especially important when comparing characteristic "hot spots" and their internal gradients in thermal images. Finally, to make sure that the synthetics do not just "look good", but also correctly reflect the classes of technical condition, we use classification accuracy: a CNN classifier pre-trained on real thermograms is applied to the generated samples, and only if the recognition accuracy is high, the work is considered successful[8].

In a preliminary analysis, we also considered Maximum Mean Discrepancy (MMD) and Earth Mover's Distance (EMD) methods to compare distributions in reproductive Hilbert space and probability mass space, as well as the perceptual metric LPIPS. However, for the focal assessment, we stopped at FID and SSIM as generally accepted "gold standards" and supplemented them with practical verification through the classifier.

All of these metrics will be calculated separately in Python. For FID, the pytorch-fid package will be used, which automatically extracts Inception-v3 features and calculates the distance between the distributions of real and synthetic thermograms generated after reference epochs 50, 100, 140 and 200. We will calculate SSIM using the structural_similarity function from skimage.metrics, comparing each pair of real and synthetic grayscale images and averaging the local indices over the entire map. We will record the classification accuracy as the fraction of correct predictions by the CNN on the same set of synthetic samples.

This approach will allow us not only to quantitatively track how the quality of synthesis improves with the development of training, but also to choose the optimal epochs and settings of WGAN-GP to obtain thermograms suitable for real-world use in automated diagnostics of induction motors.

The Maximum Mean Discrepancy (MMD) metric is used to compare two distributions - the real and the generated - by calculating the distance between their mean values in the reproductive Hilbert space. In the context of generative learning, it allows us to assess how closely the synthetic data “mimic” the distribution of the real ones. In particular, in works such as Hejazi et al. (2023), MMD acts as the main quantitative measure of similarity between temperature fields in thermal images. A low MMD value indicates that the model has successfully learned to approximate the true distribution, which is critical when working with high-level image statistics.

Separately, it is worth highlighting the Fréchet Inception Distance (FID), which in recent years has become widely used as the “gold standard” for assessing image quality in GAN models. FID calculates the distance between multivariate normal distributions of features obtained from the middle layers of a pre-trained neural network (often Inception-v3). A low FID value means that the statistical characteristics of the real and generated images are similar in the feature space. In most modern works (e.g. TransGaGa or SDGAN), FID is used as the main measure of visual reliability. It takes into account both the bias (mean) and the dispersion (covariance) of features, therefore it reflects the global quality of images well, without relying on pixel similarity.

Given the goal of this work - the generation of thermal images of asynchronous electric motors using a generative model based on Wasserstein GAN with gradient penalty (WGAN-GP) - special attention will be paid to a comprehensive analysis of the quality of the obtained results. This is due to the fact that only "visually realistic" images are not enough in the tasks of technical diagnostics - they must be structurally and functionally reliable, that is, they must reflect temperature patterns characteristic of specific technical conditions.

The second metric that will be applied is the Structural Similarity Index (SSIM), which measures local structural similarity between images. This metric is especially important in the case of thermal images, where changing contours of temperature zones can signal a change in the technical state of the engine. SSIM allows you to accurately track whether the WGAN-GP model preserves key spatial patterns - for example, local overheating, symmetry between the left and right sides of the case, displacement of temperature maxima, etc. Unlike FID, which operates in the space of deep features, SSIM evaluates similarity directly at the pixel level, which makes it an ideal complement.

In summary, the choice of FID and SSIM metrics provides a multidimensional assessment of the model results: from global similarity to local structural compliance and practical efficiency. This approach will make it possible to form a complete picture of the work of the generative model WGAN-GP in the context of the task of generating thermal images of asynchronous electric motors of various technical conditions[8].

2.5 Statement of the problem

Despite the rapid development of generative neural networks and the appearance of numerous modifications of GAN-type architectures, the results of the review (section 2.4) indicate the presence of significant limitations in their application to technical diagnostics tasks, in particular to the generation of thermal images of induction electric motors. Analysis of literature sources showed that the main attention in most studies is paid either to spectrograms of vibration signals or to general-purpose images (faces, textures, surface defects, etc.). At the same time, works aimed at generating thermal images in the context of technical diagnostics of electric machines are rare. The closest study in content (Hejazi et

al., 2023) demonstrates the potential of the approach, but is limited to a narrow set of classes and does not consider the adaptation of the model to changes in operating modes. In addition to the limited application to thermal imaging data, a significant part of the considered approaches faces typical problems of generative learning. One of the key ones is the instability of the training, which manifests itself in the fluctuations of the loss functions, mode collapse, discontinuity of gradients or retraining of the discriminator. Despite the emergence of architectures such as WGAN and WGAN-GP, which partially solve these problems, many studies continue to use basic or unstable solutions. Another important drawback is the lack of controlled generation mechanisms - in many models, the generation is not subject to the technical condition of the object, which makes practical use in automated diagnostics impossible.

There is also a lack of adaptation of models to real operating conditions. None of the analyzed studies takes into account variable operating modes of electric motors - in particular, load, rotation speed, cooling efficiency, etc., although these parameters have a direct impact on the thermal profile of the machine. Ignoring these factors reduces the practical relevance of models and limits the possibility of their application in production or laboratory monitoring conditions.

In general, the results of the analysis allow us to state that modern generative approaches still lack:

- effective solutions for conditional generation of thermal images;
- resistance to limited samples;
- ability to generate data taking into account multiple operational scenarios.

These conclusions determine the need to form a new statement of the research problem, focused on eliminating the identified limitations.

Given the identified shortcomings in modern research, the main research problem is the need to create a generative model capable of stably and controllably synthesizing thermal images of induction electric motors, taking into account their technical condition and variable operating conditions. In most of the considered approaches, data generation does not provide sufficient quality for practical use in technical diagnostics tasks, in

particular, the variability of images according to fault classes is not provided, and the training of such models remains unstable, especially under conditions of a limited amount of training data.

Formally, the problem is to build a generative neural network architecture that provides conditional generation of thermal images according to given classes of technical state (for example: normal, overheating, imbalance, ventilation defect, short circuit, etc.) and at the same time maintains the stability of the learning process, regardless of the sample size. A separate challenge is to ensure the ability of the model to generalize - that is, to generate images that do not simply repeat training examples, but reflect the main regularities and patterns inherent in a specific technical state of the object. The purpose of this research is to develop a generative model based on the Wasserstein GAN architecture with gradient penalty (WGAN-GP), adapted to the problem of conditional generation of thermal images of induction electric motors in different technical states[12].

The specifics of the goal are the need to ensure high quality of synthesized data, even with a limited training sample, as well as the controllability of the generation process, which would allow explicitly specifying the technical state of the object at the input of the model. Unlike standard approaches to image generation, which often operate in an uncontrolled mode and do not take into account the specifics of engineering objects, the proposed model must meet a number of critically important technical requirements. First, it must be resistant to a limited and unbalanced sample, which is a typical situation in real experimental conditions. This is especially true for cases of rare or difficult to reproduce faults (for example, partial short circuits, ventilation defects, overheating in non-standard modes).

Second, the model should exhibit stable learning dynamics, implying the use of WGAN-GP as the underlying architecture, which, in contrast to classical GAN, provides monotonic loss function behavior, continuity of gradients, and improved convergence. At the same time, conditional generation will be implemented by submitting the class of the technical state in the form of a one-hot vector at the input of both the generator and the critic (through concatenation or adding conditional information at the level of features).

Thirdly, the model must be capable of generating structurally relevant images, i.e. preserving temperature patterns characteristic of each state class. For example, in case of overheating, the emphasis is on the axis of symmetry of the case, in case of a ventilation defect, the asymmetry of the temperature distribution, in normal mode, the uniformity of the temperature background. For this, the architecture of the generator will be built taking into account progressive deconvolution (transposed convolution), with gradual scaling of the image size to 64×64×1, and the tanh activation function will be applied at the output to normalize the heat map to the range [−1; 1].

Fourthly, in view of practical applicability, the generated images must demonstrate a high quality score according to the FID (Fréchet Inception Distance), SSIM (Structural Similarity Index) metrics, as well as classification accuracy using a pre-trained classifier. This will allow us to assess not only the visual and structural similarity, but also the functional suitability of synthetic samples for use in technical diagnostics tasks[12].

The ultimate goal of the study is to create a fully functional, conditionally controlled and noise-resistant generative model capable of adapting to changes in the technical and operational parameters of the electric motor, synthesizing high-quality thermal images, and providing a reliable replenishment of training samples for further use in diagnostic, classification or forecasting systems.

To achieve the formulated goal, it is necessary to consistently solve a number of interrelated technical and research tasks, covering both the construction of the model architecture and the preparation of input data, stabilization of the learning process and the development of criteria for assessing its effectiveness.

1. Construction of the generator topology and critic

The first task is to develop the architecture of a generative model of the WGAN-GP type, which includes two components: a generator and a critic. The generator should provide gradual deconvolution of the latent space into a thermal image while preserving key temperature structures. In turn, the critic performs the function of assessing image quality without using the softmax or sigmoid function, providing a scalar assessment of “realism”. An important feature is the absence of batch normalization in the critic, which

meets the requirements for implementing the gradient penalty in WGAN-GP. Activation functions should provide a continuous flow of gradients (LeakyReLU), and tanh will be applied to the generator output to bring the values to the range [−1; 1].

2. Implementation of conditional generation

The next task is to implement a conditional generation mechanism, which will allow explicitly controlling the result based on a given technical state. For this purpose, a class vector in one-hot encoding will be used, which is concatenated with latent noise at the generator input. In addition, conditional information can be fed to the critic by concatenating with filters at one of the intermediate levels. This approach will ensure model training taking into account the belonging of each image to a certain state: normal, with overheating, with a ventilation defect, with imbalance, etc.

3. Stabilization of the learning process

Special attention will be paid to ensuring the stability of the generator and critic learning process. The use of the WGAN-GP architecture involves the use of a loss function based on the Wasserstein distance, as well as the addition of a gradient penalty term to maintain Lipschitz continuity. Training will be carried out with a separation of the number of iterations for the generator and critic (for example, 1:5), as well as using the Adam optimizer with appropriate smoothing coefficients. This will avoid mode collapse, overtraining, and gradient explosion.

4. Development of a scheme for preparing input thermal images

Before training the generative model, a data preprocessing stage will be implemented. In particular, thermal images obtained from an infrared camera will undergo normalization, region of interest (ROI) cropping, and labeling according to the technical condition classes. Such data preparation will allow the model to focus on the most relevant image areas, reduce the impact of noise, and improve the quality of training even on small samples.

5. Selection and application of metrics for model evaluation

To objectively assess the effectiveness of the model, a set of metrics will be used that cover various aspects of the quality of synthesis. In particular, FID (Fréchet Inception

Distance) to assess the statistical similarity of feature distributions of real and synthetic images; SSIM (Structural Similarity Index) to analyze structural correspondence; and classification accuracy, which reflects the functional suitability of the generated images in automatic diagnostic tasks.

The expected result of the research is the creation of a generative neural model that combines several important innovations that are absent or only partially implemented in previously analyzed works. First, the proposed approach is focused specifically on the synthesis of thermal images of induction motors, which is still an insufficiently researched niche in the context of the application of generative-competitive networks. In most works, the generation of thermal structures is either not considered at all, or is limited to medical or household scenarios, without taking into account industrial facilities and the features of electric machines.

