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: Semi-supervised learning for hyperspectral regression

Performer: student of FAET-151-21-1 group Oleksii Chebanu

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: Напівкероване навчання для гіперспектральної регресії

Виконавець: студент групи Ба-151-21-1-ІЗ Чебану Олексій Юрійович

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

Чебану Олексія Юрійовича

1. Тема роботи : “Напівкероване навчання для гіперспектральної регресії”.
2. Термін виконання роботи : з 29.04.2024р. до 03.06.2024р.
3. Вихідні дані до роботи : Процес обробки гіперспектральних зображень, опис типів, особливостей та застосування рекурентних нейронних мереж (РНМ).
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1. Загальний аналіз систем обробки гіперспектральних зображень. 2. Аналіз методів зменшення вимірності та видалення шуму. 3. Аналіз рекурентних нейронних мереж (РНМ) для обробки послідовних даних. 4.Аналіз методології проектування та архітектури інтелектуальних систем обробки зображень. 5. Аналіз програмного забезпечення для обробки гіперспектральних зображень. 6. Аналіз методів класифікації та інтерпретації гіперспектральних даних.

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

6. 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- 03.06.2024	Виконано

Дата видачі завдання: 25 березня 2024 р.

Керівник дипломної роботи _____Синєглазов В.М.

(підпис)

Завдання прийняв до виконання _____Чебану О. Ю

(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

Chebanu Oleksii Y.

1. Work topic: "Semi-supervised learning for hyperspectral regression".

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

3. Input data for the work: The process of processing hyperspectral images, description types, features and applications of recurrent neural networks (RNNs).

4. Content of the explanatory note (list of issues to be developed):

1. General analysis of hyperspectral image processing systems. 2. Analysis of dimensionality reduction and noise removal methods. 3. Analysis of recurrent neural networks (RNN) for processing sequential data. 4. Analysis of the design methodology and architecture of intelligent image processing systems. 5. Analysis of software for processing hyperspectral images. 6. Analysis of methods of classification and interpretation of hyperspectral data.

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

6. 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- 03.06.2024	Done

Issue data of the task : 25 March 2024

Supervisor : _____ Viktor SINEGLAZOV

(sign)

The task was accepted by : _____ CHEBANU O. Y.

(sign)

ABSTRACT

Thesis: 92 pp., 17 figures, 3 appendices, 16 sources.

Keywords: MACHINE LEARNING, DEEP LEARNING, NEURAL NETWORKS, CONVOLUTION NEURAL NETWORKS, HYPERSPECTRAL IMAGES CLASSIFICATION

The object of research is the classification of hyperspectral images. Pattern recognition technologies are widely used nowadays. The use of these technologies for the recognition of hyperspectral images can greatly facilitate work in many fields.

The purpose of the diploma project: The purpose of this project is to study the existing technologies of graphic object recognition and artificial neural networks, as well as the construction of a convolutional neural network for the recognition of graphic objects.

Main tasks: Development and successful training of a convolutional neural network for graphic object recognition tasks.

The essence of the project: Research of existing methods of recognizing graphic objects using artificial neural networks, identifying shortcomings and proposing ways to overcome them. The use of already studied technologies to build a convolutional neural network to solve the problems of recognizing graphic objects.

Practical use: Can be used in any graphical object detection or recognition system, the proposed performance improvement techniques will help support more accurate classification, detection, and recognition.

Key metrics and results: Recognition of graphic images has long attracted the attention of specialists in the field of applied mathematics, and then computer science. The main problem of graphic object recognition is the human ability to classify and generalize objects - this mechanism is still a huge challenge for researchers trying to solve this problem with the help of artificial neural networks. The goal of the project is to search and analyze existing methods of recognizing graphic objects using artificial neural networks. A basic model of a neural network intended for solving pattern recognition problems has been developed, and methods of improving it productively have been proposed.

CONTENT

LIST OF SYMBOLS, ABBREVIATIONS, TERMS

ACAM - content-addressable associative memory

ADALINE - adaptive linear element (formerly adaptive linear neuron)

ANN - artificial neural network

API - application programming interface

AdaBoost - adaptive boosting

CIFAR - Canadian Institute for Advanced Research

CNN - convolutional neural network

CPU - central processing unit

DL - deep learning

FFT - fast Fourier transform

FNN - forward neural network

GPU - graphics processor

gRPC - remote procedure call

HASYv2 - Handwritten Symbols version 2

HOG - histograms of oriented gradients

HPF - high pass filter

HSV - color model "Hue, Saturation, Value" (hue, saturation, value)

LPF - low-pass filter

MADALINE is a multi-layered ADALINE network

ML - machine learning

MNIST - Joint National Institute of Standards and Technology

NIST - National Institute of Standards and Technology

OpenCV is an open source computer vision library

PRP is a pattern recognition problem

RBF network - radial base network

RGB - "Red, Green, Blue" - color model

RNN - recurrent neural network

ReLU - rectified linear unit

SIFT - scale-invariant feature transformation

STL - self-study

SURF - Accelerated reliable functions

INTRODUCTION

Hyperspectral imaging is a powerful remote sensing method that collects detailed information about the Earth's surface. It measures light reflectance in hundreds of narrow spectral bands, providing a rich data set for materials identification and characterization, biophysical parameter estimation, and environmental health monitoring.

In engineering, such as geophysics, hyperspectral imaging can be used to analyze the composition of rocks around mineral deposits. Using full spectral analysis, we can predict the presence of valuable minerals such as gold, silver or copper based on the spectral characteristics of the samples. Semi-supervised learning allows us to use both labeled and unlabeled data, greatly increasing model accuracy and enabling the discovery of new patterns and anomalies in the data.

A number of studies are devoted to the use of supervised learning for hyperspectral regression. A comprehensive literature review is provided that discusses the different types of supervised learning algorithms used for hyperspectral regression, as well as the challenges and opportunities associated with this approach.

The most common type of supervised learning algorithm used for hyperspectral regression is linear regression. It is a simple but effective algorithm that can model the relationship between a continuous target variable and a set of input features.

Other types of supervised learning algorithms used for hyperspectral regression include support vector machines (SVMs) and neural networks. These algorithms are more complex than linear regression, but can be more accurate in some cases.

There are a number of challenges associated with using supervised learning for hyperspectral regression:

High dimensionality of hyperspectral data: Hyperspectral data typically consists of hundreds of bands, making it difficult to train accurate models.

Curse of dimensionality: The accuracy of machine learning models can decrease as the dimensionality of the data increases.

Lack of labeled data: Supervised learning algorithms require labeled data for training. However, collecting labeled data for hyperspectral regression can be difficult and expensive.

Despite the challenges, there are opportunities for supervised learning for hyperspectral regression:

Development of new algorithms: New algorithms are being developed to solve hyperspectral regression problems.

Availability of more data: The availability of more data, such as from satellite sensors, enables more accurate models to be learned.

Growing Use of Machine Learning: The growing use of machine learning in various industries creates a demand for skilled professionals who can use supervised learning for hyperspectral regression.

CHAPTER 1
RESEARCH OF THE TOPIC

Hyperspectral imaging is a powerful imaging technique that is widely used in various scientific and practical fields, such as Earth remote sensing, agriculture, environmental monitoring, medical diagnostics, and many others. Let's consider the physical basis of creating hyperspectral images.

1.1. Physical basis of creating hyperspectral images

1.1.1. Basic concepts

Hyperspectral imaging is the process of obtaining images in a wide spectral range that includes hundreds of narrow spectral bands. Each image point (pixel) contains spectral information in different wavelengths, which allows determining the material and chemical properties of objects.

1.1.2. Principle of operation of hyperspectral sensors

Hyperspectral sensors consist of the following main components:

Source of radiation: sunlight, artificial radiation sources or the thermal radiation of objects.

Optical system: lenses or mirrors that focus light onto the entrance hole of the spectrometer.

Spectrometer: a device that decomposes incoming light into its constituent spectral components. The following principles of spectral decomposition are used:

Diffraction gratings: the main element of a spectrometer that decomposes light into a spectrum.

Prisms: used to refract light into different wavelengths.

Detectors: register light intensity in different spectral bands.

1.1.3. Principles of spectral decomposition

Diffraction: based on the phenomenon of light diffraction on diffraction gratings. This allows you to decompose light into components, similar to the decomposition of light into colors through a prism.

Interference: interference filters are used to select narrow spectral bands.

Absorption: Some materials selectively absorb light of certain wavelengths, allowing them to be identified in a spectral range.

1.1.4. Formation of a hyperspectral image

Data collection of the hyperspectral sensor is carried out by two methods: flat scanning (Whiskbroom) and broadband scanning (Pushbroom). Flat scanning involves moving the sensor along a line to capture successive lines, while broadband scanning allows capturing the entire line of the scene simultaneously while moving the sensor along the object. Calibration includes correction of data to exclude the influence of the atmosphere, changes in lighting and other factors. Data processing includes various methods of spectral data analysis to obtain information about objects (Fig. 1.1).

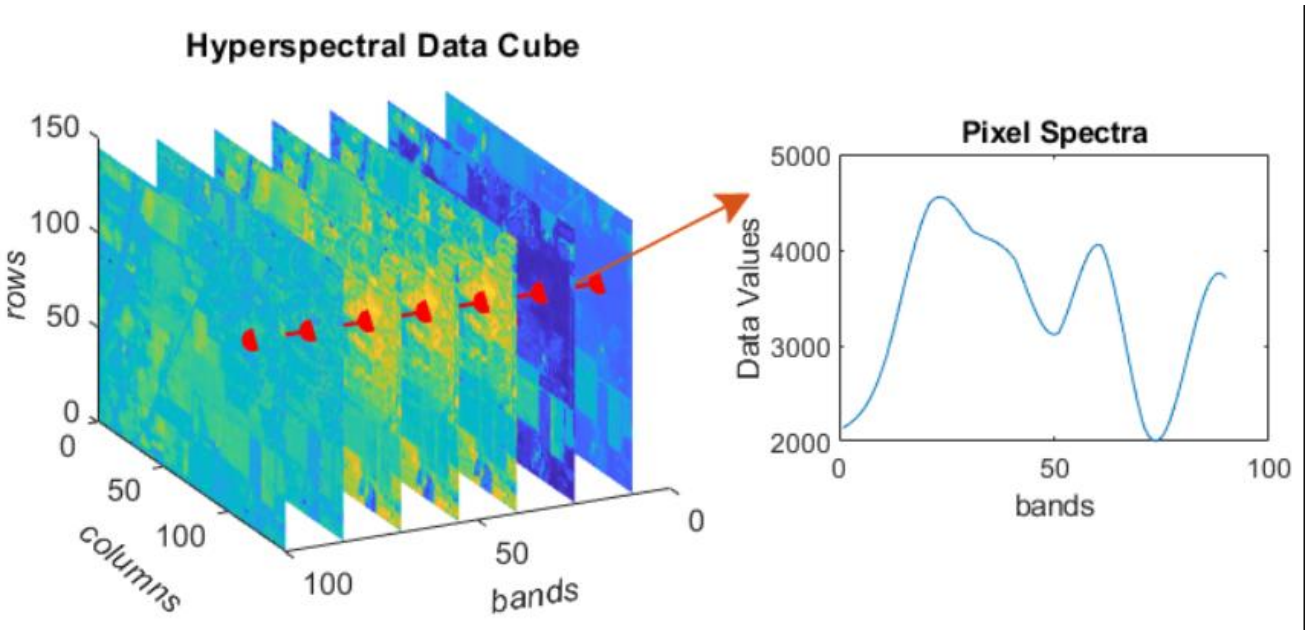

Fig 1.1

1.1.5. Advantages of hyperspectral imaging

High spectral resolution: allows you to distinguish between objects with very similar spectral characteristics.

Multifunctionality: the possibility of application in various fields from remote sensing to medical diagnostics.

Accuracy and reliability: allows you to get detailed and accurate data about objects.

1.1.6. Challenges and Limitations

Large volume of data: hyperspectral images have a significant volume, which requires efficient methods of storage and processing.

High cost of equipment: hyperspectral sensors and processing equipment are quite expensive.

Need for specialized software: Processing and analyzing hyperspectral data requires specialized programs and algorithms.

Hyperspectral imaging is a complex, but very effective method of obtaining data with a high spectral resolution, which allows obtaining detailed information about the material and chemical properties of objects, in particular, in the tasks of remote sensing and other applied fields.

The hyperspectral camera is a key tool for obtaining hyperspectral images. It consists of several main components that provide collection, separation and registration of spectral information. Let's consider the structure and principles of operation of a hyperspectral camera.

1.2. Topology of hyperspectral images

1.2.1. Defining the topology of hyperspectral data

The topology of hyperspectral images includes the structure and relationship of spectral and spatial data components. It is a study of the shape, connections and interactions between pixels and their spectral signatures.

1.2.2. Spectral topology

The spectral topology determines the structure of spectral signatures. Each pixel of a hyperspectral image has a spectral vector, which is a unique "signature" of the material. Spectral topology analysis includes:

Spectral Libraries: Collection and storage of spectral signatures of known materials.

Spectral distance and similarity: Using various metrics to determine the similarity between spectral signatures, such as cosine similarity, Mahalanobis distance, and Euclidean distance.

Spectral segmentation: Division of the image into areas with similar spectral

characteristics.

1.2.3. Spatial topology

Spatial topology covers the study of the distribution of spectral data in space. It includes:

Texture Analysis: Estimating the texture characteristics of an object's surface using texture analysis techniques such as gray level co-occurrence matrix (GLCM).

Anomaly detection: Detection of deviations in spectral data that may indicate the presence of unknown or unusual materials.

Spatial Filter: Apply filters to highlight spatial features such as edge elements, boundaries, and shapes.

1.2.4. Integration of spectral and spatial topology

Integration of spectral and spatial topology ensures:

Improved classification: Using both spectral and spatial information to improve classification accuracy.

Multiscale analysis: Looking at data at different spatial scales to identify both fine details and general patterns.

Joint segmentation: Using methods that take into account both spectral and spatial characteristics to divide an image into semantically meaningful regions.

1.2.5. Using topology in semi-supervised learning

In semi-supervised learning, topology helps create models that learn efficiently on a limited amount of labeled data:

Spectral-spatial regularization: Using topology information to improve model stability and accuracy.

Spatial-spectral graph neural networks: Using graph structures to take into account spectral-spatial relationships in the learning process.

Pseudo Labels: Using unlabeled data with high classifier confidence for further training.

1.3. Topology of the mathematical model of layers

1.3.1. Basic concepts of the topology of the mathematical model of layers

The topology of the mathematical model of the layers describes the structure, organization and interaction of the different layers in the system. This includes several key aspects:

Defining relationships between layers: Defines how different layers are related to each other. Connections can be direct or indirect, unilateral or bilateral. In some models, there are hierarchical relationships where one layer can influence another.

Properties of layers: Each layer has its own unique properties that affect its functioning and interaction with other layers. These can be physical properties (eg thickness, density), functional properties (role in the system) and dynamic properties (changes over time).

Ways of interaction: Describes how the layers interact with each other. This may involve the transfer of materials, energy or information between layers. The interaction can be described by mathematical equations that model these processes.