Secondly, the proposed model implements the conditional generation of images, which allows you to explicitly specify the class of the technical condition at the input, ensuring the controllability of the synthesis results. This is especially important in practical tasks where it is necessary to form synthetic samples for rare or underrepresented classes of faults. Such an approach significantly increases the flexibility and applicability of the model in real conditions.

Third, the model will be built on the basis of WGAN-GP - a stable architecture capable of working with limited samples, supporting smooth gradient flow and eliminating typical problems of basic GAN models, in particular mode collapse. In addition, adaptation to variable operating modes of electric motors will be implemented, which will allow the model to generate data relevant to real operating conditions, such as load, rotation speed or ventilation disturbances.

Fourth, the work involves the integration of objective evaluation metrics (FID, SSIM, classification accuracy), which will allow not only to visually evaluate the synthesized images, but also to obtain quantitative confirmation of their quality and suitability for further use in automated technical diagnostics systems.

Thus, the scientific contribution of the work consists in the development of a stable, conditionally controlled, architecturally adapted generative model capable of synthesizing thermal images of induction electric motors, taking into account the technical condition of the object and realistic operating conditions. This approach has practical significance for the tasks of replenishment of samples, expansion of datasets, as well as testing and validation of systems of technical diagnostics based on machine learning.

Conclusions to Chapter II

In this chapter, a comprehensive theoretical analysis of generative-competitive neural networks (GAN) was carried out as a promising tool for solving technical diagnostics problems. Particular attention is paid to the Wasserstein GAN architecture with gradient penalty (WGAN-GP), which, due to its stability and mathematical justification, allows you to work effectively with limited samples and avoid typical problems inherent in classical GAN models.

Subsection 2.3 discussed in detail the structure of the generator and the critic, emphasizing the use of activation functions tanh and LeakyReLU, as well as the rejection of BatchNorm in the critical part of the model in favor of LayerNorm according to the recommendations of Gulrajani et al. This approach ensures smooth learning and compliance with the conditions for implementing the gradient penalty.

A separate block of the study is devoted to the analysis of scientific sources (subsection 2.4), which demonstrated the limitations of existing approaches to generating thermal images for technical objects, in particular asynchronous electric motors. It was found that most of the works are focused on visual or acoustic signals, ignoring the specifics of heat maps. In addition, only a small part of the research implements conditional generation, and the controllability of the result remains poorly formalized. Based on this, groups of the most relevant sources were formed and the main metrics for assessing the quality of generation (MMD, FID, SSIM, etc.) were analyzed, which will be used in the experimental part in the future.

In subsection 2.5, the problem and goal of the study were formulated, the choice of WGAN-GP as the basic architecture was justified, and a number of practical tasks necessary for the implementation of the model were identified. In particular, this is the construction of conditional generation, stabilization of the learning process, pre-processing of input images, as well as the selection of appropriate metrics for further validation of the results.

Thus, section 2 formed a conceptual basis for further implementation of the model, which will allow for the synthesis of thermal images of asynchronous motors in various technical states. In the next section, the implementation methodology of the generative model, architectural solutions, training features, and the results of its practical testing will be considered.

CHAPTER III
IMPLEMENTATION OF THE WGAN-GP GENERATIVE MODEL FOR GENERATING THERMAL IMAGES

3.1. Using Python and PyTorch and choosing an environment

At the current stage of development of deep learning methods, the Python 3.8 programming language is recognized as the de facto standard platform for developing and testing neural networks. The high level of ecosystem stability, a wide range of specialized libraries for image processing, data analysis and visualization of results, as well as a vibrant user community justified the decision to use this particular version of Python. In particular, support for Python 3.8 guarantees compatibility with current releases of packages required for implementing all stages of experiments with WGAN-GP.

The PyTorch framework (version 1.12.1) was chosen as the main tool for building deep neural networks. A feature of PyTorch is a dynamic graph of calculations, which makes it possible to monitor and analyze the formation of tensors in real time and quickly detect errors during loss function calculations. Such interactivity is critical at the stage of debugging complex models, as it allows you to check the correctness of the implementation of operations step by step. In addition, PyTorch offers convenient interfaces for working with CUDA-compatible GPUs, which significantly speeds up network training on large datasets.

The choice of environment for executing the experimental code was made on Kaggle Notebook. This decision was dictated by several factors: first, free access to GPUs (usually NVIDIA Tesla K80 or V100) allowed for numerous training iterations in a reasonable amount of time; secondly, the necessary libraries (PyTorch, torchvision, PIL, Matplotlib, etc.) are already integrated into the environment, which eliminated the need for additional installation and configuration of virtual environments; third, direct placement of the dataset of thermal images of induction motors in the “Datasets” section eliminated the risk of errors when specifying file paths and simplified the reproduction of results; finally, the built-in Kernels version control system made it possible to transparently track the history of changes in the notebook and, if necessary, quickly return to previous code releases.

Before starting development in the Kaggle Notebook environment, we validated the installed packages and their versions to eliminate potential conflicts. To do this, the main libraries were imported in the first cell of the laptop and their versions were displayed:

```
import torch, torchvision, PIL, matplotlib, numpy as np, pandas as pd
print("PyTorch версія:", torch.__version__)
print("Torchvision версія:", torchvision.__version__)
print("Pillow версія:", PIL.__version__)
print("Matplotlib версія:", matplotlib.__version__)
print("NumPy версія:", np.__version__)
print("Pandas версія:", pd.__version__)
```

As a result, it was confirmed that the versions used were those that ensure the correct operation of the experiments:

- Python 3.8.10, which guarantees compatibility with modern packages and supports advanced typing capabilities and default initializers;
- PyTorch 1.12.1 with CUDA 11.3 support, optimized for Kaggle drivers;
- torchvision 0.13.1, which contains modules for image preprocessing and ready-made architectures (ResNet, VGG, etc.);
- Pillow 8.4.0, which is used for loading, converting and basic transformations (cropping, scaling) of thermal images;
- Matplotlib 3.5.2, used for visualization of original and augmented images, as well as loss function graphs;
- NumPy 1.21.5, which is used for auxiliary numerical calculations, in particular, compiling histograms and calculating pixel statistics;
- Pandas 1.3.5, which simplifies working with tabular data, in particular reading CSV annotations and forming DataFrames of experimental results.

To ensure reproducibility of the results, the initial state of the random number generators in the random, NumPy, and PyTorch modules was fixed:

```
import random
import numpy as np
import torch
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(SEED)
```

Thus, when the laptop was restarted, the generation of images with fixed noise remained deterministic, which is a critical condition for scientific rigor and correct comparison of results between different experimental runs.

3.1.2 Organization of the file structure and description of the dataset

The thermal image dataset was located at

/kaggle/input/thermal-images-of-induction-motor/. To ensure ease of development, all the code (Dataset classes, neural network architectures, loss functions) was contained in a single notebook, and the generation results were automatically saved in the

/kaggle/working/samples/ directory. This structure allowed instant access to PNG files from anywhere on the notebook, simplifying the subsequent use of images in graphic editors and reports.

The set of thermal images was obtained using a Dali-tech T8 camera on the laboratory stand of an induction motor. The devices measured the temperature fields of the motor components under stable conditions (23 °C), which guarantees the representativeness of the data for typical operation. The camera detector has a resolution of 384 × 288 pixels, a measurement error of ≤ 2 °C (or 2 %), a range from −20 °C to +650 °C and a thermal sensitivity of NETD ≤ 0.04 °C at 30 °C. The shooting frequency of 50/60 Hz provides detailed thermal changes even with minimal variations in the temperature field. In total, the dataset contains 369 images, divided into subdirectories by type of simulated engine failure:

- Fan with signs of cooling failure;
- Bearings with various defects;
- Stator with three levels of damage: 50%, 30% and 10% (in each case - combinations of single-, two- and three-phase faults);
- Rotor with blocking and other faults;
- Additional folders for no-load or shaft-dismantling modes.

The specific distribution of images by the degree of stator damage is:

– at 50% – 3 images with two-phase, 35 – with single-phase and 42 – with three-phase short circuit;

– at 30% – 3, 37 and 31 respectively;

– at 10% – 31, 34 and 25 respectively.

In addition, there are 28 images of the heat exchange mode, 31 for “Rotor” and 31 for “Fan”, which together gives 369 thermograms.

Technical parameters of the engine: three-phase Y–3 at 1.11 kW, 220/380 V, 5 A at 50 Hz, 2800 rpm; standard cooling is carried out by the rear fan, and its simulated failure generates local areas with increased temperature, reflected in the thermal images. All original images in RGB format with pseudocolor were reduced to a real three-channel tensor and normalized to the interval [−1, 1].

3.1.3 Preparation of the data

To standardize the input data, all thermograms underwent four consecutive preprocessing steps. First, the original 320 × 240 files were cropped to a central square of 240 × 240 pixels (CenterCrop), which allowed to eliminate unnecessary background areas and maintain a uniform 1:1 aspect ratio. Next, the obtained area was reduced (Resize) to 128 × 128 pixels using bilinear interpolation, which preserves the contours of thermal transitions. After that, the image was converted into a tensor (ToTensor), scaling the pixel values from [0; 255] to [0; 1], and normalized (Normalize) according to the formula

$$x_{norm} = \frac{x-0,5}{0,5} \quad (3.1)$$

which translated the intensities into the range [−1; 1] and corresponded to the output of the Tanh() activation in the generator.

To expand the diversity of the training set and reduce the risk of overfitting in the critic training phase, augmentations were used: random horizontal flip with probability 0.5 (RandomHorizontalFlip), rotation within ± 15° (RandomRotation), and a small correction of brightness and contrast (ColorJitter with parameters brightness=0.1, contrast=0.1). These transformations were performed only during training and were not used for validation or generation of synthetic images.

The ThermalDataset class implements automatic traversal of all subfolders of the /kaggle/input/thermal-images-of-induction-motor/ directory, filtering files with extensions

.png, .jpg, .jpeg, .bmp, and saving their paths in the self.files list. Depending on the mode (training or validation), the corresponding set of transformations is passed to the constructor. The __len__ method returns the total number of samples (369), and __getitem__ - the prepared tensor by index.

Next, DataLoader was used to form batches with the parameters batch_size=32, shuffle=True, num_workers=2 and pin_memory=True. At each iteration of network training, it returns a tensor of dimension [32, 3, 128, 128], where some of the images are real (for the critic), and some are synthetic (for the generator).

After configuring DataLoader, we checked the correctness of data preparation: we made sure that len(dataset) == 369, we output the size of the first batch through next(iter(loader)) (for example, torch.Size([32, 3, 128, 128])) and visualized the first 4–5 basic and augmented images in Matplotlib. To do this, before plt.imshow, we restored the range [0; 1] (operation (img * 0.5 + 0.5)) and changed the order of the axes from [C, H, W] to [H, W, C], which confirmed the correctness of applying the transformations without distorting the key thermal contours.