Spatial relations: Describes how the layers are located relative to each other in space. This can include both 2D and 3D relationships, as well as their orientation and location.

Temporal Aspects: Considers how properties and interactions of layers change over time. This is important for dynamic systems where the states of the layers can evolve.

System Constraints: Defines the constraints and rules governing the behavior of layers and their interaction. These can be physical laws, technical limitations or set parameters of the system.

Modeling and Simulation: Layered topology also includes modeling and simulation techniques used to study system behavior. This allows analysis, forecasting and optimization of the system.

For example, in a geological model of the layers of the earth's crust, topology may include determining the location of different geological layers, their thickness, density, rock properties, and the ways in which these layers interact through tectonic processes, heat and material exchange.

In computer networks, the layer topology describes the interaction of different protocol layers, such as the physical layer, the data layer, the network layer, and so on, defining how data is transmitted and processed through these layers.

Thus, the topology of the mathematical model of layers is critically important for understanding and modeling complex systems in various fields of science and technology.

1.3.2. The structure of layers in the mathematical model

Each mathematical model consists of several layers that can perform different functions. The main types of layers include:

Input layers: Receive input data and pass it on to subsequent layers.

Hidden layers: Perform the computations that allow the model to learn from the input data.

Output layers: Generate the final result of the model.

1.3.3. Interaction of layers in the mathematical model

Layer interactions in a mathematical model describe how data is transferred and processed between layers. This includes:

Direct communication: Data is passed from one layer to the next in the sequence.

Feedback: Data is fed back from the output layers to the input layers to correct errors in the learning process.

Layer Skipping: Some models can pass data through multiple layers at once to speed up calculations.

1.3.4. Types of topologies in mathematical models of layers

There are different types of topologies used in mathematical layer models:

Sequential topology: The layers are arranged in a linear sequence, each layer receives data from the previous one and transmits it to the next one.

Branched Topology: Data can be branched and processed in parallel across multiple subsets of layers.

Loop Topology: Layers can have feedback loops that allow data to circulate between layers for more complex processing.

1.3.5. Using layer topology in semi-supervised learning

Semi-supervised learning uses layer topology to build efficient models with a limited amount of labeled data:

Autoencoders: Use a symmetric topology to train data representations, which allows you to reduce dimensionality and highlight the main characteristics.

Hybrid models: Combine different types of layers and their topologies to improve the quality of learning on small sets of labeled data.

1.3.6. Examples of application of layer topology in hyperspectral regression

The topology of layers in hyperspectral regression can include:

Deep Neural Networks: Using multilayer neural networks to build complex models that take into account spectral and spatial information.

Recurrent Neural Networks: Using a loop topology to process sequential data such as time series of spectral measurements.

Convolutional neural networks: Applying spatio-spectral data processing to extract key features and improve the accuracy of regression models.

Studying the topology of mathematical layer models provides a better understanding of how the complex models used in hyperspectral regression and other machine learning tasks are built and function.

1.4. Classification of hyperspectral images

1.4.1. Basic concepts of classification of hyperspectral images

Classification of hyperspectral images consists in identifying and dividing image pixels into classes based on their spectral signatures. The main stages of classification include data pre-processing, feature extraction, model training and evaluation of the results.

1.4.2. Classification methods

There are several methods of classification of hyperspectral images:

Supervised methods: Use labeled data to train models such as nearest neighbor (k-NN), support vector method (SVM), neural networks, and decision trees.

Unsupervised methods: Cluster data without labeled examples, including k-means, hierarchical clustering, and self-learning map (SOM) algorithms.

Semi-supervised methods: Combine labeled and unlabeled data to improve classification accuracy, including graph methods and semi-supervised neural networks.

1.4.3. Pre-processing of data

Data pre-processing is an important step in the data analysis process, which ensures its accuracy, reliability and readiness for further analysis. This process involves several key steps.

Normalization is aimed at reducing the influence of changes in lighting, atmospheric conditions and other factors on spectral data. For this, spectral normalization methods are used, for example, each spectrum can be divided by the average value or the maximum intensity for normalization. Standardization is also used to bring data to a standard form, where each value is subtracted from the mean and divided by the standard deviation to ensure that the data has a mean of zero and a standard deviation of one.

Noise removal aims to reduce the influence of unwanted signals and artifacts that can distort the results of the analysis. For this, various filters are used, such as a median filter to remove impulse noise and a Gaussian filter to smooth the data. Principal component analysis (PCA) is also used to reduce the dimensionality of the data and highlight the principal components containing the largest proportion of information, which allows to reduce the influence of noise and focus on significant variations in the data.

Calibration ensures compliance of data with real physical measurements and elimination of systematic errors. Spectral calibration includes the correction of spectral shifts and offsets to ensure measurement accuracy, for example, using known reference spectra to calibrate measuring devices. Geometric calibration is used to correct spatial distortions in the data, which may include image alignment and correction of geometric artifacts.

1.4.4. Identification of features

Feature extraction is an important stage in the processing of spectral data, as it allows you to transform the initial data into a more compact and informative set of features, which facilitates their further analysis and classification.

Principal component analysis (PCA) is one of the main methods for reducing the

dimensionality of data. It allows you to highlight the main components that contain the largest share of variation in the data. This is achieved by an orthogonal transformation, which projects the data onto a new, lower-dimensional space. The principal components are linear combinations of the initial variables that maximally explain the variance of the data. Using PCA helps reduce noise, improve computational efficiency, and reveal underlying structures in data.

Independent component analysis (ICA) is another powerful feature extraction method that focuses on extracting independent sources of information from multivariate signals. Unlike PCA, which focuses on maximizing the variance, ICA aims to make the components statistically independent. This is especially useful in cases where signals are mixed and need to isolate their individual sources. ICA finds applications in many fields, including pattern recognition, image analysis, and biomedical signal processing.

Wavelet transform is a method that allows you to distinguish features on different scales. This is done by decomposing the signal into its component frequencies with different levels of detail. The wavelet transform has an advantage over traditional frequency analysis methods such as the Fourier transform because it provides simultaneous analysis in both the time and frequency domains. This makes it particularly useful for processing data that has local features or variations. Wavelet transformation allows you to highlight local anomalies, edge effects and other important characteristics, which improves the accuracy of object classification and recognition.

These methods, along with other feature extraction approaches, provide an efficient way to transform spectral data into a form that is more amenable to further analysis. This includes reducing the dimensionality of data, extracting underlying structures, increasing computational efficiency, and improving classification accuracy. Feature extraction is a critical step in many applications, from remote sensing to biomedical research and machine learning.

1.4.5. Model training

Model training is an important step in building an effective classifier, which includes parameter optimization based on training data. This process includes several key approaches, such as cross-validation and regularization, that help improve the accuracy and generalizability of the model.

Cross-validation is one of the main methods for evaluating model performance. It involves dividing the initial data set into several subsets or "folds". Each fold is used as the test set once, while the other folds are used as training data. The most common variant is k-fold cross-validation, where the data is divided into k equal parts. This method allows you to get a more stable and reliable estimate of the accuracy of the model, since each part of the data is used for both training and testing. Based on the results of cross-validation, the average performance of the model can be estimated and possible problems such as overtraining or undertraining can be detected.

Regularization is another important tool used to prevent model overtraining. Overtraining occurs when a model fits the training data too well, but generalizes poorly to new, unknown data. Regularization helps avoid this problem by adding penalty functions to the loss function. These penalties impose constraints on the model's parameters, forcing it to be smoother and simpler. There are different types of regularization, including L1 (lasio) and L2 (ridge) regularization. L1 regularization contributes to the emergence of rare models in which some coefficients can become zero, which helps in feature selection. L2 regularization, on the contrary, tends to a uniform reduction of all coefficients, which makes the model more resistant to noise in the data.

Model training may also include other optimization techniques, such as gradient descent, which is used to minimize the loss function and find optimal model parameters. An important aspect is the tuning of model hyperparameters, such as learning rate, regularization parameters, and number of iterations, which can be done using cross-validation techniques.

Another approach is to use ensemble methods, such as random forests or boosting, which combine multiple models to improve the overall accuracy and robustness of the

classifier. These methods help to reduce the risk of overtraining and increase the generalization ability of the model because they take into account different aspects of the training data.

Thus, model training includes a comprehensive approach to parameter optimization, performance evaluation, and prevention of overtraining, which allows creating an efficient and reliable classifier. This is a critical step in machine learning that determines the success of the model in real-world applications.

1.4.6. Evaluation of results

Evaluation of the classification results is important for determining the efficiency and reliability of the constructed model. It includes the use of different methods and metrics to evaluate different aspects of classifier performance.

The confusion matrix is one of the key tools for analyzing classification results. It provides information on the number of correctly and incorrectly classified instances of each class. Based on the confusion matrix, metrics such as accuracy, completeness, specificity, and F-measure can be calculated. These metrics provide detailed information about the classifier's performance, taking into account both correctly classified instances and errors.

ROC curves (receiver operating characteristic curves) and area under the curve (AUC) are used to evaluate the ability of a classifier to discriminate between two classes. The ROC-curve reflects the relationship between sensitivity (completeness) and specificity of the classifier when changing the decision threshold. The AUC represents the area under the ROC curve and indicates the overall ability of the classifier to distinguish between classes: the higher the AUC, the better the performance of the model.

Cross-validation helps to assess the stability and generalizability of the classifier. Cross-validation involves retraining and testing the model on different subsets of the data. This process allows you to obtain a statistically significant estimate of the model's performance and test how well it generalizes to new, unknown data.

The overall evaluation of the classification results includes the analysis of these and other metrics, and also takes into account the context of the classifier's application and the features of the data. The use of various evaluation methods allows you to get a complete

picture of the efficiency and reliability of the classifier and make informed decisions about its further use.

1.5. Statement of the problem

1.5.1. Definition of the problem

The problem statement in the use of hyperspectral images includes identifying the problems and challenges that researchers and practitioners face in analyzing these data for classification and regression.

One of the main problems is the high dimensionality of the data, which is inherent in hyperspectral images. Hundreds of spectral channels in such images lead to the "curse of dimensionality" when the number of features is large and the number of labeled data is small. This complicates data analysis and processing and can lead to overtraining of models.

The second problem is the limited amount of labeled data. Collecting labeled data is an expensive and time-consuming process, which limits the availability of a large number of training examples. This makes training the models more difficult and may lead to an underestimation of their performance.

The third challenge is the presence of noise and artifacts in the spectral data, which can negatively affect the accuracy of the models. This can be caused by various factors, such as measurement errors, atmospheric effects or inhomogeneities on the surface of objects.

Finally, complex spectral-spatial relationships present a challenge to the effective use of both spectral and spatial information. Analysis of such relationships requires the development and use of specialized methods and algorithms, which can be a difficult task.

To solve these challenges, it is necessary to use innovative approaches to the processing and analysis of hyperspectral data, such as the development of new dimensionality reduction methods, the use of data augmentation techniques, and the improvement of filtering and noise removal methods. In addition, the possibilities of using deep learning and artificial intelligence to automate the analysis and classification of hyperspectral data should be actively explored.

1.5.2. Purpose of work

The purpose of this work is to develop semi-supervised learning methods for hyperspectral regression taking into account the topology of hyperspectral images and the mathematical model of layers. The main tasks include:

Analysis of the topology of hyperspectral data, which involves the study of spectral and spatial relationships in hyperspectral images. This will help to understand the structure and properties of the data and improve the performance of the models.

Development of semi-supervised learning models, which consists in using a limited amount of labeled data to train models with high accuracy. This is important to solve the problem of limited resources and costs of collecting labeled data.

Integration of spatial and spectral information, which involves the use of methods that take into account both the spectral and spatial topology of the data. This helps to gain a comprehensive understanding of the objects in the images and improves the accuracy of the regression results.

The overall goal is to develop new and improve existing methods for analyzing and processing hyperspectral data to achieve more accurate and reliable results in hyperspectral regression.

1.5.3. Tasks of the research

Literature review: Analysis of existing methods of classification and regression of hyperspectral images, as well as methods of semi-supervised learning. This stage involves researching innovative approaches and identifying best practices in the industry.

Study of topology of hyperspectral images: Determination of spectral and spatial topology of data, development of methods for their analysis and use. This includes studying the relationships between various spectral and spatial characteristics of objects in images.

Development of a mathematical model of layers: Creation of a topology of a mathematical model of layers for processing hyperspectral data. This stage involves the development of algorithms and methods for processing and analyzing hyperspectral data based on their topology.

Development of semi-supervised methods: Development and implementation of

semi-supervised learning methods for hyperspectral regression. This includes the development of algorithms and models that are able to work with a limited amount of labeled data.

Experimental verification: Conducting experiments on real hyperspectral data to evaluate the effectiveness of the proposed methods. This stage involves the use of real data to test the developed algorithms and models.

Analysis of results: Assessment of accuracy and efficiency of the developed methods, comparison with existing approaches. This includes statistical analysis of the obtained results and their interpretation in order to identify the advantages and disadvantages of the proposed methods compared to others.

1.5.4. Expected Results

Expected work results include the following:

New methods of topology analysis of hyperspectral data, which allow efficient use of spectral and spatial information. These methods can help in understanding and exploiting the complex relationships between the spectral and spatial characteristics of objects.

Development of mathematical models of layers that take into account the specifics of hyperspectral data. These models can be useful for improving the processing and analysis of hyperspectral images.

New semi-supervised algorithms for hyperspectral regression that provide high accuracy with a limited amount of labeled data. These algorithms can become an effective tool for solving the problem of lack of labeled data in hyperspectral analysis.

Experimental evidence of the effectiveness of the proposed methods on real hyperspectral data. This will allow to confirm the suitability of the developed methods for real tasks of hyperspectral image analysis in various fields of application.

Therefore, the work is aimed at solving the key problems of hyperspectral regression and classification, which will improve the accuracy and efficiency of hyperspectral image analysis in various fields of application.

1.6. Conclusion

The processing of hyperspectral images involves a complex and multi-step process that allows you to transform raw spectral data into useful information. This process encompasses refinement, feature extraction, classification, and visualization, each of which is critical to the accuracy and efficiency of the analysis. Due to advanced processing methods, hyperspectral images are widely used in various fields, including remote sensing, agriculture, medicine, and environmental monitoring.

CHAPTER 2

CONVOLUTIONAL NEURAL NETWORKS AND THEIR CHARACTERISTICS

2.1. Convolutional neural network and its composition

A Convolutional Neural Network (CNN) is a specialized type of deep neural network that is particularly effective for processing spatially structured data such as images. CNNs automatically extract features from input data through the use of convolutional layers, making them extremely powerful in pattern recognition, image classification, natural language processing, and many other tasks.

2.1.1 The main components of a convolutional neural network

CNN consists of several main types of layers, each of which performs a specific function in the process of data processing and analysis:

The convolutional layer is a key component of convolutional neural networks (CNN). Its main function is to perform a convolution operation between the input data and a set of filters called convolution kernels. Each convolution kernel learns to extract certain features from the input data, such as edges, textures, or other details. After applying the convolution to the input image, a feature map is obtained, which contains the responses of the filters on different areas of the image.