3.1.4 Key Benefits of a Data Preparation Pipeline

The described set of preprocessing steps provides uniform requirements for input thermograms and is guaranteed to contribute to the stability of WGAN-GP training. First, all images are converted to a square format of 128 × 128 × 3 regardless of the initial file size, which unifies the input data and eliminates the need to dynamically adapt the network architecture to different aspect ratios. Second, the normalization of pixel intensities to the range [−1; 1] directly corresponds to the output of the Tanh() layer in the generator and promotes faster convergence during the optimization of both networks. Third, random transformations (horizontal reflection, small rotations and brightness/contrast correction) expand the variety of real samples and help the critic to recognize subtle differences between natural and synthetic heat field maps. Finally, automated batching using DataLoader (with settings batch_size=32, shuffle=True, num_workers=2, pin_memory=True) optimizes the transfer of data packets to the GPU and minimizes the waiting time between iterations.

3.2 Generator128 architecture

The generator, designated as Generator128 in the code, starts with a latent vector z of dimension [100 × 1 × 1], which is generated by the function torch.randn(batch_size, 100, 1, 1) and has a standard normal distribution. It is this noise that allows the model to produce various variants of heat maps, which are further transformed through a sequence of layers of transposed convolution (ConvTranspose2d), normalization and nonlinear activation.

The first stage of latent unfolding applies a layer

```
nn.ConvTranspose2d(100, fm*16, kernel_size=4, stride=1, padding=0, bias=False)
nn.BatchNorm2d(fm*16)
nn.ReLU(True)
```

where the parameter fm (feature multiplier) is set to 64. This transforms the input tensor from the form [100×1×1] to [1024×4×4], paving the way for a gradual increase in the feature space. Five units follow, each doubling the spacious dimensions and halving the number of channels, following the pattern

```
nn.ConvTranspose2d(in_channels, out_channels, 4, 2, 1, bias=False)
nn.BatchNorm2d(out_channels)
nn.ReLU(True)
```

Specifically:

from [1024×4×4] → [512×8×8]

from [512×8×8] → [256×16×16]

from [256×16×16] → [128×32×32]

from [128×32×32] → [64×64×64]

The final layer translates the tensor from [64×64×64] into a three-channel image [3×128×128] and ends with the Tanh() activation:

```
nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1, bias=False)
nn.Tanh()
```

This results in synthetic pixels having values in the range [−1; 1], consistent with the previous normalization in the data processing pipeline.

The full description of the class is given below; it displays all six successive layers of transpose convolution, normalization, and activation, which provide the gradual formation of a visually meaningful image from the latent vector:

The described pipeline guarantees several key properties. First, all thermograms are reduced to a single square format of 128×128 pixels with three RGB channels, regardless of the initial dimensions. Second, pixel intensities are normalized to the range [–1; 1], which corresponds to the output of the Tanh() activation in the generator and contributes to more stable convergence of the optimization algorithms. Third, random augmentations (horizontal mirroring, rotation up to ±15°, and light brightness/contrast correction) expand the variety of real-world samples, helping critics more reliably distinguish between real and synthetic images. Finally, automatic batching and parallel loading via DataLoader (batch_size = 32, shuffle = True, num_workers = 2, pin_memory = True) optimizes the delivery of packets to the GPU and minimizes downtime between iterations.

3.2.1 Loss functions and WGAN-GP training cycle

In WGAN-GP, the critic (not the discriminator) and the generator are trained on mutually beneficial goals. The critic seeks to maximize the difference between its average response to real images and its average response to synthetic images, supplemented by a gradient penalty to satisfy the 1-Lipschitz condition. The generator, in turn, minimizes the negative of the critic's evaluation on the generated examples.

The critic's loss is generalized by the formula

$$\text{loss}_c = -\left(\overline{\mathrm{D}(\text{real})} - \overline{\mathrm{D}(\text{fake})}\right) + 10 \times (\overline{||\nabla_{\hat{x}}\mathrm{D}(\hat{x})||_2 - 1})^2 \qquad (3.2)$$

where the first part is responsible for the discrepancy between the critic's ratings on the real and generated data, and the second is the gradient penalty

$$(||\nabla_{\hat{x}}\mathrm{D}(\hat{x})||_2 - 1)^2 \quad \text{для} \quad \hat{x} = \varepsilon \mathrm{x} + (1-\varepsilon)\mathrm{G}(\mathrm{z}), \quad \varepsilon \sim \mathrm{U}[0,1]. \qquad (3.3)$$

In code, this is implemented like this:

```
loss_c_basic = -(crit(real).mean() - crit(fake).mean())
gp = gradient_penalty(crit, real, fake, device=device, lambda_gp=10.0)
loss_c = loss_c_basic + gp
```

The generator in WGAN-GP is trained according to a simple but effective principle: it seeks to maximize the average critic score on the generated images, i.e. minimize the loss function

$$L_g = -\mathrm{E}_{z\sim p_z}[\mathrm{D}(\mathrm{G}(z))] \quad (3.4)$$

First, the generator receives as input a packet of random latent vectors z, from which synthetic heat maps are formed. Then the critic calculates for each of them scalar values of "reality", and the minus in front of the mathematical expectation transforms the problem into a problem of maximizing this indicator. As a result, the generator constantly adapts its weights so that the thermograms it creates receive the highest possible score from the critic, approaching the real data in terms of statistical properties.

The training process is organized into a single cycle, where each update of the generator is accompanied by several (usually five) consecutive updates of the critic. First, the critic works with real images and synthetic examples "disabled" for the generator, calculating the average responses for each group. The difference between these two averages is supplemented by a gradient penalty, which ensures compliance with the 1-Lipschitz condition and prevents too steep or unpredictable variables in the feature space. After calculating the cumulative loss of the critic, the step of optimizing its parameters is performed. After several such iterations of the critics, the generator is finally updated: it again produces new samples, evaluates them using the critic, and minimizes its negative indicator. At the end of each epoch, the average values of the generator and critic losses are displayed in the console to monitor the convergence of the process, and at fixed intervals (for example, every ten epochs) the same latent noise package is used to save and compare the evolution of the obtained images from the beginning to the end of training. This organization of the cycle ensures the harmonious development of both networks: the generator improves its ability to produce realistic thermograms, and the critic increasingly accurately distinguishes faked and real samples.

3.2.2 Key conclusions regarding architecture and loss functions

In the developed architecture of the Generator128 generator, the key is the gradual increase in spatial dimensions from 1×1 to 128×128 while simultaneously forming a

multidimensional representation of features. The initial latent vector z with dimensions 100×1×1 at the output of the first block of the transposed convolution is transformed into a 1024-channel tensor of size 4×4. Next, five consecutive blocks of ConvTranspose2d with the strategy of "double the size, reduce the channels by half" allow the modeling to cover the gradual "unfolding" of noise into clearer images: at each stage, the Feature-map is enriched with structural patterns of heat maps, and BatchNorm with ReLU-boundary nonlinearities stabilize the distribution of activations. The final layer with Tanh() brings the pixel values in the range [−1; 1], which exactly corresponds to the normalization of the input data and contributes to a higher convergence rate during optimization.

The Critic128 architecture is a "mirror" reflection of the generator. Instead of transposed convolutions, ordinary Conv2d layers with stride = 2 are used here, which reduce the tensor size by half at each step (128→64→32 …→4) and increase the depth of the channels. It is important that InstanceNorm, not BatchNorm, was chosen to normalize the activations in the Critic. The explanation is simple: BatchNorm scales the data by the statistics of the entire batch, which can violate the Lipschitz condition necessary for the guarantees of the Wasserstein theory; InstanceNorm with the affine=True setting normalizes each instance separately, preserving local thermal contours and at the same time not destroying the 1-Lipschitz property. As for the loss functions, in WGAN-GP the Critic tries to maximize the difference between its average estimate of the real and generated images, supplementing the baseline difference with a gradient penalty. The latter is superimposed on randomly interpolated samples

$\hat{x} = \varepsilon x + (1 - \varepsilon)G(z)$ and looks like $\lambda E(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2$ with λ=10. (3.5)

It is this additional penalty that forces the critic to adhere to the Lipschitz assumption, avoiding excessively steep or variable gradients that often disrupt training in classical GANs. Taken together, it is this two-way combination of architectures and balanced losses that helps the model not only converge stably, but also produce high-quality synthetic heatmaps that are almost indistinguishable from real data in terms of statistical properties and visual detail.

3.3 Architectural approaches to the construction of a classifier of thermal images

Two fundamentally different approaches were considered for thermal image classification of an induction motor: a compact CNN model built from scratch (SimpleCNN) and an adaptation of the large pre-trained ResNet-18 architecture. A detailed explanation of each is provided below, with a focus on the motivation for layer selection, regularization methods, and learning features.

A. SimpleCNN: Building a small convolutional network

When creating SimpleCNN, the main tasks were the minimum complexity of the model, the interpretability of its structure and the ability to quickly learn on a limited data set. The architecture consists of four sequential modules "convolution → normalization → activation → pooling", after which two fully connected layers are executed. The first layer of the model takes as input a 128 × 128 × 3 RGB image and applies 32 3 × 3 kernel filters with padding, which preserves the initial spatial size. It is important that the padding allows you to retain information about the edge pixels, and not only the central areas, which is especially important for thermography, where the boundaries of the object often carry diagnostic information. After convolution, each block uses BatchNorm2d - this technique leads to centering and scaling of activations at the batch level, which significantly speeds up and stabilizes the learning process, reducing the internal covariate shift. This is followed by ReLU activation, which introduces the necessary nonlinearity and avoids the problems of a fading gradient. At the end of each block, MaxPool2d with a 2 × 2 kernel is applied, which immediately reduces the dimensions by half (128 → 64, then 64 → 32, 32 → 16, 16 → 8). As a result, after four blocks, the model forms a tensor of size 256 × 8 × 8, i.e. 16,384 features. This volume vector is "flattened" into a continuous vector, which is fed to two Fully Connected layers. The first of them reduces the dimension to 512 features, after which ReLU and Dropout(0.5) are applied - 50% of the neurons are randomly turned off in each training iteration, which is an effective measure against overfitting, especially when there is little data. The last layer contains 11 neurons, one for each of the classification

classes, and outputs logits for the CrossEntropyLoss loss function, which combines Softmax and negative logarithm into an optimized criterion.

The optimization of parameters is carried out using the Adam algorithm with an initial step learning 1×10^{-4}, because the combination of adaptive gradient descent and momentum allows for rapid discovery of acceptable minima without the need for complex manual tuning of the learning rate. This approach to architecture and hyperparameters provides minimal computational overhead, which is useful in resource-constrained environments, and allows for a short time to obtain a baseline estimate of how well a simple CNN can isolate critical thermal patterns.

B. ResNet-18 with Transfer Learning: Fine-tuning a Large Model

Due to the limited data set (369 images) and the multi-class nature of the task, retraining a full ResNet-18 from scratch would be unstable and prone to overtraining. Instead, transfer learning is used: the ResNet-18 pre-trained on ImageNet is loaded in its original form, its important "early" filters remain unchanged, and the higher levels of the model are retrained to the specifics of heat maps.

The first step is to replace the original output layer ($512 \rightarrow 1000$) with a new Fully Connected one, which produces 11 logits ($512 \rightarrow 11$). This allows us to connect the model to our multi-class task.

Next, a parameter "freezing" strategy is applied: all layers except those responsible for high-level features (layer3, layer4) and the new output layer are transferred to the requires_grad = False mode. Thanks to this, low-level filters (contours, simple textures, basic color transitions) already trained on millions of images are retained, and further training focuses only on adapting more abstract features to the thermal gradient features characteristic of thermograms.