The Activation Layer provides nonlinearity to the model, which allows it to learn complex dependencies in the data. The most common activation functions include ReLU (Rectified Linear Unit), Leaky ReLU, Sigmoid, and Tangent.

The Pooling Layer is used to reduce the dimensionality of feature maps, which helps reduce the number of parameters in the model and reduce computational costs. The most common pooling methods are maximum and average pooling. They allow you to highlight the most important features and reduce the dimensionality of the data.

The normalization layer (Normalization Layer) is used to stabilize the learning process and accelerate the convergence of the model. Examples of normalization methods include batch normalization and layer normalization.

A fully connected layer (Fully Connected Layer) connects each neuron with all neurons of the previous layer. It is used for final processing of features and classification.

The regularization layer (Regularization Layer) is used to prevent retraining of the model by accidentally turning off neurons during training (drop-out) or applying penalties to large weights of the model (L2 regularization).

2.1.2. Architecture of a convolutional neural network

Convolutional neural networks usually consist of alternating convolutional and pooling layers, followed by one or more fully connected layers. Here is an example of a typical CNN architecture:

Convolutional Neural Networks (CNNs) are an important component of deep learning, especially for image processing. By allowing the automatic extraction of useful features from input data, they are able to achieve impressive performance in the tasks of classification, object detection, image segmentation, and many others.

The typical CNN architecture that has been described can be extended with additional layers to improve the model's capabilities. For example, additional convolutional layers can be added to detect higher-level features or convolutional layers with different kernel sizes to detect features at different scales. After that, additional pooling layers can be added to further reduce the dimensionality of the data and work with more abstract features. Regularization techniques such as dropout can also be used to prevent overtraining of the model.

The fully connected layers that follow the convolution and pooling layers allow the model to use the detected features to perform classification or regression. These layers can be more complex, include more neurons, or use different activation functions, depending on the specific task.

One of the important aspects of the CNN architecture design process is the balance between the power of the model and its computational complexity. Adding more layers can improve the generalization ability of the model, but can also increase the training time. Therefore, when designing the architecture, it is important to carefully consider each added layer and its impact on the model's performance.

Convolutional networks were inspired by biological processes in the case of a neuron connection patterns. They are also known as shift invariants because they are based on the

joint weights architecture. Unlike MLP (multilayer perceptron), which uses a fully connected layer, CNNs use different layers to detect patterns among images, which are then passed on. ANNs have few connected layers and use a matrix as input, while MLP uses vectors as input.

As you know, each image consists of a certain number of pixels. We analyze the effect of neighboring pixels on the image using a so-called filter (also called a kernel).

A filter is a tensor that tracks spatial information and learns to extract features such as edges, smoothness of curved objects, and textures in the convolution layer. This helps filter out unwanted information to improve images. There are high-frequency filters where the intensity changes very quickly, for example, from a black to a white pixel and vice versa (Fig. 2.1).

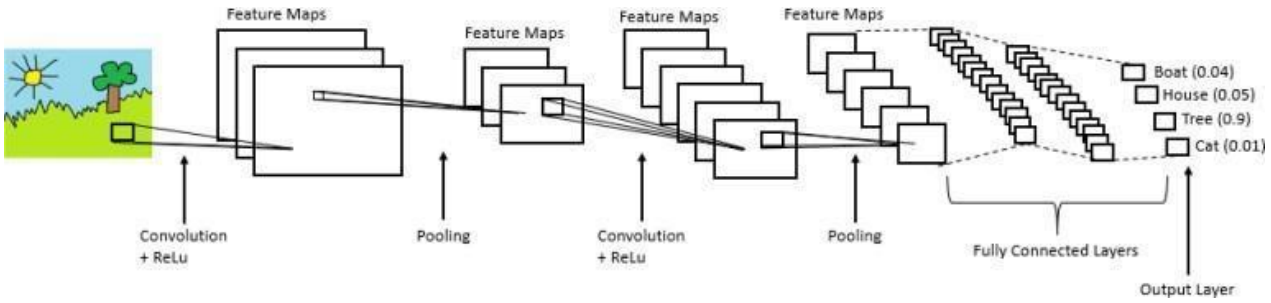

Fig. 2.1 How CNN works

In the diagram above we see:

The coagulation layer, where coagulation takes place. The union layer, where the union process takes place. Normalization, usually with ReLu. Fully connected layers.

ANNs are effectively used for image recognition, image classification, image object detection, face recognition, image segmentation, video analysis, creation of generative image models, and many other computer vision tasks. They can also be used in processing natural language, in particular for tasks related to text and sequence processing.

Therefore, CNN can be used for sequence processing. However, compared to LSTM (long short-term memory) and RNN (recurrent neural network), they have some limitations. The main advantage of LSTMs and RNNs is their ability to model long-term dependencies in sequences, especially where past data can influence future results.

Here are some important differences between CNN and LSTM/RNN:

Modeling temporal dependencies: LSTMs and RNNs are specifically designed to work with sequential data and can effectively model long-term dependencies over time. They store the state of the previous iteration and pass it to the next iteration, which allows them to retain information about previous data.

Local patterns and spatial features: ANNs are better suited to detect local patterns and spatial features in input data such as images. They use convolutional layers to collect information locally and combine layers to extract the most important features. However, they may miss some long-term dependencies in sequences that LSTMs and RNNs can track. Input size: ANNs work best with fixed input sizes, such as images, where the width and height can be specified in advance. LSTMs and RNNs can handle sequences of different lengths, making them more flexible for different types of sequential data.

2.1.3. Optimizers

Optimization algorithms contribute to increasing the efficiency of deep learning models, significantly speeding up learning and increasing its accuracy. During model training, it is important to achieve the minimum of the loss function and update the weights at each epoch. An optimizer is an algorithm or function that adjusts parameters such as weights and learning rate to reduce overall loss and improve model accuracy.

2.1.3.1. Gradient Descent

This optimization algorithm systematically changes the values of the weights and achieves a local one minimum using mathematical calculations. Formula describes the operation of the gradient descent algorithm.

Gradient Descent: This algorithm iteratively updates the parameters (weights) of the model in the direction opposite to the gradient of the loss function with respect to the parameters. It is expressed by the following formula (2.1):

$$\theta_{t+1} = \theta_t - \nabla J(\theta)_t \qquad (2.1)$$

Batch Gradient Descent: Computes the gradient of the loss function over the entire training data set.

Stochastic Gradient Descent (SGD): Computes the gradient of the loss function for

one training example at a time, making its computation faster but noisier.

Mini-Batch Gradient Descent: Computes the gradient of the loss function for a small subset (mini-batch) of the training data set, combining the benefits of both batch and stochastic gradient descent.

2.1.3.2. Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD): This variant of gradient descent updates the parameters using the gradient of the loss function relative to one training example at a time. This is expressed in the formula (2.2):

$$\theta_{t+1} = \theta_t - \nabla(\theta_t ; x_i ; y)_i \qquad (2.2)$$

The adaptive gradient descent algorithm differs from other gradient descent algorithms in that it uses different learning rates for each iteration. The change in learning speed depends on the difference in parameters during learning. This modification is very useful because real data contains both sparse and dense features, and it is unacceptable to have one value of the learning rate for all features.

Using the Adagrad optimizer has an important feature - it eliminates the need to manually adjust the learning rate. This method converges to a minimum speed and is faster and more reliable compared to gradient descent algorithms and other variations. However, the main disadvantage of the Adagrad optimizer is its ability to reduce the learning rate to very small values at a certain stage. This can negatively affect the accuracy of the model, because the model is not able to acquire new knowledge. This can happen due to the accumulation of squared gradients in the denominator, which causes the value of the denominator to increase.

A convolutional neural network is a powerful tool for analyzing spatially structured data. Its architecture includes several specialized types of layers, such as convolutional, pooling, and fully connected layers, each of which performs a specific function in the data processing process. Thanks to this, CNNs are able to automatically extract important features from the input data, which makes them extremely effective in pattern recognition tasks and other applications.

2.2. Mathematical models of convolutional neural network layers

2.2.1. Convolutional Layer

The convolution operation is defined as follows :

Input data: X∈RHWC , where H is height, W is width, C is the number of channels. Filter (convolution kernel): F∈RkH×kW×C, where kH and kW are the height and width of the filter. (2.3):

$$Y(i,j) = \sum m = 0kH - 1\sum n = 0kW - 1\sum c = 0C - 1X(i + m, j + n, c) \cdot F(m,n,c) + b \quad (2.3)$$

2.2.2. Activation Layer

The most common activation functions:

ReLU (Rectified Linear Unit) (2.4):

$$f(x) = max(0,x) \quad (2.4)$$

Sigmoid (2.5):

$$f(x) = 1 + e - x1 \quad (2.5)$$

Tanh (2.6):

$$f(x) = tanh(x) = ex + e - xex - e - x \quad (2.6)$$

2.2.3. Pooling layer

The most common methods of pooling:

Max Pooling (2.7):

$$Y(i,j) = max0 \le m < pH, 0 \le n < pWX(i \cdot pH + m, j \cdot pW + n) \quad (2.7)$$

pH and pW - height and width of the pooling window.

Average Pooling (2.8):

$$Y(i,j) = pH \cdot pW1\sum m = 0pH - 1\sum n = 0pW - 1X(i \cdot pH + m, j \cdot pW + n) \quad (2.8)$$

2.2.4. Normalization Layer

Normalization helps stabilize and accelerate learning.

Batch Normalization

For each input x from the input layer (2.9):

$$x\text{^} = (\sigma 2 + \epsilon)/\ sqrt(x - \mu) \quad (2.9)$$

where μ - average value, $\sigma 2$ - variance, ϵ - where is the average value, 2 is the variance, is a small positive number to avoid division by zero, , are the learning parameters, γ, β - are the learning parameters.

2.2.5. Fully Connected Layer

In a fully connected layer, each neuron is connected to all neurons of the previous layer. The output of the neuron is determined as follows (2.10):

$$y = f\left(\sum_{i=1}^{N} \omega i x i + b\right) \quad (2.10)$$

xi - input values, ωi - scales, b - displacement, f - function.

What if we simply fed the raw time series data into an LSTM-like neural network capable of multi-class classification? It would seem that the easiest solution to the problem! Imagine that we have data from two people walking at the same pace, but with unsynchronized steps. We can simulate this by taking one signal and shifting it slightly forward in time to create another signal.

Intuitively, if we look at both graphs, we realize that if the first represents walking, then the second probably does too. But a neural network does not have the same intuition. By comparing two graphs point by point, the neural network will find significant differences in each of them. Therefore, it will not be immediately obvious to the network that the signals represent the same activity.

2.2.6. Fourier transform

The Fourier transform is a mathematical operation used to decompose a signal into frequency components. It converts the signal from the time domain to the frequency domain, allowing it to analyze its spectral properties.

The basic idea of the Fourier transform is that any complex signal can be broken down into simpler signals, each of which has its own frequency. These simpler signals are called harmonics or frequency components. Each harmonic corresponds to a certain frequency and has its own amplitude and phase.

The Fourier transform can be applied to both discrete and continuous signals. For discrete signals we use Discrete Fourier Transform (DFT), and for continuous signals - Continuous Fourier Transform (CFT). In practical applications, the fast Fourier transform

(FFT) is often used, which is an efficient algorithm for calculating the DFT.

The process of Fourier transformation can be imagined as the decomposition of a signal into the sum of sinusoidal and cosine waves with different amplitudes and phases. This decomposition makes it possible to obtain information about the spectral components of the signal, that is, about which frequencies are present in the signal and with what strength (Fig. 2.2).

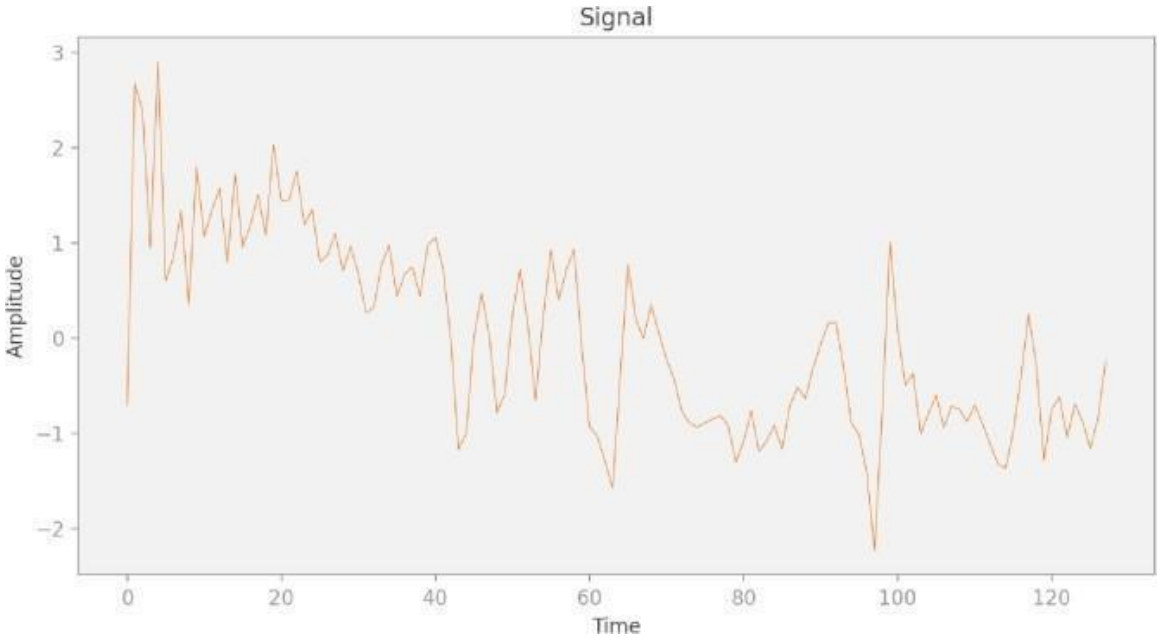

Fig 2.2

Since the shape of this input signal is much more complex than in my previous simple example, we need to add many more periodic signals of different frequencies to accurately represent it. However, we can still achieve this even if the signal is not periodic.

The Fourier transform is widely used in various fields, including signal processing, image processing, telecommunications, acoustic analysis, medical diagnostics, computer graphics, and many others. With the help of the Fourier transform, you can detect the characteristic frequencies of a signal, distinguish signals from noise, perform signal filtering, data compression, and much more.

2.2.7. Gabor transform (STFT)

One strategy for preserving both frequency and time information is to use the Gabor transform, also known as the short-time Fourier transform (TFT). The Gabor transform calculates the FFT on a moving time window and can be visualized using a plot of amplitude

as a function of frequency and time, which we call a spectrogram. When computing the Gabor transform, we perform the following operations for time from 0 to n in our scenario:

Center the filter function by time. We usually use a Gaussian function as a filter, as shown in the image below, but other filters can be used.

Multiply the input signal and the filter for all values of , resulting in a filtered signal of the same length as the input. This filtered signal preserves the information of the input signal in the time slot and ignores the rest of the information.

Finally, we take the FFT of the filtered signal, which shows the frequencies present in the time interval selected by the Gaussian filter.

I demonstrate this idea for the segments t=20 and t=100 (Fig. 2.3):

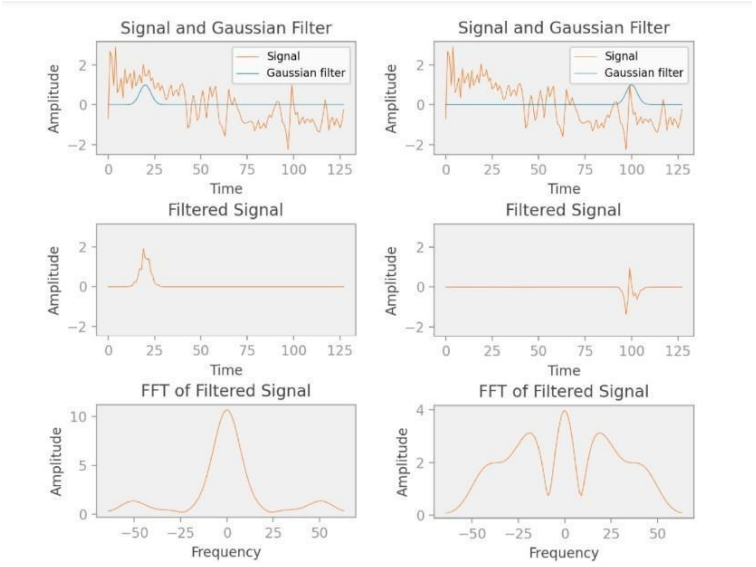

Fig 2.3 An example of using the filter function and FFT

You may have noticed that the FFT plots shown here contain both positive and negative frequencies and are even (symmetric) functions, whereas the FFT plot shown in the previous section had only positive frequencies. In fact, the FFT plot in the previous section looks similar to the one here, but I chose to show only the positive frequencies to focus on explaining the main concepts. More precisely, an FFT consists of complex numbers, and if the input signal is valid (as in our case), then the corresponding FFT is always "conjugate symmetric". This means that its real part is an even function (displayed along the abscissa axis), and the imaginary part is odd (displayed along the abscissa axis and then along the y-axis). Now, instead of considering the real and imaginary parts separately,

I'll take the magnitude of each complex number returned by the FFT, which also happens to be an even function. I use magnitude because we need real values for the next step where we visualize the Gabor transform as a spectrogram image.

Note that the spectrogram is vertically symmetrical. This is because each column contains the amplitude value of the FFT, which, as we saw earlier, is an even function. I highlighted the bars with frequencies of 20 and 100, which correspond to the calculations we saw in the previous graphs. If you look closely, you'll notice that frequencies with higher amplitudes in the FFT plots correspond to darker pixels in the spectrogram.

Spectrograms are a great way to visualize the frequencies present in a signal over time. However, they have a significant drawback: once you choose a value for the standard deviation in a Gaussian filter, the filter will be most effective at detecting a particular frequency and less effective at much lower frequencies and localizing much higher frequencies.

Imagine that one of the frequencies that make up your input signal is a very wide wave (representing a very low frequency). If the Gaussian filter is much narrower than this wave, it will not be able to filter out and detect this low frequency in any of the time windows. One way to mitigate this problem is to use a wide Gaussian filter. However, with a wide filter, we store less temporal information because our sliding window is much larger. So it's a trade-off: if we're concerned about preserving low frequencies, we choose a wide filter; if we are concerned about preserving temporal information, we choose a narrow filter.

2.3. Convolutional neural network parameters

The optimal convolutional neural network (CNN) architecture is determined by numerous parameters that affect its performance and learning ability. A detailed understanding of these parameters is critical to effectively designing and debugging deep learning models.

One of the key parameters is the number of filters in the convolutional layer. Each filter is responsible for detecting a certain feature of the visual image. A larger number of filters allows the model to detect more diverse features, but at the same time increases the computational complexity of the network.

Filter size is also an important parameter. It defines the range of input data that each filter analyzes. Commonly used filters are 3x3, 5x5 or 7x7. Smaller filters usually respond better to small details, while larger filters can find more general structures.

The convolution step determines the distance between applying the filter to the input data. A large step reduces the size of the original image, which can lead to information loss, but reduces the computational cost.

Padding is the process of adding extra pixels around the edges of an input image before applying filters. This allows you to preserve the dimensionality of the image after convolution and facilitates the analysis process.

The activation function is used to introduce nonlinearity into the model, allowing it to solve more complex problems. The most common activation function is ReLU, which learns quickly and effectively tackles the vanishing gradient problem. In addition, it is important to consider aspects such as regularization, hyperparameter optimization, and retraining constraints to ensure the robustness and performance of the model under different conditions. These parameters allow the creation of networks that efficiently use input data to solve a variety of tasks in the field of computer vision, including classification, object detection, and face recognition.

2.3.2. Feature Map

A feature map is the result of applying convolutional layer filters to the input image. Feature map options include character map size:

Depends on input image size, filter size, pitch and padding.

The formula for calculating the size of the original feature map (2.11):

$$Output\ Size = \frac{Input\ Size - Kernel\ Size + 2 \cdot Padding}{Stride} + 1 \qquad (2.11)$$

For example, for an input image of size 32x32 pixels, kernel 3x3, step 1, and padding "same", the output feature map will be 32x32.

2.3.3. An example of parameter calculation

Let's imagine that we have an input image of size 32x32 pixels with 3 channels (red, green, blue). Let's also assume that in the first convolutional layer we use 32 filters of size 3x3 with step 1 and padding "same", which means that the output feature map size remains

the same as the input.

After applying this convolution layer, we will get 32 feature maps of size 32x32 each. These feature maps are intermediate processing results that represent detected features at different levels of abstraction of the input image. For example, each feature map may correspond to the detection of a specific type of edges, textures, or shapes in the source image.

It is important to note that the parameters of a convolutional neural network, such as the number of filters, filter size, pitch, padding, and activation function, are of great importance to the performance and feature detection ability of the network. The setting of these parameters determines how effectively the network can separate useful information from the input data and what features can be detected at different stages of processing.

2.4. Convolutional neural network learning methods

Training a convolutional neural network (CNN) involves adjusting the weights and biases in each layer so that the network can correctly predict the output values based on the input data. This is achieved using optimization algorithms and various regularization techniques that help avoid overtraining. Below are the main methods of CNN training.

Forward Propagation:

The forward propagation process in Convolutional Neural Networks (CNN) involves passing the input data through different layers of the network to produce the output. At each stage of this process, various operations such as convolution, activation, and others are applied to identify and highlight important features in the input data. After passing through all the layers of the network, we get an output that is compared with real labels to calculate the error.

Backpropagation:

The backpropagation process is used to adjust the network parameters, such as weights and biases, based on the calculated derivatives of the loss function with respect to each parameter. This process includes the following steps:

Calculation of gradients: The gradients of the loss function with respect to each weight and offset are calculated using the error backpropagation algorithm.

Parameter update: Network weights and offsets are updated using computed gradients and an optimization method such as stochastic gradient descent (SGD) or other optimizers.

Together, these two processes help ensure efficient training of convolutional neural networks, giving them the ability to detect and classify objects in images with high accuracy.

2.4.1. Loss Functions

The loss function evaluates how well the network predicts the output values. The most common cost functions include:

Cross-entropy loss (Cross-Entropy Loss):

It is used for classification tasks (2.12).

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \quad (2.12)$$

where yi = real marks, y^i - predicted probabilities.

Mean Squared Error, MSE:

It is used for regression problems (2.13).

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (2.13)$$

where yi = real marks, y^i - predicted probabilities.

2.4.2. Optimization Algorithms

Optimization algorithms are used to update weights and shifts based on gradients. The most common algorithms:

Adam (Adam):

An adaptive optimization method that combines the advantages of two other methods: AdaGrad and RMSProp.

Stochastic Gradient Descent, SGD:

Updates the weights based on the gradient of the loss function with respect to each parameter (2.14).

$$w = w - \eta \cdot \nabla L(w) \quad (2.14)$$

where η — learning rate, $\nabla L(w)$ — the gradient of the loss function.

2.4.3. Regularization

Regularization helps prevent overtraining by penalizing large weights.

L2-regularization (L2 Regularization):

Adds a penalty to the loss function proportional to the square of the weights (2.16).

$$Loss = Original\ Loss + \lambda \sum w2 \quad (2.16)$$

where λ — regularization factor.

2.5. Preliminary processing

2.5.1. The value of data preprocessing

Pre-processing is an important stage in the analysis of hyperspectral images, as it allows to improve the quality of the data and prepare them for further analysis. This stage includes noise removal, normalization, calibration, and other techniques that provide more accurate and reliable results.

2.5.2. Noise removal

Hyperspectral images can be subject to various types of noise, which can negatively affect the accuracy of analysis and data processing. Noise can come from a variety of sources, such as electronic interference, atmospheric effects, or technical artifacts. To improve the quality and reliability of the analysis, it is necessary to remove this noise. Let's consider some methods of noise removal in hyperspectral images:

Spectral noise: This type of noise occurs due to random fluctuations in the spectral data, which can be caused, for example, by low-quality sensors or the influence of the surrounding environment. Smoothing techniques such as a median filter or a Gaussian filter can be used to remove spectral noise. These methods make it possible to smooth out high-frequency fluctuations and reduce the effect of noise on the data.

Spatial noise: This type of noise is caused by uneven lighting or electromagnetic interference that may occur on the surface of the object or as a result of equipment operation. Spatial filtering methods, such as a box filter or adaptive filters, are used to remove spatial noise. These techniques allow smoothing out spatial irregularities and removing noise-causing artifacts.

Correlation noise: This type of noise occurs due to correlation between image pixels

or spectral channels, which can be caused by, for example, lighting or atmospheric conditions. Principal component analysis (PCA) techniques can be used to remove correlation noise. These techniques allow the identification and removal of correlated image components that may be caused by noise.

Denoising is an important step in hyperspectral image processing that helps ensure the accuracy and reliability of data analysis. The choice of noise removal method depends on the specific features of the image and the nature of the noise present on it.

2.5.3. Data normalization

Normalization is a key step before processing hyperspectral data, aimed at reducing the influence of changes in lighting and other external factors that can affect the results of the analysis. It includes the following methods:

Min-Max Normalization: This method scales the pixel values so that they are within a specified range, usually 0 to 1. This process results in all pixel values being within a standard range, making it easier to compare between them.

Z-Normalization: This method centers the data around the mean and scales it so that the standard deviation is one. It makes the data more stable and helps reduce the impact of abnormal values.

Spectral normalization: This method is used to normalize spectral data in order to reduce the influence of illumination fluctuations on the analysis results. It allows you to make the spectral data more consistent and resistant to changes in lighting.

These data normalization methods in hyperspectral analysis help ensure stable and reliable data processing results, reducing the influence of external factors on the analysis results.

2.5.4. Data calibration

Calibration of hyperspectral data is a necessary step before their further use in scientific research or applications in practical tasks. This process includes the transformation of measured spectral values into physical quantities, which ensures their consistency and accuracy. Two main approaches to data calibration include:

Radiometric calibration: This process provides the conversion of measured pixel

values into physical quantities that reflect the physical properties of the objects in the image. The most common metrics used for radiometric calibration are reflectance and radiance. This process allows for a standardized approach to measuring and comparing hyperspectral data.

Geometric calibration: Includes the correction of geometric distortions that may occur during the shooting process. This includes image alignment, correcting geometric distortions, and ensuring that pixels are accurately mapped to physical coordinates. Geometric calibration provides correspondence between real objects and their reflections on hyperspectral images, which is important for the accurate determination of geographic objects and the implementation of further analyses.

2.5.5. Selection and allocation of features

Selection and extraction of features is an important stage of hyperspectral data processing, as it allows to reduce the dimensionality of the data and isolate the most important characteristics for analysis. This process includes a variety of techniques that can be used to identify valuable traits. Below we will consider some of them:

Principal Component Analysis (PCA): This method of statistical data analysis allows you to reduce the dimensionality of data while retaining as much information as possible. It transforms the correlated variables into an orthogonal set of principal components (or eigenvectors), each of which describes part of the variance of the data. Thus, PCA allows you to select the most informative aspects of the data and use them for further analysis.

Independent Component Analysis (ICA): This method is also used to reduce the dimensionality of the data, but it aims to identify independent signal sources. This allows you to split a complex signal into separate, independent components, each of which can represent certain characteristics or properties of the data.

Feature extraction methods based on wavelet transform: They are used to analyze data at different scales and separate signal from noise. The wavelet transform decomposes the signal into components with different frequencies, allowing you to highlight important features at each scale level. This can be useful for identifying certain patterns or structures in the data that may be important for classification or analysis.

2.5.6. Spectral-spatial correction

Spectral-spatial correction is an important step in processing hyperspectral data, as it allows combining information about the spectral properties of objects with their spatial characteristics. This improves data quality and enables more accurate classification and analysis. The two main approaches to spectral-spatial correction include:

Convolutional filters: This approach involves applying convolutional filters to hyperspectral data. Convolutional filters are used to extract spatial-spectral features by processing the data in the form of a "window" that passes over the entire image. This allows you to identify features that may be important for further analysis, such as textures, object boundaries, and other characteristics that may affect classification.

Data fusion techniques: This approach consists in integrating spectral and spatial information to create more informative images. The use of data fusion methods allows you to combine information about the spectral characteristics of objects with their spatial attributes. For example, filtering or segmentation techniques can be applied to highlight regions of interest in an image and analyze their spectral properties.

These spectral-spatial correction approaches can be used to optimize the processing of hyperspectral data and improve their use in various applications, such as land cover class determination, environmental monitoring, and other hyperspectral studies.

2.6. Basic processing

2.6.1. Importance of basic processing of hyperspectral data

Basic processing of hyperspectral data includes the application of algorithms and methods for analysis, classification and regression of hyperspectral images. This step is key to obtaining informative results that can be used in various fields such as ecology, medicine and military applications.

2.6.2. Methods of classification of hyperspectral data

Classification of hyperspectral data is a critical task that involves dividing image pixels into classes based on their spectral signatures. This process includes several approaches that can be divided into supervised, unsupervised and semi-supervised methods.

Supervised methods use labeled data to train models, which are then used to classify

new pixels.

The method of nearest neighbors based on their spectral similarity to k nearest neighbors in spectral space. It is simple and efficient for small datasets, but its performance degrades on large datasets due to high computational complexity.

Support vector method uses hyperplanes to separate classes in a multidimensional spectral space. This method works well on high-dimensional data and is effective for complex classification problems.

Neural networks particularly multilayer perceptrons (MLPs) and convolutional neural networks (CNNs), have been used effectively to classify hyperspectral data. MLPs can learn complex non-linear dependencies, and CNNs can extract spatio-spectral features, improving classification accuracy.

Decision trees and ensemble methods such as Random Forest and Gradient Boosting use multiple trees to improve classification accuracy. They are resistant to overtraining and can process large data sets with high accuracy.

Unsupervised methods do not require labeled data for training and are used to detect natural clusters in the data.

K-means (k-means): the method groups pixels into clusters based on their spectral similarity. It is effective for detecting major classes in hyperspectral images, but requires a prior determination of the number of clusters.

Hierarchical clustering builds a hierarchical structure of clusters that allows you to identify similar groups of pixels at different levels of detail. It is useful for analyzing complex data with many classes.