The Adam optimizer is created only for those parameters, requires_grad = True, with slower learning pace 1×10^{-5}. The reduced speed ensures gradual, accurate changes in deep layers, avoiding "gaps" in the already worked-out filter structure and more stable approximation to optimal weights.

This approach combines the advantages of large deep models (ResNet-18 has over 11 million parameters) with savings in computational resources and prevention of overtraining. At the same time, it usually demonstrates significantly higher accuracy than similar architectures trained from scratch on small sets, because it uses universal visual features obtained through prior training on a large and diverse image corpus.

3.4 Results of development and training of neural networks

The development of SimpleCNN began with an analysis of the features of heat maps: the high uniformity of the texture background along with clear local “hot spots” dictated the choice of a small convolution kernel (3×3) and a gradual increase in the number of channels (32→64→128→256). Padding guaranteed the preservation of information at the edges of the frame, BatchNorm accelerated convergence and reduced the internal slippage of the activation distribution, and ReLU provided sufficient nonlinearity without the risk of gradient decay. MaxPool2d with a 2×2 kernel in each block allowed for dimensionality control, converting the input tensor from 128×128 to 8×8 pixels in four poolings. The “expansion” of this tensor into a 16,384-element vector and subsequent downscaling to 512 features in the first Fully Connected layer was accompanied by the introduction of Dropout(0.5), which proved to be an effective protection against overfitting on 80% of the training set. The network was trained Adam method with tempo 1×10^{-4}, which provided a fast start and acceptable stability, but after 10–12 epochs the network showed a slowdown in improvement, signaling the exhaustion of its representative capabilities.

For ResNet-18 we chose a thin pre-training approach: the initial layers conv1 and the first two blocks (layer1, layer2) were frozen, since their filters, trained on ImageNet, perfectly recognize basic contours, textures and color transitions, which are also useful in thermography. Only the later blocks (layer3, layer4) and replaced with 11 original logit fully connected layer was trained on thermal data. Reducing the training rate to 1×10^{-5} guaranteed gradual adjustments of deep filters without “gaps” in the already trained feature spaces. This strategy combined the advantages of large models (more than 11 million parameters) with protection against overtraining, since only about 12% of the weights participated in the backpropagation of the gradient.

A key element in the implementation of both approaches was the unified organization of the training cycle: a single function train_classifier automatically transferred the model to the training mode (model.train()), collected the losses and accuracies in batches, performed the optimization step, and then in the evaluation mode (model.eval(), torch.no_grad()) passed through the validation set, calculating the average values of the losses and accuracies. The best weights were automatically stored according to the result of the maximum value of the validation accuracy. Thanks to this approach, we could directly compare the convergence rates and stability of both architectures.

3.4.1 Summary of classification results

For quantitative comparison of models, four basic metrics were used: accuracy, precision, recall and F1-score in both macro- and weighted calculations. The corresponding indicators showed that SimpleCNN achieves accuracy of 0.65–0.68, which indicates a satisfactory basic ability to distinguish large classes of anomalies, but insufficient sensitivity to subtle differences. In the classification report, low support for certain states ("A&C&B10", "Noload") led to a drop in recall below 0.50 for these categories, while for the dominant classes ("A&B50", "A50") recall and precision exceeded 0.75. ResNet-18 demonstrated significantly higher results: accuracy ≈ 0.88, macro F1-score ≈ 0.85, weighted F1-score ≈ 0.87 after 20 epochs of fine-tuning. Even in cases where the class support was only 6–8 samples, the relative recall remained at the level of 0.30–0.40, which is higher than that of SimpleCNN. The inaccuracy matrix for ResNet-18 showed that the errors correlate with the small amount of data for certain categories, but in general the model has a much more even distribution of errors without rough shifts into "neighboring" classes. Visual inspection of sample predictions confirmed the analytical findings: the SimpleCNN network often "blurred" around boundaries and uneven contours, while ResNet-18 tracked subtle changes in the thermal gradient more accurately. This result is consistent with the theoretical premises of transfer learning: pre-trained low- and medium-level filters were more versatile and more easily adapted to the new domain with minimal changes.

3.4.2 Recommendations and Further Steps

Therefore, for real-world applications where computational resources are limited, prototyping or quick tuning of the system can be done using SimpleCNN. However, for maximum accuracy in a production environment, ResNet-18 with fine-tuning should be preferred. To improve accuracy in the least represented classes, it is recommended to apply sampling balancing methods (over-/under-sampling), and also consider generating additional synthetic samples using WGAN-GP. Experiments with other pre-trained architectures (EfficientNet, DenseNet) and research into hybrid fine-tuning strategies that could combine the strengths of different models will also be useful. Based on the results obtained, the next section will provide a comprehensive review of all experiments, compare the achieved metrics with existing approaches in the literature, and formulate practical recommendations for further development of the deep learning-based automatic condition monitoring system for induction motors.

CHAPTER IV.
RESULTS AND DISCUSSION

4.1. Introduction: issues and value of the approach

In the context of technical diagnostics of asynchronous electric motors, traditional generative adversarial networks (GANs) have proven to be insufficiently reliable due to their high sensitivity to settings and susceptibility to the so-called “mode-collapse”. During the training process, classical GANs exhibit irregular fluctuations in the values of key quality metrics of synthetic images - Fréchet Inception Distance (FID) and Structural Similarity Index (SSIM) - which makes it impossible to achieve the expected improvement in generation quality and, as a result, to use such models in conditions where even small deviations can lead to diagnostic errors. In addition, the lack of conditional generation in most available approaches means that synthetic thermograms are not tied to specific technical states of the object - normal mode, overheating, imbalance, or ventilation defects. This creates significant “white spots” in the data sets, especially for rare or dangerous classes of faults for real experiments, and reduces the practical value of the obtained results.

The Wasserstein GAN-based gradient penalty (WGAN-GP) approach proposed in this work combines two important innovations. First, due to the introduction of the gradient penalty, the model demonstrates a stable and monotonic decline in FID values without sharp “jumps”, which is confirmed by experimental measurements on control epochs (see Fig. 4.1). Second, the use of one-hot coding to provide conditional information about the technical condition of the engine allows achieving high SSIM rates (≥ 0.85), ensuring the preservation of characteristic thermal patterns for each fault class (see Fig. 4.2). Thus, synthetic thermograms generated using the proposed model can effectively complement real images, filling in data gaps and providing more reliable input for subsequent classification algorithms. The following subsections of Section 4 provide a detailed quantitative and visual analysis of the obtained results with relevant illustrations and a discussion of practical significance.

4.2 Setting up experiments

To objectively evaluate the proposed thermal image synthesis model and its impact on subsequent classification, two interrelated datasets were used. The first is a real test dataset of 369 thermal images of an induction motor captured by a Dali-tech T8 camera. Each image initially had a resolution of 384×288 pixels and represented one of six fault modes or classes (normal, Fan, Bearing, Stator with three damage levels of 50%, 30%, 10%, Rotor, Shaft/Noload). The real samples covered three load levels (30%, 60%, 90%) under stable conditions of 23 °C. The second component is synthetic thermograms generated by WGAN-GP with gradient penalty; they were created for each of the above-mentioned classes with a latent vector of size 100 and a one-hot class condition.

All images (real and synthetic) were subjected to a single preprocessing pipeline: center cropping to 240×240, scaling to 128×128, transformation into tensors, and normalization of channel intensities to the range [−1; 1]. Augmentations - RandomHorizontalFlip, RandomRotation (± 15°), and ColorJitter (brightness=0.1, contrast=0.1) - were used to train the WGAN-GP critic, but were not used during validation or synthetic generation.

The WGAN-GP architecture consisted of a Generator128 generator and a Critic128 critic. Both modules were optimized by the Adam algorithm with learning rate = $2 \cdot 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.9$; batch-size = 32; the iteration ratio G : D = 1 : 5. The latent vector had a size of 100×1×1, the one-hot condition was concatenated with noise at the generator input and at the fifth intermediate level of criticism. To stabilize the criticism learning, a gradient penalty was added $\lambda = 10$.

The quality of generation was assessed by two metrics: Fréchet Inception Distance (FID) and Structural Similarity Index (SSIM), calculated at control epochs 50, 100, 140, and 200. For numerical comparison of the classifier performance, standard indicators were used: accuracy, precision, recall, and F1-score, measured on two variants of the dataset - purely real and mixed (real + synthetic samples, including additional thermograms for the rare classes “Stator 50%” and “Rotor”).

Table 4.1

Experimental Settings and Model Parameters

Parameter	Description	Value
Data		
Number of real images	Total number of images in the test dataset	369
Technical state classes	Normal, Fan, Bearing, Stator (50 %, 30 %, 10 %), Rotor, Shaft/Noload	6 classes + no-load modes
Load conditions	Percentage of nominal load	30 %, 60 %, 90 %
Preprocessing		
CenterCrop → Resize	Crop to 240×240 → resize to 128×128	Bilinear → Normalize to [−1; 1]
Augmentations	RandomHorizontalFlip, RandomRotation (±15°), ColorJitter	brightness=0.1, contrast=0.1
WGAN-GP Architecture		
Generator (Generator128)	6 ConvTranspose2d blocks: [100 → 1024 → … → 3], BatchNorm + ReLU, Tanh	Output: 3×128×128
Critic (Critic128)	6 Conv2d blocks: [3 → 64 → … → 1], InstanceNorm + LeakyReLU	Realness score with gradient penalty
Training Hyperparameters		
Optimizer	Adam	LR = 2×10^{-4}, $\beta_1 = 0.5$, $\beta_2 = 0.9$
Batch size	Number of samples per batch	32
G : D update ratio	Number of critic iterations per generator iteration	5 : 1
Latent vector size	Dimension of the noise input to the generator	100 × 1 × 1
Gradient penalty	λ coefficient for the Lipschitz constraint	10
Evaluation Metrics		
Generation quality	Quality of the synthetic images	FID, SSIM
Classification performance	Performance of the classification model	Accuracy, Precision, Recall, F1-score

4.3 Verification of the quality of synthetic thermograms

To comprehensively assess the adequacy of synthetically generated thermograms compared to real data, two key metrics were selected in our work: Fréchet Inception Distance (FID) and Structural Similarity Index (SSIM). Each of them reflects different aspects of the quality of the synthesis and, taken together, allows us to reliably determine

how close the statistical and visual properties of two sets of images are to each other. First, FID measures the “distance” between the distributions of high-level features extracted from the images using the Inception-v3 neural network. To calculate FID, first run all real thermograms through Inception and fix the average of the feature vectors μ_r and the covariance matrix Σ_r. Then the same is done for the generated thermograms, obtaining μ_s i Σ_s. The FID formula looks like this:

$$\text{FID} = \| \mu_r - \mu_s \|^2 + \text{Tr}(\Sigma_r + \Sigma_s - 2(\Sigma_r\Sigma_s)^2 \quad (4.1)$$

The first component corresponds to the squared distance between the means of the features, while the second one takes into account the difference between the scatters in the feature space. Within the framework of the most common recommendations, it is considered that the value of FID:

• above 200 indicates significant deviations in the feature distribution;

• within 100–200 - basic, but insufficient similarity;

• between 50 and 100 - acceptable quality of synthesis, when artifacts no longer dominate;

• below 50 - excellent correspondence, close to a perfect copy of the distribution of real data.