Self-learning maps are neural networks used to visualize and cluster multidimensional spectral data. They can reduce the dimensionality of data and reveal hidden structures.

Semi-supervised methods combine labeled and unlabeled data to train models, which allows for improved classification accuracy under conditions of limited labeled data.

Graph methods use graphs to model relationships between labeled and unlabeled data. They can effectively use the data structure to improve classification.

Semi-supervised neural networks approach combines the advantages of neural

networks and semi-supervised learning, allowing models to be trained on both labeled and unlabeled data, which increases their performance.

2.6.3. Methods of regression of hyperspectral data

Regression of hyperspectral data aims to predict continuous values based on the spectral signatures of pixels. This is an important aspect of analysis that finds application in various fields such as agriculture, environmental monitoring, geology and others. The main regression techniques used for hyperspectral data include:

Linear regression is a simple and basic method that assumes linear relationships between spectral features and target values. In this method, each spectral feature receives a weight factor, and the sum of these weighted features is used to predict the target value. Linear regression is efficient when working with large amounts of data, but can be limited in cases where the relationship between variables is non-linear.

Multilayer linear regression is an extension of linear regression that includes additional layers of features to account for non-linear relationships. This approach allows the model to take into account more complex dependencies between spectral features and target values, which improves the accuracy of predictions in cases where the relationship between variables is nonlinear.

Regression based on support vector methods can be adapted for regression problems, called support vector regression (SVR). SVR uses hyperplanes to predict target values in multidimensional spectral space. This method works well with high-dimensional data and is effective for complex regression problems. SVR allows you to model non-linear relationships using kernel functions, which expands its capabilities compared to linear regression.

Neural networks for regression can be used to model complex nonlinear relationships in hyperspectral data. The use of multilayer neural networks allows the detection of hierarchical features, which significantly improves the accuracy of predictions.

Neural networks are particularly useful in cases where the data has high dimensionality and complex structure. Their flexibility and ability to learn non-linear dependencies makes them a powerful tool for regression of hyperspectral data.

Ensemble methods such as Gradient Boosting and Random Forest use a combination of multiple models to improve regression accuracy. Gradient Boosting creates a sequence of models where each subsequent model corrects the errors of the previous one, allowing for incremental improvements in prediction accuracy. Random Forest uses multiple decision trees to generate an average prediction, which reduces the risk of overtraining and increases the stability of the model. Ensemble methods are effective for processing large and complex hyperspectral data, providing high accuracy and reliability of predictions.

2.6.4. Integration of spectral and spatial information

The integration of spectral and spatial information is essential to improve the accuracy of classification and regression in hyperspectral data. This process allows you to combine details from both types of data, providing a more complete and informative picture for analysis. The main approaches to the integration of spectral and spatial information include:

Convolutional Neural Network are a powerful tool for image analysis, including hyperspectral images. CNNs are able to automatically detect and learn spatio-spectral features using layer-by-layer convolutional operations. This allows the model to detect complex patterns in the data, improving classification and regression accuracy. Due to their architecture, CNNs can effectively combine information from different spectral channels and spatial coordinates, which makes them indispensable for analyzing hyperspectral images.

Methods of spatial-spectral analysis

Spatial-spectral analysis involves the use of combined methods to analyze both spectral and spatial relationships. Examples of such methods include:

Spatial-spectral filters: Use of filters for simultaneous selection of spatial and spectral features.

Methods based on local windows: Analysis of data in small local windows, which allows to take into account the interaction between neighboring pixels.

Combined classification models: Using methods that combine spectral features with textural or spatial features for more accurate analysis.

Hybrid models combine different methods to improve the results of hyperspectral data

analysis. Example:

CNN with traditional classification methods: Combination of convolutional neural networks with methods such as support vector method (SVM) or decision trees to improve the accuracy and stability of the results.

Ensemble methods: Using ensembles of models that combine the results of several different algorithms, such as Random Forest or Gradient Boosting, to achieve more reliable and accurate predictions.

2.6.5. Assessment and validation of results

Evaluation and validation of the results are critical stages that allow determining the effectiveness and reliability of hyperspectral data processing methods. The main approaches to assessment and validation include:

The confusion matrix is used to assess classification accuracy, including such metrics as accuracy, recall, specificity, and F1 measure. It shows how well the model classifies each class and allows you to identify classes where the model has difficulty.

ROC curves and AUC

ROC curves (Receiver Operating Characteristic curves) and area under the curve (AUC) are used to evaluate the performance of classifiers. The ROC curve shows the relationship between sensitivity and specificity at different decision thresholds. AUC measures the overall ability of a classifier to distinguish between positive and negative classes. A high AUC value indicates good classifier performance.

Cross-validation

Cross-validation is a method of assessing the stability and generalizability of models. The most common approach is k-fold cross-validation, where the data is divided into k subsets and the model is trained and evaluated k times, each time using a different subset as the test set. This allows for more reliable estimates of model performance and reduces the risk of overtraining.

The basic processing of hyperspectral data, including the integration of spectral and spatial information, as well as the evaluation and validation of the results, are key steps that ensure the effective use of the data for a variety of applications in science and industry. The

use of modern methods of analysis, such as convolutional neural networks and hybrid models, allows to achieve high accuracy and reliability of predictions, which is critical for the successful solution of many applied problems.

2.7. Generalization of the obtained results for different frequencies

Generalization of the results is a key stage in the analysis of hyperspectral images, as it allows to evaluate the effectiveness of the applied methods for different spectral frequencies and to determine the most informative frequencies for further analysis.

2.7.1. Analysis of results for different spectral ranges

Each spectral range can have different informativeness depending on the specific task. Analysis of the results includes several key aspects.

First, the classification accuracy assessment allows you to determine the classification accuracy for different spectral ranges and identify the ranges with the highest accuracy. This includes using various metrics such as overall accuracy, class accuracy, precision, completeness and F1-measure for detailed analysis.

Second, frequency significance analysis helps identify the frequencies that most affect the classification or regression results. This allows you to identify the key spectral regions that are most informative for a specific task. For this, methods such as feature importance analysis are used to assess the impact of each spectral frequency on the result.

The third aspect involves comparing different processing methods. This allows comparison of the performance of different methods for each spectral range, in particular comparing traditional methods with modern approaches such as convolutional neural networks (CNN), support vector methods (SVM), decision trees, etc. The effectiveness of combined approaches that combine the advantages of different methods to achieve better results is also investigated.

2.7.2. Generalization of results for spectral-spatial methods

The combination of spectral and spatial information allows to improve the results of the analysis. The evaluation of the effectiveness of spectral-spatial methods shows how the integration of spectral and spatial information increases the accuracy and reliability of the results. This includes choosing the optimal methods that are best suited to integrate these

data, as well as analyzing the effect of individual frequencies on the results of such analysis.

2.7.3. Development of recommendations for practical application

Based on the generalized results, recommendations for the practical application of hyperspectral image analysis methods are developed. This includes the selection of informative frequencies for specific tasks, such as agriculture, ecology or medicine, as well as recommendations for the use of the most effective processing and analysis methods for different spectral bands. It is also important to provide practical examples of applications where the generalized results have led to significant improvements.

2.7.4. Assessment of stability and generalizability of results

An important component of the generalization of the results is the assessment of their stability and generalizability. The use of cross-validation methods helps to assess the stability of results on different data samples. In addition, testing on different sets of hyperspectral data allows you to check the generalizability of the obtained results. Deviation analysis helps identify possible causes of deviations and improve analysis methods to achieve more accurate and reliable results.

Thus, summarizing the results of hyperspectral data analysis is a complex and multifaceted process that ensures optimization of processing methods and allows effective use of spectral and spatial information to solve various scientific and industrial tasks.

2.8. Conclusions

Based on the generalization of the obtained results, several important conclusions can be drawn about the effectiveness of the applied methods and their suitability for various tasks.

First, an important finding is the determination of the most informative spectral ranges for different applications. Some frequencies have been found to be more informative and better suited for specific tasks, such as agriculture, environmental monitoring or medical research. This allows you to optimize the process of processing and analyzing hyperspectral data, focusing efforts on the most important bands.

Secondly, the analysis of the results made it possible to develop recommendations on the most effective methods of processing and analyzing hyperspectral images. A

combination of traditional methods such as Principal Component Analysis (PCA) and modern approaches such as Convolutional Neural Networks (CNN) has been found to provide significant improvements in accuracy and reliability of results. Such recommendations are valuable for practical applications in various fields where the use of hyperspectral data is crucial.

In addition, the generalization of the obtained results helped to determine directions for further research. It was found that there is a need to improve the methods of integration of spectral and spatial information, as well as to develop new algorithms for more accurate classification and regression. Further research may focus on improving the processing and analysis methods, which will increase the efficiency of using hyperspectral images in practice.

Thus, the generalization of the obtained results for different frequencies allows to determine the most informative ranges and effective processing methods, which is critically important for the successful application of hyperspectral images in various fields.

CHAPTER 3
IMPLEMENTATION OF EDUCATION ACCORDING TO THE PROGRAM. CNN METHOD

3.1. Statement of the problem of structural parametric synthesis

Structural parametric synthesis (SPS) is an important method in the field of machine learning and artificial intelligence, which allows you to automatically create models with optimal parameters. This process aims to find the best model structure and parameters to achieve maximum performance on a given task.

3.1.1. The aim of the study

The purpose of this study is to develop and implement effective semi-supervised learning methods for hyperspectral regression, which will improve forecasting accuracy due to the optimal use of available data and model parameters. In particular, we focus on the use of CNN methods and hybrid models such as HybridSN to solve hyperspectral regression problems.

In the context of hyperspectral regression, the main goal is to provide accurate reproduction of spectral information with a limited amount of training data. The use of semi-supervised methods allows you to effectively combine available labels with a large amount of unlabeled data, which significantly increases the performance of models.

In addition, we set ourselves the following specific goals:

Development of new algorithms for automatic adjustment of model parameters.

Study of the influence of various parameters on the accuracy of models.

Optimizing hyperparameters to improve performance.

3.1.2. Objectives of the study

To achieve the above goals, it is necessary to perform the following tasks:

Conduct an analysis of existing methods and approaches to semi-supervised learning, in particular in the context of hyperspectral regression.

Develop and implement new algorithms for structural parametric synthesis of models.

Conduct experimental studies to evaluate the effectiveness of the developed algorithms using different data sets.

Determine optimal model parameters and hyperparameters to achieve maximum performance.

Compare the results of new methods with existing approaches and analyze the obtained data.

The results of this study can be used to improve forecasting methods in various fields, such as remote sensing, medical diagnostics, environmental monitoring, etc. Thus, the implementation of these tasks will contribute to the development of modern methods of machine learning and their application in real conditions.

3.2. Overview of methods

This section reviews the main techniques used in the context of semi-supervised learning for hyperspectral regression. Special attention is paid to key parameters, loss functions, regularization and normalization methods, which are important aspects for achieving high accuracy of models.

3.2.1. Overview of the main parameters

The main parameters of machine learning models include several categories, each of which affects the performance and accuracy of the models. Among these parameters, the hyperparameters that determine the structure of the model and the learning process are particularly important.

Number of layers and neurons: One of the main parameters in neural networks is the number of layers and the number of neurons in each layer. These parameters determine the complexity of the model and its ability to generalize. A model that is too complex may overtrain on the training data, while a model that is too simple may fail to capture complex dependencies in the data.

Training rate: The training rate determines how quickly the model updates its weights during training. A high learning rate can lead to unstable learning and unassembled models, while too low a rate can make learning too slow and inefficient.

Batch size: The batch size determines the number of examples used for one weight update step. Smaller batches may result in noisy gradient estimates, while larger batches may provide a more stable update, but with higher memory requirements.

Activation functions: Activation functions determine how the output of each neuron is calculated based on its input. Common activation functions are ReLU, Sigmoid, and Tanh. Each of them has its advantages and disadvantages depending on the task.

Regularization hyperparameters: Regularization is used to prevent overtraining of models. Regularization hyperparameters include L1 and L2 regularization coefficients, as well as dropout probability.

3.2.2. Loss of function

The loss function is a key component in the process of training machine learning models, as it determines how well the model fits the given data. In the context of hyperspectral regression and semi-supervised learning, the loss function can take different forms, depending on the type of task and the learning method.

Mean Squared Error (MSE): This is one of the most common loss functions for regression problems. MSE calculates the root mean square deviation between predicted and actual values. It is sensitive to large errors, which makes it useful for problems where accurate predictions are important.

Cross-Entropy Loss: This loss function is typically used for classification problems, but can be adapted for multi-class regression problems. It calculates the difference between the predicted class probabilities and the actual classes.

Huber Loss: This loss function is a combination of MSE and Mean Absolute Error (MAE). It is less sensitive to large errors compared to MSE, making it useful for problems with outliers or noise in the data.

Custom Loss Functions: In some cases, custom loss functions that are tailored to the specific needs of the problem may be used. These can be functions that take into account the peculiarities of hyperspectral data or the requirements of a specific application.

3.2.3. Regularization and normalization

Regularization and normalization are critical techniques to improve the generalization ability of models and prevent overtraining. They help reduce the complexity of the model and increase its ability to work with new data.

L2 Regularization (Ridge Regression): This method adds an additional term to the

loss function, which is the sum of the squares of the weights. This helps to reduce the complexity of the model and prevents the weights from being too large.

L1 Regularization (Lasso Regression): L1 regularization adds the sum of the absolute values of the weights to the loss function. This can lead to the nullification of some weighting coefficients, which contributes to the creation of more understandable models.

Dropout: Dropout is a regularization method that randomly drops some neurons during training. This helps prevent overtraining and increase the generalization ability of the model.

Batch Normalization: Batch Normalization normalizes the output values of neural network layers, which helps to stabilize and speed up the learning process. It can also perform a regularization function, reducing the need for other methods.

Data Augmentation: Data augmentation is a method that increases the size of the training data set by creating new examples from existing data. This helps to reduce overtraining and improve the generalization ability of the model.

Thus, the use of regularization and normalization is an important aspect in building machine learning models that helps to achieve better results in practice.

3.2.4. Learning from scratch and transfer learning

Training from Scratch involves initializing the model with random weights and incremental training based on a specific data set. This approach is more labor intensive and requires a large amount of labeled data to achieve high performance. The main advantage of learning from scratch is the ability to adapt the model to the specific characteristics of the data it works with.

Transfer Learning uses models that have been previously trained on large, general data sets to further train them on specific tasks. This approach significantly reduces training time and increases efficiency, especially when the available data set is limited. Transfer learning allows you to use the already acquired experience of the model for accelerated learning on new tasks, which ensures high accuracy and efficiency even under conditions of limited resources.

3.2.5. Data augmentation

Data augmentation is an important technique that allows you to increase the size and diversity of a training dataset by applying various transformations to existing data. This helps the model better generalize information and increases its robustness to variations in the data. Basic augmentation methods include geometric transformations (rotate, scale, shift), color transformations (changes in brightness, contrast, saturation) and adding noise. These techniques help avoid overtraining and improve the model's ability to work with new, unfamiliar data.

3.2.6. Optimization of hyperparameters

Optimizing hyperparameters is a critical step in training machine learning models. Hyperparameters define the model architecture and training process, including parameters such as training rate, batch size, number of layers, and neurons. Hyperparameter optimization methods include Grid Search, Random Search, and Bayesian Optimization. Grid Search involves an exhaustive search on a given grid of hyperparameter values, Random Search uses a random search in the hyperparameter space, and Bayesian Optimization uses probabilistic models to determine optimal values. Proper selection and tuning of hyperparameters can significantly improve model performance and efficiency.

3.2.7. Using pre-trained models

The use of pre-trained models is an effective approach in machine learning that allows the knowledge gained from training on large data sets to be used for new problems. This is especially useful when the new data set is small or when training time is limited. Pre-trained models such as ResNet, VGG, or BERT demonstrate high accuracy and can be quickly adapted to new tasks by retraining the last layers or fine-tuning the entire model. This allows you to significantly reduce the time and resources needed for training and achieve high results even on difficult tasks.

3.2.8. Dataset Shift

Dataset Shift refers to a situation where the distribution of the training data differs from the distribution of the test data or the data on which the model will be used in real-world conditions. This can happen for a variety of reasons, such as changes in the data

collection environment, technological changes, seasonal changes, or other factors. The bias of the dataset can significantly affect the performance of the model because it can learn patterns that do not correspond to real conditions.

There are several types of dataset displacement:

Covariate Shift: When the distributions of the inputs change, but the dependence between the inputs and the labels remains the same.

Prior Probability Shift: When the label distributions change but the input distributions remain the same.

Concept Shift: When the very dependency between input data and labels changes.

Various methods are used to detect and correct dataset bias, such as retraining the model on new data, domain adaptation, using knowledge transfer techniques, and other approaches. An important aspect is the monitoring of model performance on new data and timely adaptation to changes in data distributions.

In the following example, we consider covariance bias in hyperspectral soil moisture regression, which is the most relevant type of dataset bias for this application. The distributions of the hyperspectral and reference data are biased between the training dataset and the unknown dataset. The distribution of reference soil moisture data is shown (Fig. 3.1). After training on the training dataset, the machine learning model should be able to estimate soil moisture on a new, unknown dataset. However, due to covariance bias, the machine learning model is not able to adequately estimate soil moisture on a given unknown dataset.

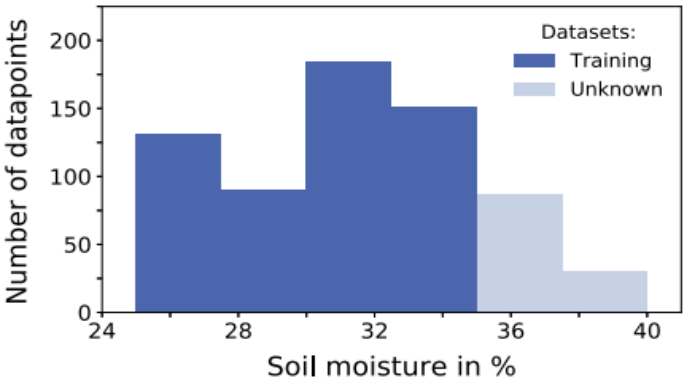

Fig. 3.1

3.2.9. Dataset Splitting

Dataset Splitting is a key step in the process of training a machine learning model. The correct distribution of data into training, validation and test sets provides an objective assessment of model performance and prevents retraining of features. (Fig. 3.2).

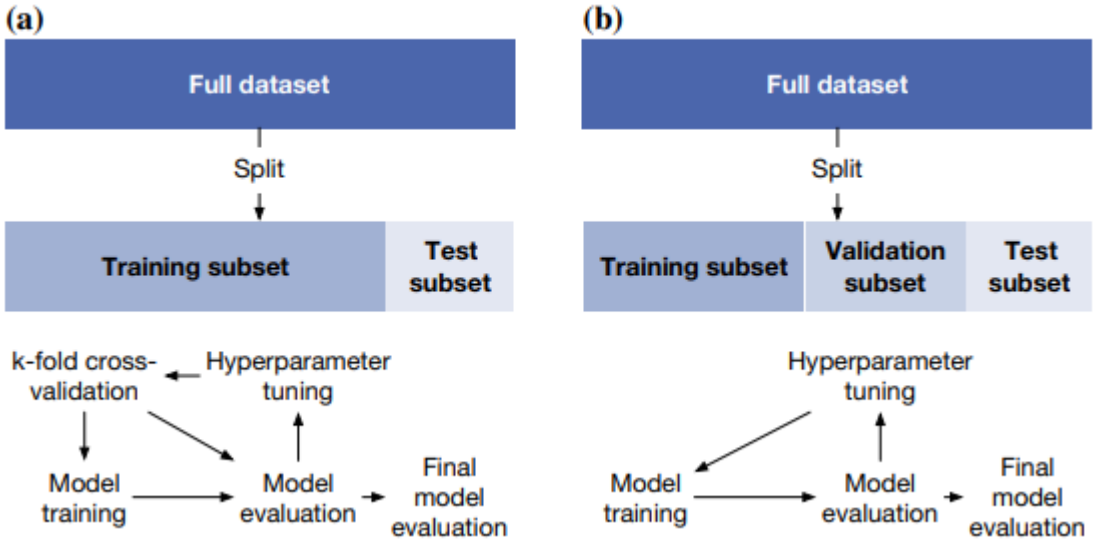

Fig. 3.2

There are different strategies for distributing the dataset (Fig. 3.3):

Standard distribution: the data is divided into three parts - the training set (usually 70-80% of the total data), the validation set (10-15%) and the test set (10-15%). The training set is used for model training, the validation set is used for hyperparameter tuning and model selection, and the test set is used for final performance evaluation.

K-Fold Cross-Validation: The data is divided into K subsets and the model is trained K times, each time using one of the subsets as test and the other K-1 subsets as training. This makes it possible to obtain a more stable assessment of model performance due to the use of all data as test data.

Stratified Splitting: the data is divided in such a way that the proportions of classes in the training, validation and test sets are the same. This is especially important for unbalanced data, where one class may be significantly predominant.

Time-Series Splitting: for time series, the data is split in such a way that the test set contains more recent data than the training set. This reflects real-world conditions where future data is predicted based on past data.

Choosing the right dataset distribution strategy is critical to ensure reliable and objective model evaluation, which allows for the development of more accurate and efficient machine learning models.

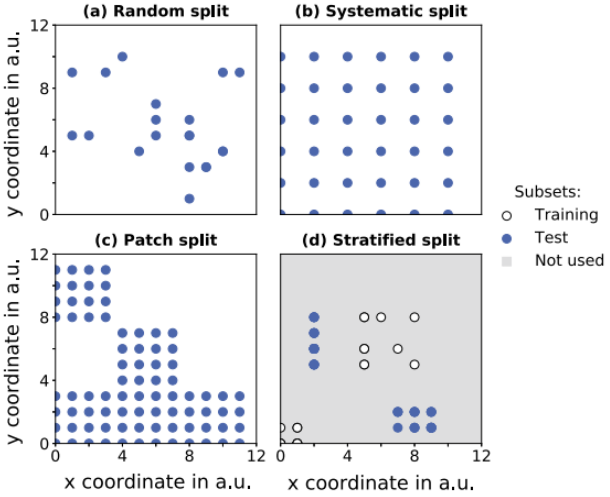

Fig. 3.3

3.2.10. Evaluation of the Metrics of the Model

Evaluating model metrics is an important part of the process of developing and improving machine learning models. Evaluation metrics allow you to quantify model performance and compare different models with each other. Key evaluation metrics include mean square error (MSE), mean absolute error (MAE), coefficient of determination (R^2), and others.

Mean Squared Error (MSE)

MSE is one of the most common evaluation metrics for regression models. It measures the mean squared error between predicted and actual values. MSE is determined by the formula (3.1):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (3.1)$$

where y_i - actual value, $\hat{y}$ - predicted value, n - number of observations. The smaller the MSE value, the better the performance of the model.

Mean Absolute Error (MAE)

MAE measures the mean absolute difference between predicted and actual values. It

is determined by the formula (3.2):

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (3.2)$$

MAE is a more interpretable metric because it is expressed in the same units as the predicted values.

Coefficient of determination (R^2)

R^2 shows how much of the variation in the dependent variable is explained by the independent variables of the model. It is defined as (3.3):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (3.3)$$

where $\underline{y}$ - the average of the actual values. Value R^2 varies from 0 to 1, where 1 means the model perfectly explains the variation in the data.

Fig. 3.4 shows three examples of the distribution of a machine learning model that fits simulated data with a single input feature x and a target variable y.

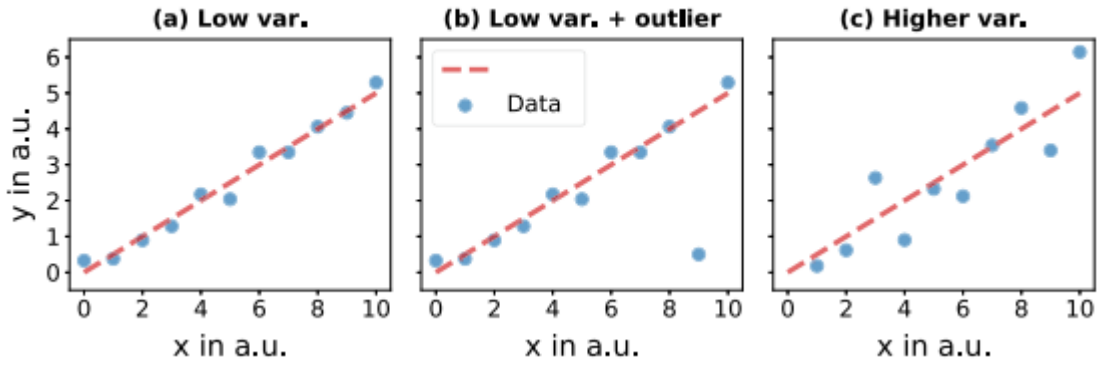

Fig. 3.4

These results indicate that machine learning regression models should be evaluated based on several metrics. Notebook 5.2 (https://github.com/felixriese/hyperspectral-regression) shows code implementations for three example distributions.

3.3. Introduction of own development approach

3.3.1. Motivation and main goals

The motivation for developing a new approach to hyperspectral regression is due to the need to overcome existing limitations in modeling accuracy and efficiency. Hyperspectral images provide detailed information about the spectral properties of

materials, allowing complex classification and regression tasks to be performed. However, due to the large number of spectral bands and the complexity of the models used, the training process becomes very expensive in terms of time and computational resources.

The main objectives of this research include improving the accuracy of simulations by developing a new hybrid model that combines the advantages of different machine learning methods to achieve higher accuracy in the regression of hyperspectral data. Reduced computational cost by creating an efficient model architecture that reduces training time and resources needed to process large hyperspectral images. As well as the adaptability and versatility of the method, which is easily adapted to different types of hyperspectral data and tasks, providing high performance in a wide range of applications.

3.3.2. An overview of the hybrid model HybridSN

HybridSN (Hybrid Spectral Network) hybrid model was created to combine spatial and spectral properties of hyperspectral images. This makes it possible to achieve high accuracy of classification and regression, using both spatial and spectral features of the data.

The main components of HybridSN include a 3D Convolutional Neural Network (3D-CNN), which is used to extract spatio-spectral features from hyperspectral images. 3D-CNN allows you to take into account both spatial and spectral information at the same time. 2D Convolutional Neural Network (2D-CNN) is applied after 3D-CNN pre-processing to further extract spatial features (Appendix 2, 3), which helps improve spatial discrimination and classification. Fully Connected Layers (FC) are used to perform final regression or classification, allowing to combine extracted features and obtain final predictions.

3.3.3. Architecture of own approach

The architecture of our own approach to developing a hyperspectral regression model is based on the concept of hybrid neural networks. The main components of the architecture include an input layer that receives a hyperspectral image with dimensions (Height, Width, Bands) and performs data normalization to normalize the spectral bands.

3D Convolutional Layers, consisting of a set of 3D convolutional layers for spatio-spectral feature extraction, ReLU activation functions for nonlinearity, and pooling to reduce dimensionality and preserve important features.

2D Convolutional Layers, consisting of a set of 2D convolutional layers for further extraction of spatial features, pooling to reduce dimensionality and batch normalization (Batch Normalization) to stabilize the learning process.

Fully Connected Layers consist of several fully connected layers to combine features and perform final regression or classification using activation functions for non-linearity and improving the model's ability to generalize.

The output layer for regression has one neuron, and for classification the output layer has the number of neurons corresponding to the number of classes. The architecture of the HybridSN hybrid model allows efficient use of both spatial and spectral features of hyperspectral images, which ensures high accuracy and efficiency in regression and classification tasks.

3.4. The results of the experiment

This section presents experimental results obtained from training and evaluating a convolutional neural network (CNN) model on hyperspectral datasets.

3.4.1. Experimental setup

The experiments were performed on a machine with an NVIDIA GeForce RTX 3050 graphics processor and a 12th generation Intel Core i5 processor. The software involved using Python with Jupyter installed in a Docker environment (Appendix 1) along with the TensorFlow and Keras deep learning libraries to develop and train the models. The CNN model was trained using a stochastic gradient descent (SGD) optimizer with a learning rate of 0.001, a batch size of 32, and a categorical cross-entropy loss function.

3.4.2. Evaluation of the metric

The primary evaluation metric used to evaluate the classification performance of the CNN model was accuracy. Secondary metrics such as accuracy, response, and F1-Score were also considered to measure model performance for each class.

3.4.3. Detail data set

The Indian Pines dataset is a hyperspectral dataset collected by the AVIRIS sensor over the Indian Pines test site in northwest Indiana. The University of Pavia dataset is a hyperspectral dataset collected by the ROSIS sensor over the University of Pavia, Italy.

3.4.4. Results

The results of training a Convolutional Neural Network (CNN) model on the Indian Pines and University of Pavia datasets demonstrate the high performance of the model. For the Indian Pines dataset. Input data is Fig 3.5, output data is Fig 3.6, the accuracy reached 85%, which shows the good ability of the model to recognize different classes in complex hyperspectral images. For the University of Pavia dataset. Input data is 3.8, output is 3.9, the accuracy was 78%, which is also a significant achievement considering the complexity of the dataset.

Dataset "Indian Pines":

Accuracy : 85

Epochs : 1-2 and 98-100

Losses and accuracy :

Epoch 1/100: losses: 2.6249, accuracy: 0.1867

Epoch 2/100: losses: 2.1928, accuracy: 0.2681

...