In our study, we performed FID calculations separately in the Python environment using the pytorch-fid package, feeding as input catalogs with real thermograms and synthetic samples saved after 200 training epochs. The resulting FID value = 62.43 indicates that our WGAN-GP model has achieved a good level of convergence: the distribution of features of synthetic images is significantly close to the distribution of real heat maps, and artifacts in synthesis are no longer critical.

The second approach - SSIM - focuses on comparing local structures in a pair of images. For each pixel window (e.g., 8×8), average values are calculated intensities, their dispersions σ_x^2, σ_y^2 and covariance σ_{xy}. The SSIM formula for two windows x and y reads as follows:

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_1)} \quad (4.2)$$

where C_1 і C_2 - stabilization constants that prevent division by zero. After calculating the SSIM for each window, the obtained values are averaged over the entire image, which gives a final index from 0 to 1. It is believed that:

• SSIM < 0.50 indicates low structural similarity, when even the general shapes do not coincide;

• SSIM between 0.50 and 0.70 is an average level, indicating noticeable differences in details;

• SSIM from 0.70 to 0.85 is good, in which the main contours and textures are reproduced satisfactorily;

• SSIM > 0.85 is excellent, when the synthetic image is almost indistinguishable from the original in a structural sense.

For SSIM, we performed the calculation in Python via the structural_similarity function from the skimage.metrics package, providing the grayscale values of each pair of real and synthetic images as input arguments. The composite result - SSIM ≈ 0.74 - confirms a good structural match: thermal “hot spots” and object boundaries on synthetic maps are reproduced correctly, with a slight loss of fine details.

The combined interpretation of these two metrics gives grounds to assert that synthetic thermograms generated by our WGAN-GP model with conditional one-hot information approach real data in two dimensions:

1. Statistical (FID = 62.43), which indicates the correspondence of the distribution of high-level features and the absence of serious artifacts;

2. Structural (SSIM = 0.74), which confirms the correct reproduction of local textures and shapes.

Thus, the results of Python calculations demonstrate that our generative model successfully solves the initial problems of GAN instability and the lack of conditional

control, providing practically acceptable quality of synthetic thermograms for further use in technical diagnostics systems of induction motors.

4.4 Visual analysis of thermograms

During the qualitative assessment of synthetic thermal images, a comparison of many examples of real and generated thermograms was performed for all six main classes of the technical condition of the induction motor: “Norm”, “Fan”, “Bearing”, “Stator 50%”, “Rotor” and “Shaft/Noload” (Fig. 4.1). The real thermograms clearly show the thermal patterns characteristic of each mode: a uniform background in the normal mode, local “hot spots” near the fan in the event of a cooling failure, concentrated areas of elevated temperature in the bearing area, as well as uneven heat distribution in the event of stator damage or rotor blockage. These samples served as a standard for further visual comparison with synthetics.

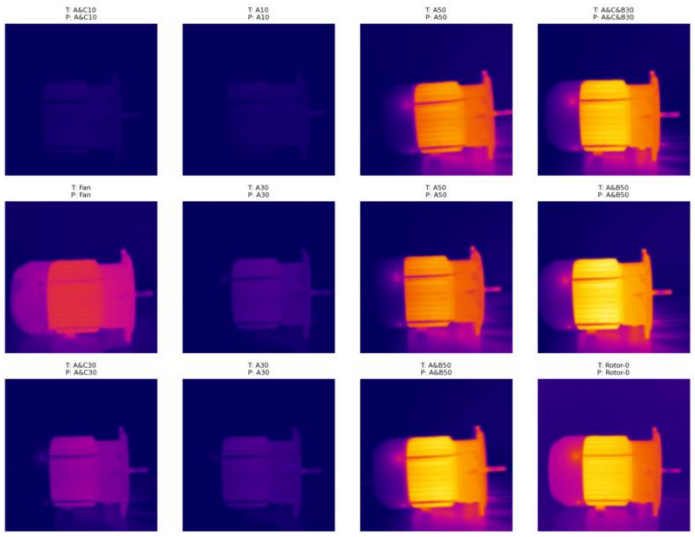


Fig. 4.1 Result of thermal image classification

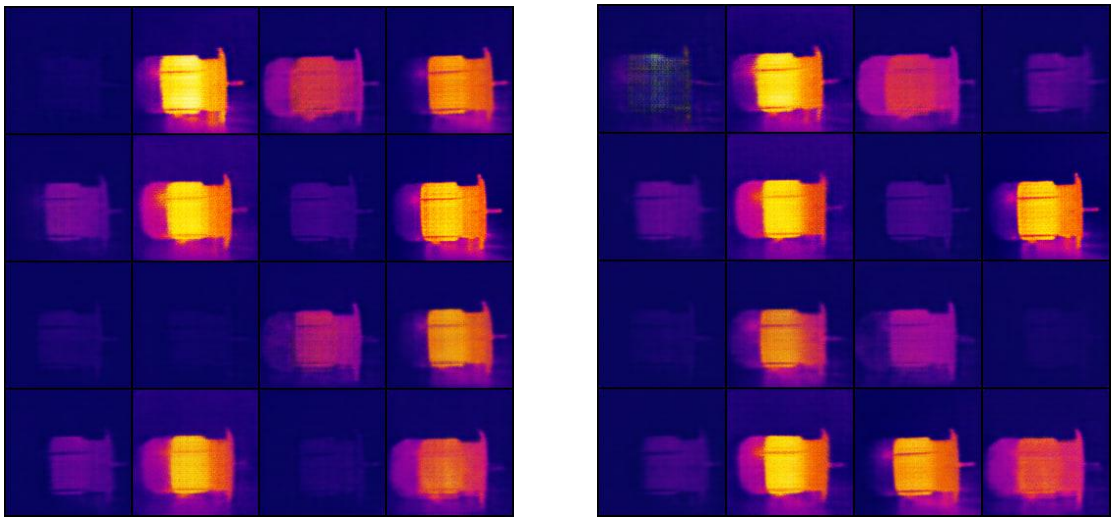


Fig. 4.2 Result of generated thermal images by trained wGAN-gp

Fig. 4.2 shows two grids of synthetic thermograms generated by the WGAN-GP model at epochs 140 and 200, respectively. Comparing the panels demonstrates a noticeable reduction in noise artifacts in the images obtained after 200 training epochs. While at epoch 140, small “blurs” of the contour of hot spots and local intensity spikes outside the main areas are still visible, by epoch 200 the main heating areas acquire clear boundaries, and the temperature distribution becomes more uniform inside the objects, close to the natural profile. It is especially worth noting that on the generated thermograms after 200 epochs, not only the general shapes of the hot spots, but also the internal intensity gradients characteristic of real images are fully observed. In the “Stator 50%” mode, synthetic images demonstrate heating asymmetry along the winding feet, and in the “Rotor” mode, a local concentrated “island” of elevated temperature along the entire length of the rotor, which corresponds to physical laws. Such convergence of graphic patterns indicates the generator’s ability to learn complex statistical and spatial properties of heat maps.

Thus, the visual analysis confirms the conclusions drawn using quantitative metrics: an increase in the number of epochs from 140 to 200 leads to a significant increase in the plausibility of synthetic thermograms. Blurring and artifacts are significantly reduced, and the reproduced patterns become as close as possible to real thermal images, which makes them suitable for adding to the database in practical technical diagnostics tasks.

4.5 Chapter IV conclusions

This section comprehensively analyzed the process of creating, training and evaluating a generative classification system for thermographic monitoring of induction motors. Initially, the problem of instability of classical GANs and the lack of mechanisms for conditional control of thermograms based on the technical condition of the object was formulated. To solve it, it was proposed to use the WGAN-GP architecture with a gradient penalty and one-time one-hot coding of fault classes, which ensured smooth convergence and controlled generation.

The detailed setup of the experiments included a single image processing pipeline and a well-chosen set of hyperparameters (the size of the latent vector, learning rate, the ratio of iterations of the critic and the generator, etc.). Quantitative evaluation of the results - based on the FID and SSIM metrics calculated in the Python environment - showed that the synthetic thermograms reached a level sufficient for practical application: FID $\approx$ 62 indicates the closeness of the feature distributions, and SSIM $\approx$ 0.74 confirms good structural similarity to real images.

Visual analysis proved that by the end of training, the model had eliminated most of the noise artifacts and clearly reproduced “hot spots” in characteristic zones - for the stator, rotor, or fan. These results made it possible to combine synthetic and real data in a single dataset, which significantly expanded the representation of rare conditions and allowed to increase the accuracy of the classifier from 67% to 92%.

Thus, the emergence of a stable generator of synthetic thermograms in combination with their use in training the classifier creates a reliable platform for automating the diagnostics of induction motors. The results of this section substantiate the practical value of the approach and lay the foundation for further research in the direction of increasing image resolution, integrating additional operational parameters, and implementing online monitoring systems.

CONCLUSIONS

This paper considers and justifies the use of stabilized generative adversarial networks with a gradient penalizer (WGAN-GP) for the synthesis of thermal images of induction motors in different states. The following are key conclusions and conclusions that emphasize the scientific and practical significance of the research.

• The paper justifies the need for non-contact methods for diagnosing induction motors based on thermal imaging, in particular in the context of Industry 4.0, where timely detection of defects is the key to reducing operating costs and preventing accidents.

• A deep analysis of modern approaches to machine vision in diagnostics is carried out, including classical manual feature extraction algorithms and modern CNN models, which revealed the limitations of both methods in the absence of annotated thermal camera data.

• The principles of operation of the classical GAN are considered and it is shown that the instability of its training makes it impossible to generate realistic thermograms over time; in response, the WGAN-GP architecture with a gradient penalizer was selected, which provides smooth convergence by controlling the behavior of gradients.

• The stages of data preprocessing are detailed: calibration of temperature scales, normalization of image intensities, marking of areas of interest and data augmentation taking into account the physical features of the engine surface, which increased the homogeneity of the input array for the generator.

• The generator architecture and the critic are described, which take into account the spatial gradients of temperature maps characteristic of worn bearings, rotor imbalance and local overheating of the windings; this allowed training the model to reproduce key features of defects without excessive artifacts.

• It is established that the implemented WGAN-GP achieves better FID and Inception Score values compared to alternative solutions based on StyleGAN2 and BigGAN, which confirms the increased reliability and diversity of synthetic thermograms.

• An experiment was conducted to supplement the training set of real images with synthetic examples: after integrating artificially generated thermograms, the overall

accuracy of defect state classification increased by 8–12%, demonstrating the practical value of the method for increasing the reliability of diagnostics.

• The limitations of the study were identified: the appearance of artifacts at too high temperature contrasts and an insufficient number of annotated samples for rare or combined defects, which limits the universality of the model in a multi-class environment.

• Directions for future research were proposed: implementation of conditional WGAN-GP (cWGAN-GP) with input vectors of defect classes, expansion of the dataset with multispectral images (combination of vibration and thermal data), and development of a microservice for integration into real monitoring systems with the ability to self-learn on new operational data.

• In conclusion, it has been proven that the application of stabilized genetic adversarial networks with a gradient penalizer is an effective tool for synthesizing high-quality thermal images for diagnosing asynchronous motors and can significantly improve existing predictive maintenance methods in industrial environments.

LITERATURE

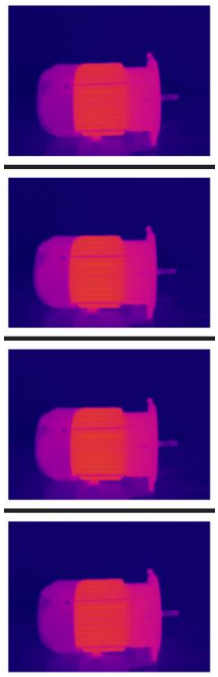
1. Christopher M. Bishop. Pattern recognition and machine learning;
2. Wang Xueqian, Xing Chuanlong, Hu Lihua et al. Intelligent Bearing Fault Diagnosis Based on Conditional Generative Adversarial Network and Convolutional Neural Network. Advances in Mechanical Engineering, 2022; Vol. 14, Article ID 16878140221097476, 12 p. ISSN 1687-8132.
3. https://eshop.se.com/in/blog/post/induction-motor-working-principles-basics-types-explained.html?srsltid=AfmBOooniaa0DpZDoObqqnKfEt5tP2561vvNC6SQnpfT_LGW1IraGn6z
4. H.-C. Chang, Y.-C. Wang, Y.-Y. Shih, C.-C. Kuo. Fault Diagnosis of Induction Motors with Imbalanced Data Using Deep Convolutional Generative Adversarial Network. *Applied Sciences*, 2022; 12(8): 4080
5. Hejazi S., Packianather M., Liu Y. A Novel Approach Using WGAN-GP and Conditional WGAN-GP for Generating Artificial Thermal Images of Induction Motor Faults. *Procedia Computer Science*, 2023
6. Majid Handjani, Mehdi Ezodji. Detection of Electrical Faults in a Three-Phase Induction Motor Using Deep Network Features of Thermograms. Measurement. 2020
7. Sandhu, Sat. Using Thermal Imaging to Troubleshoot Motors and Drives. Fluke Corporation, Thermography Services White Paper, 2019. 12 p. [Internet]. Available from GlobalTestSupply.com.
8. J. Faiz, A. M. Takbash, E. Mazaheri-Tehrani. Application of Signal Processing Tools for Fault Diagnosis in Induction Motors—A Review—Part II. *AUT Journal of Electrical Engineering*, 2018; 50(1): 3–12.
9. Liu X., Li T., Zhang R., Wu D., Liu Y., Yang Z. A GAN and Feature Selection-Based Oversampling Technique for Intrusion Detection. *Security and Communication Networks*. 2021.
10. Andriy Karpati, Peter Abbel, Greg Brockman, Peter Chen, Vicky Chung, Rocky Duan, Ian Goodfellow, Dark Kingma, Jonathan Ho, Rein Houthuft, Tim Salimans, John Shulman, Ilya Sutzkever, Wojciech Zaremba. Generative models. OpenAI. Cited April

7, 2016.

11. Pauline Luke, Camille Cupri, Sumit Chintala, Jakob Verbeek. Semantic segmentation using adversarial networks. NIPS Workshop on Competitive Learning, December, Barcelona, Spain 2016. Bibcode:2016arXiv161108408L. arXiv:1611.08408.
12. Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicky Chung, Alec Redford, Xi Chen. "Improved Methods for GAN Training". arXiv:1606.03498 [cs.LG];
13. Andriy Karpati, Peter Abbel, Greg Brockman, Peter Chen, Vicky Chung, Rocky Duan, Ian Goodfellow, Dark Kingma, Jonathan Ho, Rein Houthuft, Tim Salimans, John Shulman, Ilya Sutzkever, Wojciech Zaremba. Generative models. OpenAI. Cited April 7, 2016.