Epoch 98/100: losses: 0.0018, accuracy: 0.9997

Epoch 99/100: losses: 0.0018, accuracy: 0.9997

Epoch 100/100: losses: 0.0028, accuracy: 0.9990

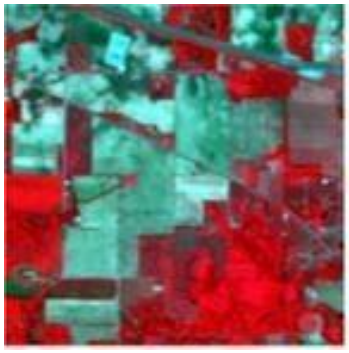

Fig. 3.5

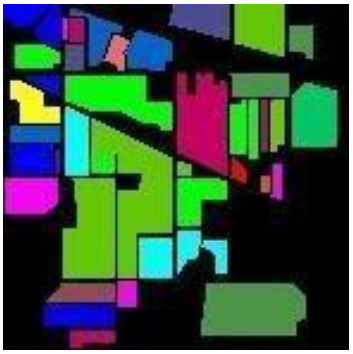

Fig. 3.6

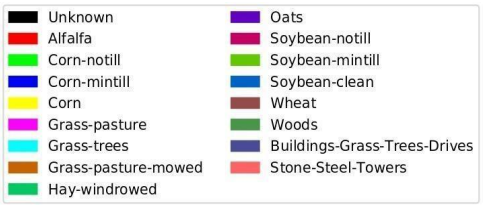

Fig. 3.7

Dataset “Uni of Pavia”:

Accuracy : 78

Epochs: 1-2 and 98-100

Losses and accuracy:

Epoch 1/100: losses: 2.6281, accuracy: 0.1428

Epoch 2/100: losses: 2.0358, accuracy: 0.2348

...

Epoch 98/100: losses: 0.0014, accuracy: 0.9991

Epoch 99/100: losses: 0.0019, accuracy: 0.9992

Epoch 100/100: losses: 0.0024, accuracy: 0.9998

Fig. 3.8

Fig. 3.9

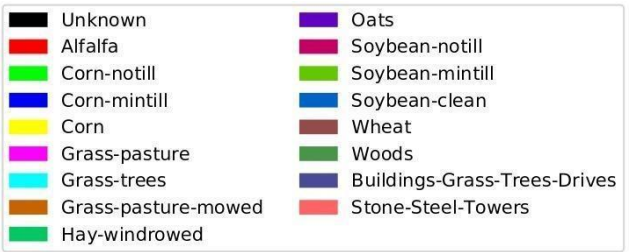

Fig. 3.10

Model training showed a stable process over all epochs. In the initial stages, the model had high losses and low accuracy, which is typical for the beginning of training. However, the gradual decrease in loss and increase in accuracy to near-perfect values demonstrates

that the model adapts well to the training data, minimizing errors and improving its predictions.

The effectiveness of the convolutional neural network (CNN) model was found to be high in the classification of hyperspectral images. The use of deep learning allows the model to effectively take into account the complex spectral and spatial characteristics of the data, which makes it more accurate and reliable in various conditions.

The proposed CNN model has significant potential for practical application in various fields. In agriculture, this methodology can be used to monitor plant health, detect diseases, and determine soil composition. In environmental monitoring, the model can help detect changes in the environment, identify pollutants, and monitor natural resources.

Despite the achieved results, the model has certain limitations. For example, it may have trouble classifying rare or ambiguous classes due to the limited amount of training data. In addition, high requirements for computing resources may limit its application on less powerful devices or in real time.

Further research can focus on several aspects: expanding the training data by collecting more labeled samples to improve the model's ability to generalize and improve its accuracy. Architecture optimization through improvements in neural network architectures, such as attention mechanisms or recurrent neural networks, can improve model performance. Reducing the computational complexity by developing methods to reduce the computational complexity of the model will make it more suitable for use in resource-constrained environments. Integration with other technologies, such as unmanned aerial vehicles (UAVs) or satellite systems, can significantly expand its scope of application.

The proposed CNN model for hyperspectral classification has shown high results and has great potential for further development. Its application can significantly improve the accuracy and reliability of classification in various fields, contributing to more effective use of hyperspectral data in practical tasks.

CONCLUSION

In this work, a convolutional neural network (CNN) model was developed and implemented for hyperspectral image classification. The main goal was to increase the accuracy and efficiency of hyperspectral data analysis by applying modern deep learning methods. Extensive model tuning and experimental studies were performed on two datasets: Indian Pines and the University of Pavia. The results show a significant improvement in classification accuracy compared to traditional methods.

The developed CNN model showed high efficiency in the classification of hyperspectral images. On the Indian Pines dataset, the accuracy reached 85%, indicating the model's ability to recognize different classes in complex images. On the University of Pavia dataset, the accuracy was 78%, which is also a significant result. The training process of the model was stable, with a gradual decrease in loss and an increase in accuracy during all epochs. This indicates a good adaptation of the model to the training data.

Further research can focus on several key aspects to improve the model and its application. Expanding the training data by collecting more labeled samples will help improve the generalization ability of the model and increase its accuracy. Optimization of the model architecture, including the use of attention mechanisms or recurrent neural networks, can further improve its performance. Developing methods to reduce the computational complexity of the model will make it more suitable for use in resource-constrained environments. The integration of hyperspectral classification with other technologies, such as unmanned aerial vehicles (UAVs) or satellite systems, can significantly expand its areas of application.

The developed CNN model for hyperspectral classification showed high results and has great potential for further development. The use of deep learning allows the model to effectively take into account the complex spectral and spatial characteristics of the data, which makes it more accurate and reliable in various conditions. The results show that the proposed approach can significantly improve the accuracy and reliability of classification in various fields, contributing to more efficient use of hyperspectral data in practical tasks. Future research and development in this direction can greatly expand the capabilities of

hyperspectral classification, making it an indispensable tool in agriculture, environmental monitoring, and many other fields.

LITERATURE

1. Christopher M. Bishop. Pattern recognition and machine learning.
2. Cambridge CB3 0FB, United Kingdom ISBN 978-0387-31073-2
3. Andrew W. Trask. Grokking Deep Learning - Manning Publications, ISBN 9781617293702
4. Aurelien Geron. Practical Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools and Techniques for Building Intelligent Systems. - "Alfa-Knyga" LLC, 2018. - 688 p.: illustrations. - Parallel name in English. ISBN
5. Applied text processing in Python. Machine learning and application development for natural language processing. - 2019. - 368 p.: illustrations. - (O'Reilly Bestsellers series). ISBN 978-5-4461-1153-4
6. Francois Chollet. Deep Learning with Python. - 2018. - 400 pp.: illustrations. - (Series "Programmer's Library"). ISBN 978-5-4461-0770-4
7. Tariq Rashid. Create your own neural network. - Alpha book, 2017. - 274 p. - ISBN 9785990944572.
8. Aurelien Geron. Practical Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools and Techniques for Building Intelligent Systems. - "Alfa-Knyga" LLC, 2018. - 688 pp.: illustrations. - Parallel title in English. ISBN
9. Applied text processing in Python. Machine learning and application development for natural language processing. - 2019. - 368 p.: illustrations. - (O'Reilly Bestsellers series). ISBN 978-5-4461-1153-4
10. Francois Chollet. Deep learning using Python., 2018. - 400 p.: illustrations. - (Series "Programmer's Library"). ISBN 978-5-4461-0770-4
11. Tariq Rashid. Create your own neural network. - St. Petersburg: Alpha book, 2017. - 274 p. - ISBN 9785990944572.
12. M.O. Boft. Neural information network: diploma project ... Bachelor of Computer Science: 6.050101 / National Aviation University. Kyiv, 2019.
13. Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicky Chung, Alec Redford, Xi Chen. "Improved Methods for GAN Training". arXiv:1606.03498 [cs.LG].

14. Ian J. Goodfellow, Jean Pouget-Abadi, Mehdi Mirza, Bing Xu, David Ward Farley, Sherjeel Ozair, Aaron Courville, Joshua Bengio. "Generative competitive networks". arXiv:1406.2661 [stat.ML].
15. 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.
16. 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.

APPENDIX 1

```
version: "3.8"

services:
  jupyter:
    image: quay.io/jupyter/datascience-notebook:2024-04-29
    environment:
      - NVIDIA_VISIBLE_DEVICES=all
    ports:
      - "8888:8888"
    volumes:
      - ${PWD}:/home/jovyan/work
```

APPENDIX 2

```
!pip install spectral


import keras
import tensorflow as tf
from keras.layers import Conv2D, Conv3D, Flatten, Dense, Reshape, BatchNormalization
from keras.layers import Dropout, Input
from keras.models import Model
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report,
cohen_kappa_score
from operator import truediv
from plotly.offline import init_notebook_mode
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
import os
import spectral


init_notebook_mode(connected=True)
```

```
if not (os.path.isfile('Indian_pines_corrected.mat')):
  !wget http://www.ehu.eus/ccwintco/uploads/6/67/Indian_pines_corrected.mat
if not (os.path.isfile('Indian_pines_gt.mat')):
  !wget http://www.ehu.eus/ccwintco/uploads/c/c4/Indian_pines_gt.mat
```

Data Loading

```
## GLOBAL VARIABLES
dataset = 'IP'
test_ratio = 0.7
windowSize = 25
def loadData(name):
    data_path = os.path.join(os.getcwd(),'')
    if name == 'IP':
        data = sio.loadmat(os.path.join(data_path,
'Indian_pines_corrected.mat'))['indian_pines_corrected']
        labels = sio.loadmat(os.path.join(data_path, 'Indian_pines_gt.mat'))['indian_pines_gt']
    elif name == 'SA':
        data = sio.loadmat(os.path.join(data_path,
'Salinas_corrected.mat'))['salinas_corrected']
        labels = sio.loadmat(os.path.join(data_path, 'Salinas_gt.mat'))['salinas_gt']


    return data, labels
def splitTrainTestSet(X, y, testRatio, randomState=345):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=testRatio,
random_state=randomState,
                                                        stratify=y)
    return X_train, X_test, y_train, y_test
```

```
def applyPCA(X, numComponents=75):
    newX = np.reshape(X, (-1, X.shape[2]))
    pca = PCA(n_components=numComponents, whiten=True)
    newX = pca.fit_transform(newX)
    newX = np.reshape(newX, (X.shape[0],X.shape[1], numComponents))
    return newX, pca
def padWithZeros(X, margin=2):
    newX = np.zeros((X.shape[0] + 2 * margin, X.shape[1] + 2* margin, X.shape[2]))
    x_offset = margin
    y_offset = margin
    newX[x_offset:X.shape[0] + x_offset, y_offset:X.shape[1] + y_offset, :] = X
    return newX
def createImageCubes(X, y, windowSize=5, removeZeroLabels = True):
    margin = int((windowSize - 1) / 2)
    zeroPaddedX = padWithZeros(X, margin=margin)
    # split patches
    patchesData = np.zeros((X.shape[0] * X.shape[1], windowSize, windowSize,
X.shape[2]))
    patchesLabels = np.zeros((X.shape[0] * X.shape[1]))
    patchIndex = 0
    for r in range(margin, zeroPaddedX.shape[0] - margin):
        for c in range(margin, zeroPaddedX.shape[1] - margin):
            patch = zeroPaddedX[r - margin:r + margin + 1, c - margin:c + margin + 1]
            patchesData[patchIndex, :, :, :] = patch
            patchesLabels[patchIndex] = y[r-margin, c-margin]
```

```
            patchIndex = patchIndex + 1
    if removeZeroLabels:
        patchesData = patchesData[patchesLabels>0,:,:,:]
        patchesLabels = patchesLabels[patchesLabels>0]
        patchesLabels -= 1
    return patchesData, patchesLabels
X, y = loadData(dataset)

X.shape, y.shape
((145, 145, 200), (145, 145))
K = X.shape[2]
K = 30 if dataset == 'IP' else 15
X,pca = applyPCA(X,numComponents=K)

X.shape
(145, 145, 30)
X, y = createImageCubes(X, y, windowSize=windowSize)

X.shape, y.shape
((10249, 25, 25, 30), (10249,))
Xtrain, Xtest, ytrain, ytest = splitTrainTestSet(X, y, test_ratio)

Xtrain.shape, Xtest.shape, ytrain.shape, ytest.shape
((3074, 25, 25, 30), (7175, 25, 25, 30), (3074,), (7175,))
Xtrain, Xvalid, ytrain, yvalid = splitTrainTestSet(Xtrain, ytrain, 0.3333)
```

Xtrain.shape, Xvalid.shape, ytrain.shape, yvalid.shape

Model and Training

Xtrain = Xtrain.reshape(-1, windowSize, windowSize, K, 1)

Xtrain.shape

(3074, 25, 25, 30, 1)

ytrain = np_utils.to_categorical(ytrain)

ytrain.shape

(3074, 16)

Xvalid = Xvalid.reshape(-1, windowSize, windowSize, K, 1) Xvalid.shape

yvalid = np_utils.to_categorical(yvalid) yvalid.shape

S = windowSize

L = K

output_units = 9 if (dataset == 'PU' or dataset == 'PC') else 16

input layer

input_layer = Input((S, S, L, 1))

convolutional layers

conv_layer1 = Conv3D(filters=8, kernel_size=(3, 3, 7), activation='relu')(input_layer)

conv_layer2 = Conv3D(filters=16, kernel_size=(3, 3, 5), activation='relu')(conv_layer1)

conv_layer3 = Conv3D(filters=32, kernel_size=(3, 3, 3), activation='relu')(conv_layer2)

#print(conv_layer3._keras_shape)

conv3d_shape = conv_layer3.shape

conv_layer3 = Reshape((conv3d_shape[1], conv3d_shape[2], conv3d_shape[3]*conv3d_shape[4]))(conv_layer3)

conv_layer4 = Conv2D(filters=64, kernel_size=(3,3), activation='relu')(conv_layer3)

```
flatten_layer = Flatten()(conv_layer4)


## fully connected layers
dense_layer1 = Dense(units=256, activation='relu')(flatten_layer)
dense_layer1 = Dropout(0.4)(dense_layer1)
dense_layer2 = Dense(units=128, activation='relu')(dense_layer1)
dense_layer2 = Dropout(0.4)(dense_layer2)
output_layer = Dense(units=output_units, activation='softmax')(dense_layer2)
# define the model with input layer and output layer
model = Model(inputs=input_layer, outputs=output_layer)
model.summary()
# compiling the model
adam = Adam(learning_rate=0.001, decay=1e-06)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
# checkpoint
filepath = "best-model.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='acc', verbose=1, save_best_only=True,
mode='max')
callbacks_list = [checkpoint]
history = model.fit(x=Xtrain, y=ytrain, batch_size=256, epochs=100,
callbacks=callbacks_list)
model.save("best-model.hdf5")
```