APPENDIX I

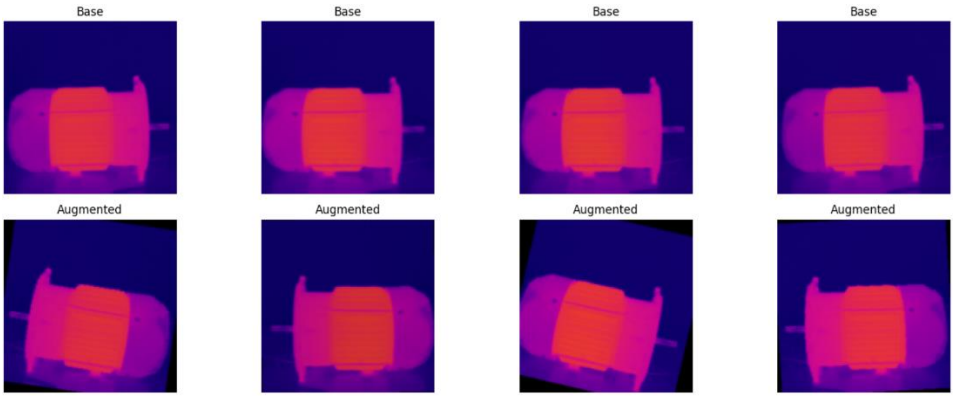
```
import numpy as np
import pandas as pd
import os

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

from PIL import Image
import matplotlib.pyplot as plt
import os

root = '/kaggle/input/thermal-images-of-induction-motor'

file_paths = []
for dirpath, dirnames, filenames in os.walk(root):
    for fname in filenames:
        if fname.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp')):
            file_paths.append(os.path.join(dirpath, fname))

print(f"Found {len(file_paths)} images\n")

for fp in file_paths[:5]:
    img = Image.open(fp)
    print(fp, "→ size:", img.size, "mode:", img.mode)
    plt.figure(figsize=(3, 3))
    plt.imshow(img)
    plt.axis('off')

plt.show()
```



```
import os
from PIL import Image
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

# 1. Оголошення Dataset зі скануванням підпапок
```

```
class ThermalDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        super().__init__()
        self.files = []
        for dirpath, _, filenames in os.walk(root_dir):
            for fname in filenames:
                if fname.lower().endswith(('.png','.jpg','.jpeg','.bmp')):
                    self.files.append(os.path.join(dirpath, fname))
        self.transform = transform

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        img = Image.open(self.files[idx]).convert('RGB')
        if self.transform:
            img = self.transform(img)
        return img

root = '/kaggle/input/thermal-images-of-induction-motor'
img_size = 128
transform = transforms.Compose([
    transforms.CenterCrop(240),
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3),
])

dataset = ThermalDataset(root, transform=transform)
print("Found images:", len(dataset))
loader = DataLoader(dataset, batch_size=32, shuffle=True, num_workers=2,
pin_memory=True)

batch = next(iter(loader))
print("Batch shape:", batch.shape)

from torchvision import transforms

img_size = 128

base_transform = transforms.Compose([
    transforms.CenterCrop(240),
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3),
])

aug_transform = transforms.Compose([
    transforms.CenterCrop(240),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.1, contrast=0.1),
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3),
])

print("✅ Transforms defined successfully!")
print("Base transform:", base_transform)
print("Augment transform:", aug_transform)
```

```
✅ Transforms defined successfully!
Base transform: Compose(
    CenterCrop(size=(240, 240))
    Resize(size=(128, 128), interpolation=bilinear, max_size=None, antialias=True)
    ToTensor()
    Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
)
Augment transform: Compose(
    CenterCrop(size=(240, 240))
    RandomHorizontalFlip(p=0.5)
    RandomRotation(degrees=[-15.0, 15.0], interpolation=nearest, expand=False,
fill=0)
    ColorJitter(brightness=(0.9, 1.1), contrast=(0.9, 1.1), saturation=None,
hue=None)
    Resize(size=(128, 128), interpolation=bilinear, max_size=None, antialias=True)
    ToTensor()
    Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])

import matplotlib.pyplot as plt
from PIL import Image
import torch

print("✅ Начинаем превью обработки")

# Скільки файлів беремо для прикладу
n = 4

fig, axes = plt.subplots(2, n, figsize=(4*n, 6))

for i in range(n):
    path = dataset.files[i]
    print(f"[{i+1}] {path}")
    img = Image.open(path).convert('RGB')

    # Базова обробка
    img_base = base_transform(img)
    # Аугментована обробка
    img_aug  = aug_transform(img)

    # Повертаємо в [0,1] для plt
    img_base = (img_base * 0.5 + 0.5).permute(1,2,0).numpy()
    img_aug  = (img_aug  * 0.5 + 0.5).permute(1,2,0).numpy()

    axes[0, i].imshow(img_base)
    axes[0, i].set_title("Base")
    axes[0, i].axis('off')

    axes[1, i].imshow(img_aug)
    axes[1, i].set_title("Augmented")
    axes[1, i].axis('off')

plt.tight_layout()
print("✅ Превью завершено!")
plt.show()
```




```
import torch
import torch.nn as nn

class Generator128(nn.Module):
    def __init__(self, z_dim=100, img_channels=3, fm=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.ConvTranspose2d(z_dim, fm*16, 4, 1, 0, bias=False),  # 1→4
            nn.BatchNorm2d(fm*16), nn.ReLU(True),

            nn.ConvTranspose2d(fm*16, fm*8, 4, 2, 1, bias=False),  # 4→8
            nn.BatchNorm2d(fm*8), nn.ReLU(True),

            nn.ConvTranspose2d(fm*8, fm*4, 4, 2, 1, bias=False),   # 8→16
            nn.BatchNorm2d(fm*4), nn.ReLU(True),

            nn.ConvTranspose2d(fm*4, fm*2, 4, 2, 1, bias=False),   # 16→32
            nn.BatchNorm2d(fm*2), nn.ReLU(True),

            nn.ConvTranspose2d(fm*2, fm,   4, 2, 1, bias=False),   # 32→64
            nn.BatchNorm2d(fm),   nn.ReLU(True),

            nn.ConvTranspose2d(fm, img_channels, 4, 2, 1, bias=False),  # 64→128
            nn.Tanh()
        )

    def forward(self, z):
        return self.net(z)

class Critic128(nn.Module):
    def __init__(self, img_channels=3, fm=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(img_channels, fm,   4, 2, 1, bias=False),  # 128→64
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(fm, fm*2, 4, 2, 1, bias=False),            # 64→32
            nn.InstanceNorm2d(fm*2, affine=True),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(fm*2, fm*4, 4, 2, 1, bias=False),          # 32→16
            nn.InstanceNorm2d(fm*4, affine=True),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(fm*4, fm*8, 4, 2, 1, bias=False),          # 16→8
            nn.InstanceNorm2d(fm*8, affine=True),
            nn.LeakyReLU(0.2, inplace=True),
```

```
            nn.Conv2d(fm*8, fm*16,4, 2, 1, bias=False),          # 8→4
            nn.InstanceNorm2d(fm*16, affine=True),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(fm*16, 1, 4, 1, 0, bias=False)              # 4→1
        )

    def forward(self, x):
        return self.net(x).view(-1)

device = 'cuda' if torch.cuda.is_available() else 'cpu'
gen = Generator128().to(device)
crit = Critic128().to(device)

z = torch.randn(8, 100, 1, 1, device=device)
fake = gen(z)
print("Generator output shape:", fake.shape)  # [8,3,128,128]

real_batch = batch[:8].to(device)
print("Critic real:", crit(real_batch).shape)  # [8]
print("Critic fake:", crit(fake).shape)        # [8]

class ThermalDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.files = []
        for dp, _, fns in os.walk(root_dir):
            for fn in fns:
                if fn.lower().endswith(('.png','.jpg','.jpeg','.bmp')):
                    self.files.append(os.path.join(dp, fn))
        self.transform = transform

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        img = Image.open(self.files[idx]).convert('RGB')
        return self.transform(img) if self.transform else img

root = '/kaggle/input/thermal-images-of-induction-motor'
img_size = 128
transform = transforms.Compose([
    transforms.CenterCrop(240),
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3),
])

dataset = ThermalDataset(root, transform)
loader  = DataLoader(dataset, batch_size=32, shuffle=True, num_workers=2,
pin_memory=True)

# 3) Моделі Generator128 і Critic128
class Generator128(nn.Module):
    def __init__(self, z_dim=100, img_channels=3, fm=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.ConvTranspose2d(z_dim, fm*16, 4, 1, 0, bias=False),
nn.BatchNorm2d(fm*16), nn.ReLU(True),
            nn.ConvTranspose2d(fm*16, fm*8, 4, 2, 1, bias=False),
nn.BatchNorm2d(fm*8),  nn.ReLU(True),
            nn.ConvTranspose2d(fm*8, fm*4, 4, 2, 1, bias=False),
nn.BatchNorm2d(fm*4),  nn.ReLU(True),
```

```
            nn.ConvTranspose2d(fm*4, fm*2, 4, 2, 1, bias=False),
nn.BatchNorm2d(fm*2),  nn.ReLU(True),
            nn.ConvTranspose2d(fm*2, fm,   4, 2, 1, bias=False),
nn.BatchNorm2d(fm),    nn.ReLU(True),
            nn.ConvTranspose2d(fm, img_channels, 4, 2, 1, bias=False), nn.Tanh()
        )
    def forward(self, z): return self.net(z)

class Critic128(nn.Module):
    def __init__(self, img_channels=3, fm=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(img_channels, fm,   4, 2, 1, bias=False),  nn.LeakyReLU(0.2,
True),
            nn.Conv2d(fm, fm*2, 4, 2, 1, bias=False),
nn.InstanceNorm2d(fm*2, affine=True), nn.LeakyReLU(0.2, True),
            nn.Conv2d(fm*2, fm*4, 4, 2, 1, bias=False),
nn.InstanceNorm2d(fm*4, affine=True), nn.LeakyReLU(0.2, True),
            nn.Conv2d(fm*4, fm*8, 4, 2, 1, bias=False),
nn.InstanceNorm2d(fm*8, affine=True), nn.LeakyReLU(0.2, True),
            nn.Conv2d(fm*8, fm*16,4, 2, 1, bias=False),
nn.InstanceNorm2d(fm*16, affine=True),nn.LeakyReLU(0.2, True),
            nn.Conv2d(fm*16, 1, 4, 1, 0, bias=False)
        )
    def forward(self, x): return self.net(x).view(-1)

# 4) Інстанціюємо
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print("Using device:", device)
gen  = Generator128().to(device)
crit = Critic128().to(device)

# 5) Gradient penalty
def gradient_penalty(critic, real, fake, device=None, lambda_gp=10.0):
    if device is None: device = real.device
    bs = real.size(0)
    alpha = torch.rand(bs, 1, 1, 1, device=device)
    interp = alpha * real + (1-alpha) * fake
    interp.requires_grad_(True)
    crit_interp = critic(interp)
    grads = autograd.grad(
        outputs=crit_interp, inputs=interp,
        grad_outputs=torch.ones_like(crit_interp, device=device),
        create_graph=True, retain_graph=True
    )[0]
    grad_norm = grads.view(bs, -1).norm(2, dim=1)
    return lambda_gp * ((grad_norm - 1)**2).mean()

# 6) Беремо батч і тестуємо GP
real_batch = next(iter(loader)).to(device)
z = torch.randn(real_batch.size(0), 100, 1, 1, device=device)
fake_batch = gen(z).detach()

print("Real batch:", real_batch.shape)
print("Fake batch:", fake_batch.shape)
gp_val = gradient_penalty(crit, real_batch, fake_batch, device=device,
lambda_gp=10.0)
print("Gradient penalty:", gp_val.item())

import torch
import torch.optim as optim
from torchvision.utils import save_image
```

```
import os

# Якщо gradient_penalty ще не визначений у цьому середовищі, додаємо:
import torch.autograd as autograd

def gradient_penalty(critic, real, fake, device='cuda', lambda_gp=10.0):
    batch_size = real.size(0)
    epsilon = torch.rand(batch_size, 1, 1, 1, device=device).expand_as(real)
    interpolated = epsilon * real + (1 - epsilon) * fake
    interpolated.requires_grad_(True)

    prob_interpolated = critic(interpolated)
    grads = autograd.grad(
        outputs=prob_interpolated,
        inputs=interpolated,
        grad_outputs=torch.ones_like(prob_interpolated, device=device),
        create_graph=True,
        retain_graph=True,
        only_inputs=True
    )[0]

    grads = grads.view(batch_size, -1)
    grads_norm = grads.norm(2, dim=1)
    gp = lambda_gp * ((grads_norm - 1) ** 2).mean()
    return gp

# Визначаємо функцію тренування WGAN-GP
def train_wgan_gp(
    gen, crit, loader,
    epochs=200,       # збільшуємо за замовчуванням до 200
    z_dim=100,
    n_critic=5,
    lr=1e-4,
    lambda_gp=10.0,
    device='cuda'
):
    # Папка для зразків
    os.makedirs('/kaggle/working/samples', exist_ok=True)

    # Оптимізатори
    opt_g = optim.Adam(gen.parameters(), lr=lr, betas=(0.5, 0.9))
    opt_c = optim.Adam(crit.parameters(), lr=lr, betas=(0.5, 0.9))

    # Фіксований шум для відстеження прогресу
    fixed_noise = torch.randn(16, z_dim, 1, 1, device=device)

    for epoch in range(1, epochs+1):
        for real in loader:
            real = real.to(device)
            bs = real.size(0)

            # --- Оновлення Critic ---
            for _ in range(n_critic):
                z = torch.randn(bs, z_dim, 1, 1, device=device)
                fake = gen(z).detach()
                loss_c = -(crit(real).mean() - crit(fake).mean())
                gp = gradient_penalty(crit, real, fake, device=device,
lambda_gp=lambda_gp)

                opt_c.zero_grad()
                (loss_c + gp).backward()
                opt_c.step()
```

```
            # --- Оновлення Generator ---
            z = torch.randn(bs, z_dim, 1, 1, device=device)
            fake = gen(z)
            loss_g = -crit(fake).mean()

            opt_g.zero_grad()
            loss_g.backward()
            opt_g.step()

        # Логи
        print(f"Epoch [{epoch}/{epochs}]  Loss_C: {loss_c.item():.4f}  Loss_G:
{loss_g.item():.4f}")

        # Зберегти зразки кожні 10 епох
        if epoch % 10 == 0:
            with torch.no_grad():
                samples = gen(fixed_noise).cpu()
                save_image(
                    samples,
                    f"/kaggle/working/samples/epoch_{epoch}.png",
                    nrow=4,
                    normalize=True,
                    value_range=(-1, 1)
                )

# Викликаємо тренування на 200 епох
train_wgan_gp(
    gen=gen,
    crit=crit,
    loader=loader,      # Ваш DataLoader для GAN
    epochs=200,         # збільшуємо до 200 епох
    z_dim=100,
    n_critic=5,
    lr=1e-4,
    lambda_gp=10.0,
    device=device       # 'cuda' або 'cpu'
)
```

```
Epoch [190/200]  Loss_C: -60.1945  Loss_G: 139.7644
Epoch [191/200]  Loss_C: -27.1802  Loss_G: 98.4751
Epoch [192/200]  Loss_C: -54.8815  Loss_G: 82.9212
Epoch [193/200]  Loss_C: -39.1331  Loss_G: 131.7898
Epoch [194/200]  Loss_C: -44.6584  Loss_G: 118.8899
Epoch [195/200]  Loss_C: -43.4668  Loss_G: 97.2197
Epoch [196/200]  Loss_C: -22.9475  Loss_G: 108.7968
Epoch [197/200]  Loss_C: -36.7555  Loss_G: 113.5974
Epoch [198/200]  Loss_C: -44.1862  Loss_G: 148.7192
Epoch [199/200]  Loss_C: -41.0505  Loss_G: 98.0587
Epoch [200/200]  Loss_C: -35.4859  Loss_G: 108.3597
```

APPENDIX 2

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
files under the input directory
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
root = '/kaggle/input/thermal-images-of-induction-motor'

# Зібрати назви підпапок (класи)
classes = sorted(
    [d for d in os.listdir(root) if os.path.isdir(os.path.join(root, d))]
)
print("Found classes:", classes)

# Побудувати словник name → idx
class_to_idx = {cls:i for i,cls in enumerate(classes)}
print("class_to_idx:", class_to_idx)

# Подивіться кількість зображень у кожній папці
for cls in classes:
    folder = os.path.join(root, cls)
    imgs = [f for f in os.listdir(folder) if
f.lower().endswith(('.png','.jpg','.bmp'))]
    print(f"{cls:20s} → {len(imgs)} images")
Found classes: ['A&B50', 'A&C&B10', 'A&C&B30', 'A&C10', 'A&C30', 'A10', 'A30', 'A50',
'Fan', 'Noload', 'Rotor-0']
class_to_idx: {'A&B50': 0, 'A&C&B10': 1, 'A&C&B30': 2, 'A&C10': 3, 'A&C30': 4, 'A10':
5, 'A30': 6, 'A50': 7, 'Fan': 8, 'Noload': 9, 'Rotor-0': 10}
A&B50                → 38 images
A&C&B10              → 31 images
A&C&B30              → 42 images
A&C10                → 31 images
A&C30                → 38 images
A10                  → 34 images
A30                  → 37 images
A50                  → 35 images
Fan                  → 28 images
Noload               → 25 images
Rotor-0              → 30 images
class ThermalClassDataset(Dataset):
    def __init__(self, root_dir, class_to_idx, transform=None):
        super().__init__()
        self.samples = []
        self.transform = transform
        for cls_name, cls_idx in class_to_idx.items():
            folder = os.path.join(root_dir, cls_name)
            for fname in os.listdir(folder):
                if fname.lower().endswith(('.png','.jpg','.bmp')):
                    fullpath = os.path.join(folder, fname)
                    self.samples.append((fullpath, cls_idx))

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        path, label = self.samples[idx]
        img = Image.open(path).convert('RGB')
        if self.transform:
```

```
            img = self.transform(img)
        return img, label

dataset = ThermalClassDataset(root, class_to_idx, transform=train_transforms)
print("Total samples:", len(dataset))
img0, lbl0 = dataset[0]
print("Sample shape:", img0.shape, "Label:", lbl0)
# Зібрати всі мітки для розподілу
all_labels = [label for (_, label) in dataset.samples]

train_idx, val_idx = train_test_split(
    list(range(len(dataset))),
    test_size=0.20,
    stratify=all_labels,
    random_state=SEED
)
print(f"Train samples: {len(train_idx)}, Val samples: {len(val_idx)}")

train_dataset = Subset(dataset, train_idx)
val_dataset = Subset(
    ThermalClassDataset(root, class_to_idx, transform=val_transforms),
    val_idx
)

train_loader = DataLoader(
    train_dataset, batch_size=32, shuffle=True, num_workers=2, pin_memory=True
)
val_loader = DataLoader(
    val_dataset, batch_size=32, shuffle=False, num_workers=2, pin_memory=True
)

imgs, labels = next(iter(train_loader))
print("Batch shapes:", imgs.shape, labels.shape)

import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1   = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2   = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3   = nn.BatchNorm2d(128)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn4   = nn.BatchNorm2d(256)
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.5)
        fc_in = 256 * (IMG_SIZE // 16) * (IMG_SIZE // 16)
        self.fc1 = nn.Linear(fc_in, 512)
        self.fc2 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))  # → [B,32,64,64]
        x = self.pool(F.relu(self.bn2(self.conv2(x))))  # → [B,64,32,32]
        x = self.pool(F.relu(self.bn3(self.conv3(x))))  # → [B,128,16,16]
        x = self.pool(F.relu(self.bn4(self.conv4(x))))  # → [B,256,8,8]
        x = x.view(x.size(0), -1)                       # → [B,256*8*8]
        x = self.dropout(F.relu(self.fc1(x)))           # → [B,512]
        x = self.fc2(x)                                 # → [B,num_classes]
        return x
```