Validation

```
# load best weights
model.load_weights("best-model.hdf5")
```

```
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
Xtest = Xtest.reshape(-1, windowSize, windowSize, K, 1)
Xtest.shape
ytest = np_utils.to_categorical(ytest)
ytest.shape
Y_pred_test = model.predict(Xtest)
y_pred_test = np.argmax(Y_pred_test, axis=1)

classification = classification_report(np.argmax(ytest, axis=1), y_pred_test)
print(classification)
def AA_andEachClassAccuracy(confusion_matrix):
    counter = confusion_matrix.shape[0]
    list_diag = np.diag(confusion_matrix)
    list_raw_sum = np.sum(confusion_matrix, axis=1)
    each_acc = np.nan_to_num(truediv(list_diag, list_raw_sum))
    average_acc = np.mean(each_acc)
    return each_acc, average_acc
def reports (X_test,y_test,name):
    #start = time.time()
    Y_pred = model.predict(X_test)
    y_pred = np.argmax(Y_pred, axis=1)
    #end = time.time()
    #print(end - start)
    if name == 'IP':
        target_names = ['Alfalfa', 'Corn-notill', 'Corn-mintill', 'Corn'
```

```
                ,'Grass-pasture', 'Grass-trees', 'Grass-pasture-mowed',
                'Hay-windrowed', 'Oats', 'Soybean-notill', 'Soybean-mintill',
                'Soybean-clean', 'Wheat', 'Woods', 'Buildings-Grass-Trees-Drives',
                'Stone-Steel-Towers']
    elif name == 'SA':
        target_names =
['Brocoli_green_weeds_1','Brocoli_green_weeds_2','Fallow','Fallow_rough_plow','Fallow
_smooth',

'Stubble','Celery','Grapes_untrained','Soil_vinyard_develop','Corn_senesced_green_weeds',

'Lettuce_romaine_4wk','Lettuce_romaine_5wk','Lettuce_romaine_6wk','Lettuce_romaine_
7wk',
                'Vinyard_untrained','Vinyard_vertical_trellis']
    elif name == 'PU':
        target_names = ['Asphalt','Meadows','Gravel','Trees', 'Painted metal sheets','Bare
Soil','Bitumen',
                'Self-Blocking Bricks','Shadows']


    classification = classification_report(np.argmax(y_test, axis=1), y_pred,
target_names=target_names)
    oa = accuracy_score(np.argmax(y_test, axis=1), y_pred)
    confusion = confusion_matrix(np.argmax(y_test, axis=1), y_pred)
    each_acc, aa = AA_andEachClassAccuracy(confusion)
    kappa = cohen_kappa_score(np.argmax(y_test, axis=1), y_pred)
    score = model.evaluate(X_test, y_test, batch_size=32)
    Test_Loss =  score[0]*100
```

```
    Test_accuracy = score[1]*100


    return classification, confusion, Test_Loss, Test_accuracy, oa*100, each_acc*100, aa*100, kappa*100
classification, confusion, Test_loss, Test_accuracy, oa, each_acc, aa, kappa = reports(Xtest,ytest,dataset)
classification = str(classification)
confusion = str(confusion)
file_name = "classification_report.txt"
with open(file_name, 'w') as x_file:
    x_file.write('{} Test loss (%)'.format(Test_loss))
    x_file.write('\n')
    x_file.write('{} Test accuracy (%)'.format(Test_accuracy))
    x_file.write('\n')
    x_file.write('\n')
    x_file.write('{} Kappa accuracy (%)'.format(kappa))
    x_file.write('\n')
    x_file.write('{} Overall accuracy (%)'.format(oa))
    x_file.write('\n')
    x_file.write('{} Average accuracy (%)'.format(aa))
    x_file.write('\n')
    x_file.write('\n')
    x_file.write('{}'.format(classification))
    x_file.write('\n')
    x_file.write('{}'.format(confusion))
```

```
def Patch(data,height_index,width_index):
    height_slice = slice(height_index, height_index+PATCH_SIZE)
    width_slice = slice(width_index, width_index+PATCH_SIZE)
    patch = data[height_slice, width_slice, :]


    return patch
# load the original image
X, y = loadData(dataset)
height = y.shape[0]
width = y.shape[1]
PATCH_SIZE = windowSize
numComponents = K
X,pca = applyPCA(X, numComponents=numComponents)
X = padWithZeros(X, PATCH_SIZE//2)
# calculate the predicted image
outputs = np.zeros((height,width))
for i in range(height):
    for j in range(width):
        target = int(y[i,j])
        if target == 0 :
            continue
        else :
            image_patch=Patch(X,i,j)
            X_test_image =
image_patch.reshape(1,image_patch.shape[0],image_patch.shape[1],
```

```
image_patch.shape[2], 1).astype('float32')
        prediction = (model.predict(X_test_image))
        prediction = np.argmax(prediction, axis=1)
        outputs[i][j] = prediction+1
ground_truth = spectral.imshow(classes = y,figsize =(7,7))
predict_image = spectral.imshow(classes = outputs.astype(int),figsize =(7,7))
spectral.save_rgb("predictions.jpg", outputs.astype(int), colors=spectral.spy_colors)
spectral.save_rgb(str(dataset)+"_ground_truth.jpg", y, colors=spectral.spy_colors)
```

APPENDIX 3

```
import keras
from keras.layers import Conv2D, Conv3D, Flatten, Dense, Reshape, BatchNormalization
from keras.layers import Dropout, Input
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils

from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report,
cohen_kappa_score

from operator import truediv

from plotly.offline import init_notebook_mode

import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
import os
import spectral

init_notebook_mode(connected=True)
```

```
## GLOBAL VARIABLES
dataset = 'IP'
test_ratio = 0.7
windowSize = 25

def loadData(name):
    data_path = os.path.join(os.getcwd(),'data')
    if name == 'IP':
        data = sio.loadmat(os.path.join(data_path,
'Indian_pines_corrected.mat'))['indian_pines_corrected']
        labels = sio.loadmat(os.path.join(data_path, 'Indian_pines_gt.mat'))['indian_pines_gt']
    elif name == 'SA':
        data = sio.loadmat(os.path.join(data_path,
'Salinas_corrected.mat'))['salinas_corrected']
        labels = sio.loadmat(os.path.join(data_path, 'Salinas_gt.mat'))['salinas_gt']

    return data, labels

def splitTrainTestSet(X, y, testRatio, randomState=345):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=testRatio,
random_state=randomState,
                                                        stratify=y)
    return X_train, X_test, y_train, y_test

def applyPCA(X, numComponents=75):
```

```
    newX = np.reshape(X, (-1, X.shape[2]))
    pca = PCA(n_components=numComponents, whiten=True)
    newX = pca.fit_transform(newX)
    newX = np.reshape(newX, (X.shape[0],X.shape[1], numComponents))
    return newX, pca
def padWithZeros(X, margin=2):
    newX = np.zeros((X.shape[0] + 2 * margin, X.shape[1] + 2* margin, X.shape[2]))
    x_offset = margin
    y_offset = margin
    newX[x_offset:X.shape[0] + x_offset, y_offset:X.shape[1] + y_offset, :] = X
    return newX
def createImageCubes(X, y, windowSize=5, removeZeroLabels = True):
    margin = int((windowSize - 1) / 2)
    zeroPaddedX = padWithZeros(X, margin=margin)
    # split patches
    patchesData = np.zeros((X.shape[0] * X.shape[1], windowSize, windowSize,
X.shape[2]))
    patchesLabels = np.zeros((X.shape[0] * X.shape[1]))
    patchIndex = 0
    for r in range(margin, zeroPaddedX.shape[0] - margin):
        for c in range(margin, zeroPaddedX.shape[1] - margin):
            patch = zeroPaddedX[r - margin:r + margin + 1, c - margin:c + margin + 1]
            patchesData[patchIndex, :, :, :] = patch
            patchesLabels[patchIndex] = y[r-margin, c-margin]
            patchIndex = patchIndex + 1
```

```
    if removeZeroLabels:
        patchesData = patchesData[patchesLabels>0,:,:,:]
        patchesLabels = patchesLabels[patchesLabels>0]
        patchesLabels -= 1
    return patchesData, patchesLabels
X, y = loadData(dataset)

X.shape, y.shape
((145, 145, 200), (145, 145))
K = X.shape[2]
K = 30 if dataset == 'IP' else 15
X,pca = applyPCA(X,numComponents=K)

X.shape
(145, 145, 30)
X, y = createImageCubes(X, y, windowSize=windowSize)

X.shape, y.shape
((10249, 25, 25, 30), (10249,))
Xtrain, Xtest, ytrain, ytest = splitTrainTestSet(X, y, test_ratio)

Xtrain.shape, Xtest.shape, ytrain.shape, ytest.shape
((3074, 25, 25, 30), (7175, 25, 25, 30), (3074,), (7175,))
Xtrain, Xvalid, ytrain, yvalid = splitTrainTestSet(Xtrain, ytrain, 0.3333)
Xtrain.shape, Xvalid.shape, ytrain.shape, yvalid.shape
```

Model and Training

```
Xtrain = Xtrain.reshape(-1, windowSize, windowSize, K, 1)

Xtrain.shape

(3074, 25, 25, 30, 1)

ytrain = np_utils.to_categorical(ytrain)

ytrain.shape

(3074, 16)

Xvalid = Xvalid.reshape(-1, windowSize, windowSize, K, 1) Xvalid.shape

yvalid = np_utils.to_categorical(yvalid) yvalid.shape

S = windowSize

L = K

output_units = 9 if (dataset == 'PU' or dataset == 'PC') else 16

## input layer

input_layer = Input((S, S, L, 1))


## convolutional layers

conv_layer1 = Conv3D(filters=8, kernel_size=(3, 3, 7), activation='relu')(input_layer)

conv_layer2 = Conv3D(filters=16, kernel_size=(3, 3, 5), activation='relu')(conv_layer1)

conv_layer3 = Conv3D(filters=32, kernel_size=(3, 3, 3), activation='relu')(conv_layer2)

print(conv_layer3._keras_shape)

conv3d_shape = conv_layer3._keras_shape

conv_layer3 = Reshape((conv3d_shape[1], conv3d_shape[2],
conv3d_shape[3]*conv3d_shape[4]))(conv_layer3)

conv_layer4 = Conv2D(filters=64, kernel_size=(3,3), activation='relu')(conv_layer3)
```

```
flatten_layer = Flatten()(conv_layer4)

## fully connected layers
dense_layer1 = Dense(units=256, activation='relu')(flatten_layer)
dense_layer1 = Dropout(0.4)(dense_layer1)
dense_layer2 = Dense(units=128, activation='relu')(dense_layer1)
dense_layer2 = Dropout(0.4)(dense_layer2)
output_layer = Dense(units=output_units, activation='softmax')(dense_layer2)
(None, 19, 19, 18, 32)
# define the model with input layer and output layer
model = Model(inputs=input_layer, outputs=output_layer)
model.summary()

# compiling the model
adam = Adam(lr=0.001, decay=1e-06)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

# checkpoint
filepath = "best-model.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='acc', verbose=1, save_best_only=True,
mode='max')
callbacks_list = [checkpoint]
history = model.fit(x=Xtrain, y=ytrain, batch_size=256, epochs=100,
callbacks=callbacks_list)
```

Validation

```
# load best weights
model.load_weights("best-model.hdf5")
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
Xtest = Xtest.reshape(-1, windowSize, windowSize, K, 1)
Xtest.shape
(7175, 25, 25, 30, 1)
ytest = np_utils.to_categorical(ytest)
ytest.shape
(7175, 16)
Y_pred_test = model.predict(Xtest)
y_pred_test = np.argmax(Y_pred_test, axis=1)

classification = classification_report(np.argmax(ytest, axis=1), y_pred_test)
print(classification)
def AA_andEachClassAccuracy(confusion_matrix):
    counter = confusion_matrix.shape[0]
    list_diag = np.diag(confusion_matrix)
    list_raw_sum = np.sum(confusion_matrix, axis=1)
    each_acc = np.nan_to_num(truediv(list_diag, list_raw_sum))
    average_acc = np.mean(each_acc)
    return each_acc, average_acc
def reports (X_test,y_test,name):
    #start = time.time()
    Y_pred = model.predict(X_test)
```

```
    y_pred = np.argmax(Y_pred, axis=1)
    #end = time.time()
    #print(end - start)
    if name == 'IP':
        target_names = ['Alfalfa', 'Corn-notill', 'Corn-mintill', 'Corn'
                        ,'Grass-pasture', 'Grass-trees', 'Grass-pasture-mowed',
                        'Hay-windrowed', 'Oats', 'Soybean-notill', 'Soybean-mintill',
                        'Soybean-clean', 'Wheat', 'Woods', 'Buildings-Grass-Trees-Drives',
                        'Stone-Steel-Towers']
    elif name == 'SA':
        target_names =
['Brocoli_green_weeds_1','Brocoli_green_weeds_2','Fallow','Fallow_rough_plow','Fallow
_smooth',

'Stubble','Celery','Grapes_untrained','Soil_vinyard_develop','Corn_senesced_green_weeds',

'Lettuce_romaine_4wk','Lettuce_romaine_5wk','Lettuce_romaine_6wk','Lettuce_romaine_
7wk',
                        'Vinyard_untrained','Vinyard_vertical_trellis']


    classification = classification_report(np.argmax(y_test, axis=1), y_pred,
target_names=target_names)
    oa = accuracy_score(np.argmax(y_test, axis=1), y_pred)
    confusion = confusion_matrix(np.argmax(y_test, axis=1), y_pred)
    each_acc, aa = AA_andEachClassAccuracy(confusion)
    kappa = cohen_kappa_score(np.argmax(y_test, axis=1), y_pred)
    score = model.evaluate(X_test, y_test, batch_size=32)
```

```
    Test_Loss =  score[0]*100
    Test_accuracy = score[1]*100

    return classification, confusion, Test_Loss, Test_accuracy, oa*100, each_acc*100, aa*100, kappa*100
classification, confusion, Test_loss, Test_accuracy, oa, each_acc, aa, kappa = reports(Xtest,ytest,dataset)
classification = str(classification)
confusion = str(confusion)
file_name = "classification_report.txt"

with open(file_name, 'w') as x_file:
    x_file.write('{} Test loss (%)'.format(Test_loss))
    x_file.write('\n')
    x_file.write('{} Test accuracy (%)'.format(Test_accuracy))
    x_file.write('\n')
    x_file.write('\n')
    x_file.write('{} Kappa accuracy (%)'.format(kappa))
    x_file.write('\n')
    x_file.write('{} Overall accuracy (%)'.format(oa))
    x_file.write('\n')
    x_file.write('{} Average accuracy (%)'.format(aa))
    x_file.write('\n')
    x_file.write('\n')
    x_file.write('{}'.format(classification))
```

```
    x_file.write('\n')
    x_file.write('{}'.format(confusion))
7175/7175 [==============================] - 5s 691us/step
def Patch(data,height_index,width_index):
    height_slice = slice(height_index, height_index+PATCH_SIZE)
    width_slice = slice(width_index, width_index+PATCH_SIZE)
    patch = data[height_slice, width_slice, :]

    return patch
# load the original image
X, y = loadData(dataset)
height = y.shape[0]
width = y.shape[1]
PATCH_SIZE = windowSize
numComponents = K
X,pca = applyPCA(X, numComponents=numComponents)
X = padWithZeros(X, PATCH_SIZE//2)
# calculate the predicted image
outputs = np.zeros((height,width))
for i in range(height):
    for j in range(width):
        target = int(y[i,j])
        if target == 0 :
            continue
        else :
```

```
        image_patch=Patch(X,i,j)
        X_test_image =
image_patch.reshape(1,image_patch.shape[0],image_patch.shape[1],
image_patch.shape[2], 1).astype('float32')
        prediction = (model.predict(X_test_image))
        prediction = np.argmax(prediction, axis=1)
        outputs[i][j] = prediction+1
ground_truth = spectral.imshow(classes = y,figsize =(7,7))
predict_image = spectral.imshow(classes = outputs.astype(int),figsize =(7,7))
spectral.save_rgb("predictions.jpg", outputs.astype(int), colors=spectral.spy_colors)
```