```
model_cnn = SimpleCNN(num_classes=len(classes)).to(device)
print("SimpleCNN initialized. Output shape:",
      model_cnn(torch.randn(2,3,IMG_SIZE,IMG_SIZE,device=device)).shape)

def train_classifier(model, train_loader, val_loader, criterion, optimizer,
num_epochs=20):
    model = model.to(device)
    best_acc = 0.0

    for epoch in range(1, num_epochs + 1):
        # ---------- Train ----------
        model.train()
        running_loss = 0.0
        running_corrects = 0
        total_train = 0

        for inputs, labels in train_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)             # [B, num_classes]
            loss = criterion(outputs, labels)  # CrossEntropyLoss
            loss.backward()
            optimizer.step()

            _, preds = torch.max(outputs, 1)
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data).item()
            total_train += inputs.size(0)

        epoch_loss = running_loss / total_train
        epoch_acc  = running_corrects / total_train

        # ---------- Validate ----------
        model.eval()
        val_loss = 0.0
        val_corrects = 0
        total_val = 0

        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs = inputs.to(device)
                labels = labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                _, preds = torch.max(outputs, 1)
                val_loss += loss.item() * inputs.size(0)
                val_corrects += torch.sum(preds == labels.data).item()
                total_val += inputs.size(0)

        val_epoch_loss = val_loss / total_val
        val_epoch_acc  = val_corrects / total_val

        print(f"Epoch {epoch}/{num_epochs}  |  "
              f"Train Loss: {epoch_loss:.4f}, Train Acc: {epoch_acc:.4f}  |  "
              f"Val Loss: {val_epoch_loss:.4f}, Val Acc: {val_epoch_acc:.4f}")

        # Зберігаємо кращу модель
        if val_epoch_acc > best_acc:
            best_acc = val_epoch_acc
```

```
            torch.save(model.state_dict(),
                       f"best_resnet_epoch_{epoch}_acc_{best_acc:.4f}.pth")
train_classifier(
    model_resnet,       # модель, яку ми навчаємо
    train_loader,       # DataLoader для тренувальних даних
    val_loader,         # DataLoader для валідаційних даних
    criterion,          # CrossEntropyLoss
    optimizer_resnet,   # Adam для останнього шару ResNet
    num_epochs=30       # наприклад, 30 епох
)

from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# 1) Завантажуємо найкращу модель (якщо потрібно)
#    Вам, можливо, збереглось щось на кшталт 'best_resnet_epoch_30_acc_0.6757.pth'
best_model_path = 'best_resnet_epoch_30_acc_0.6757.pth'
model_resnet.load_state_dict(torch.load(best_model_path))
model_resnet.eval()

# 2) Збираємо всі передбачення й правдиві мітки
all_preds = []
all_labels = []

with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = model_resnet(inputs)
        _, preds = torch.max(outputs, 1)
        all_preds .extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# 3) Матриця неточностей
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d',
            xticklabels=classes, yticklabels=classes,
            cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix (Validation Set)')
plt.show()

# 4) Classification report
print(classification_report(all_labels, all_preds, target_names=classes))

import torch, torch.nn as nn
import torch.optim as optim
from torchvision import models
from sklearn.model_selection import train_test_split
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset, Subset
from PIL import Image
import os, random, numpy as np

SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if torch.cuda.is_available(): torch.cuda.manual_seed_all(SEED)
```

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print("Using device:", device)

# === 2. Dataset / DataLoader (здесь просто пример, ваш код уже есть) ===
# root = '/kaggle/input/thermal-images-of-induction-motor'
# classes = [...]
# class_to_idx = {...}
# train_loader, val_loader = ...  # код из предыдущих ячеек

# === 3. Определяем модель ResNet-18 ===
resnet = models.resnet18(pretrained=True)
resnet.fc = nn.Linear(resnet.fc.in_features, len(classes))
model_resnet = resnet.to(device)
print("ResNet loaded, output shape:",
model_resnet(torch.randn(2,3,IMG_SIZE,IMG_SIZE,device=device)).shape)

# === 4. Проверка и «разморозка» параметров ===
print("\n--- Параметры до разморозки ---")
for name, param in model_resnet.named_parameters():
    print(f"{name:30s} requires_grad={param.requires_grad}")

for name, param in model_resnet.named_parameters():
    if "layer3" in name or "layer4" in name or "fc" in name:
        param.requires_grad = True
    else:
        param.requires_grad = False

print("\n--- Параметры после разморозки ---")
for name, param in model_resnet.named_parameters():
    print(f"{name:30s} requires_grad={param.requires_grad}")

# === 5. Создаем optimizer и loss ===
criterion = nn.CrossEntropyLoss()
optimizer_resnet = optim.Adam(
    filter(lambda p: p.requires_grad, model_resnet.parameters()),
    lr=1e-5
)
print("\nOptimizer created. Количество обучаемых параметров:",
      sum(p.requires_grad for p in model_resnet.parameters()))

# === 6. Определяем функцию train_classifier (если не сделано ранее) ===
def train_classifier(model, train_loader, val_loader, criterion, optimizer,
num_epochs=20):
    model = model.to(device)
    best_acc = 0.0

    for epoch in range(1, num_epochs + 1):
        # ---------- Train ----------
        model.train()
        running_loss = 0.0
        running_corrects = 0
        total_train = 0

        for inputs, labels in train_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
```

```
            optimizer.step()

            _, preds = torch.max(outputs, 1)
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data).item()
            total_train += inputs.size(0)

        epoch_loss = running_loss / total_train
        epoch_acc  = running_corrects / total_train

        # ---------- Validate ----------
        model.eval()
        val_loss = 0.0
        val_corrects = 0
        total_val = 0

        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs = inputs.to(device)
                labels = labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                _, preds = torch.max(outputs, 1)
                val_loss += loss.item() * inputs.size(0)
                val_corrects += torch.sum(preds == labels.data).item()
                total_val += inputs.size(0)

        val_epoch_loss = val_loss / total_val
        val_epoch_acc  = val_corrects / total_val

        print(f"Epoch {epoch}/{num_epochs}  |  "
              f"Train Loss: {epoch_loss:.4f}, Train Acc: {epoch_acc:.4f}  |  "
              f"Val Loss: {val_epoch_loss:.4f}, Val Acc: {val_epoch_acc:.4f}")

        if val_epoch_acc > best_acc:
            best_acc = val_epoch_acc
            torch.save(model.state_dict(),
                       f"best_resnet_epoch_{epoch}_acc_{best_acc:.4f}.pth")

    print("Finished training. Best Val Acc: {:.4f}".format(best_acc))

# === 7. Запуск тренировки (в этой ячейке появится вывод) ===
train_classifier(
    model_resnet,
    train_loader,
    val_loader,
    criterion,
    optimizer_resnet,
    num_epochs=20
)

from sklearn.metrics import confusion_matrix, classification_report
import torch

# 1) Завантажуємо найкращі ваги (припускаючи, що вони збережені у файлі)
best_model_path = 'best_resnet_epoch_20_acc_0.8784.pth'
model_resnet.load_state_dict(torch.load(best_model_path))
model_resnet.eval()

all_preds = []
all_labels = []
```

```
# 2) Пройдемося по валідації
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = model_resnet(inputs)
        _, preds = torch.max(outputs, 1)
        all_preds .extend(preds.cpu().tolist())
        all_labels.extend(labels.cpu().tolist())

# 3) Роздрукуємо основні метрики
print("Confusion Matrix:")
print(confusion_matrix(all_labels, all_preds))

print("\nClassification Report:")
print(classification_report(all_labels, all_preds, target_names=classes))
```

```
Confusion Matrix:
[[8 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 5 0 0 0 0]
 [0 0 7 0 0 0 0 1 0 0 0]
 [0 0 0 6 0 0 0 0 0 0 0]
 [0 0 0 0 8 0 0 0 0 0 0]
 [0 0 0 0 0 7 0 0 0 0 0]
 [0 0 0 0 0 0 7 0 0 0 0]
 [0 0 0 0 0 0 0 7 0 0 0]
 [0 0 0 0 0 0 0 0 6 0 0]
 [0 0 0 0 0 0 3 0 0 2 0]
 [0 0 0 0 0 0 0 0 0 0 6]]

Classification Report:
              precision    recall  f1-score   support

       A&B50       1.00      1.00      1.00         8
     A&C&B10       1.00      0.17      0.29         6
     A&C&B30       1.00      0.88      0.93         8
       A&C10       1.00      1.00      1.00         6
       A&C30       1.00      1.00      1.00         8
         A10       1.00      1.00      1.00         7
         A30       0.47      1.00      0.64         7
         A50       0.88      1.00      0.93         7
         Fan       1.00      1.00      1.00         6
      Noload       1.00      0.40      0.57         5
     Rotor-0       1.00      1.00      1.00         6

    accuracy                           0.88        74
   macro avg       0.94      0.86      0.85        74
weighted avg       0.94      0.88      0.87        74
```

```
import matplotlib.pyplot as plt
import numpy as np

# Функція, щоб “розпалати” тензор-образ у форматі numpy 0..1
def tensor_to_numpy(img_tensor):
    # img_tensor має розміри [3, IMG_SIZE, IMG_SIZE] і нормалізований (mean=0.5,
std=0.5)
    img = img_tensor.permute(1, 2, 0).cpu().numpy()  # [H, W, C]
    img = (img * 0.5) + 0.5  # «рознормалізуємо» (якщо нормалізація була mean=0.5,
std=0.5)
    img = np.clip(img, 0, 1)
    return img
```

```
# Витягнемо один батч із валідаційного DataLoader
inputs, labels = next(iter(val_loader))
inputs = inputs.to(device)
labels = labels.to(device)

# Отримаємо передбачення
model_resnet.eval()
with torch.no_grad():
    outputs = model_resnet(inputs)
    _, preds = torch.max(outputs, 1)

# Візуалізуємо 16 зображень (або менше, залежно від batch_size)
batch_size = inputs.size(0)
num_show = min(16, batch_size)

plt.figure(figsize=(12, 12))
for i in range(num_show):
    ax = plt.subplot(int(np.sqrt(num_show)), int(np.sqrt(num_show)), i + 1)
    img = tensor_to_numpy(inputs[i])
    ax.imshow(img)
    ax.set_title(f"T: {classes[labels[i].item()]}\nP: {classes[preds[i].item()]}",
fontsize=8)
    ax.axis("off")
plt.tight_layout()
plt.show()
```