Enhancing clothing pattern classification using neural networks and transfer learning

Vasyl Teslyuk[†] and Stanislav Ivasiv[*,†]

Lviv Polytechnic National University, 12 S. Bandera Str., Lviv, Ukraine

Abstract

The task of pattern classification remains relevant in the fields of trends, style, fashion, personalization, manufacturing, and design. Research aimed at the design and development of models and means of classification of patterns of clothing elements using machine learning is highlighted. The study addresses a pertinent issue in computer vision, namely: increasing the efficiency of classification of patterns of clothing elements. The research was conducted with a proprietary dataset comprising 600 images. The following patterns are defined for classification: "checkered", "dotted", "vegetation/floral", "print", "solid", "striped". A convolutional neural network was developed using the Python programming language and deep learning frameworks Keras and TensorFlow. The scalable Keras-Tuner framework was used to optimize the hyperparameters of the developed network. The structure of the convolutional neural network includes an input layer, a feature extraction part, and a pattern type determination part. The architecture of the applied convolutional neural network is described. The CUDA Toolkit, the cuDNN library and the WSL layer are applied to train a convolutional neural network using a GPU, significantly speeding up the training process. Metrics including accuracy, precision, and recall were used to evaluate the developed convolutional neural network. The web application is developed in the Python programming language with the FastAPI framework. The web application has a described API for interacting with a convolutional neural network, and uses the Pillow (PIL) libraries for working with images and Rembg for image background removal. The user interface is developed in the JavaScript programming language with HTML, CSS and the React framework. The user interface is presented as an intuitive tool for interacting with the system. The developed software uses the modular principle, which allows for rapid modernization of the software. As a result of applying transfer learning, a testing accuracy of 93.33% was achieved, and with fine-tuning, the final version of the convolutional neural network for the classification of patterns of clothing elements with a test accuracy of 95% was obtained. The trained neural network was tested on new images of the specified types of patterns, examples for two patterns are given.

Keywords

Visual Geometry Group 16 (VGG16), patterns classification, fine-tuning, transfer learning, convolutional neural network

1. Introduction

Recognition of patterns on clothing items is a significant and relevant aspect for various fields, such as fashion, manufacturing, design, and technology. Some aspects of the relevance of this issue are revealed in the topics of trends and style [1], production automation [2], personalization [3 - 5], augmented reality [6, 7], production optimization [8], and anti-counterfeiting [9]. Insights [10] can provide valuable frameworks to ensure that technological systems for clothing pattern classification are efficient and adaptable to local strategies and expert input, fostering a holistic integration of innovative technology and practical applications in the fashion industry.

In general, pattern recognition on clothing makes it possible to combine traditional elements with innovative technologies, making the fashion world more efficient, interesting, and responsible.

The primary objective of the study is the classification of clothing pattern elements, which relies on the selected neural network architecture, its hyperparameters, and the training approaches used;

CIAW-2024: Computational Intelligence Application Workshop, October 10-12, 2024, Lviv, Ukraine

[*] Corresponding author.

[†] These authors contributed equally.

vasyl.m.teslyuk@lpnu.ua (V. Teslyuk); stanislav.s.ivasiv@lpnu.ua (S. Ivasiv)

0000-0002-5974-9310 (V. Teslyuk); 0009-0007-3406-7376 (S. Ivasiv)

an equally important factor is the dataset employed for training, validation, and testing of the neural network.

Currently, there are a number of applications of pattern recognition and classification, and approaches to their study are constantly being improved and applied to various fields of activity, so the topic under study and the development of models and tools for pattern classification using machine learning are relevant.

The object of research is the process of classifying patterns of clothing items using machine learning.

The subject of research is models and tools for classifying patterns of clothing elements using machine learning.

The study aims to explore and implement models and tools to enhance the efficiency of clothing pattern classification through machine learning.

To achieve this goal, the following main research objectives have been identified:

- review of scientific research on the classification of clothing item patterns;
- synthesis and evaluation of new models based on machine learning methods;
- development of tools for classifying clothing item patterns using machine learning and research.

2. Materials and methods of the study

The research employed the object-oriented approach for application development and the supervised machine learning method, utilizing data structures such as dictionaries, lists, and tuples. In applying artificial neural networks, principles of convolutional neural networks [11], modern CNN architectures, and aspects of linear algebra, mathematical analysis, optimization fundamentals, and probability theory were used.

The study was carried out using the Python programming language in the PyCharm and Jupyter Notebook environments [12]. The TensorFlow and Keras frameworks were used to organize the artificial neural network [13, 14]. The CUDA toolkit, cuDNN library, and WSL layer are used to train artificial neural networks using a GPU [15 - 17]. The application is based on the FastAPI framework [18]. The user interface was developed using the JavaScript programming language and the React framework in the WebStorm environment. The Pillow (PIL) and Rembg libraries [19] were used to remove the background of images. Graphs were created using the matplotlib library [20].

3. Analysis of recent research and publications

The paper [21] describes approaches to determining the location of clothing in an image, classifying a clothing item, and determining the color of a clothing item. However, the interaction with the applied neural network for determining the location of clothing and its classification occurs through the application program interface provided by the Clarifai platform. The platform offers a number of networks, but does not provide access to the architecture of the models, so it is not possible to change the architecture and retrain the network on its own data.

The paper [22], which describes the problems and solutions for pattern classification, was analyzed. In this article, the authors explore the use of a finger-mounted camera to recognize colors and visual patterns on clothing. The main focus is on neural network models, in particular the use of transfer learning to train a deep neural network. The authors describe the process of collecting data and creating a dataset for model training. They use six common visual patterns that are difficult or impossible to distinguish by touch: solid, striped, checkered, dotted, zigzag, and floral. The authors use the ResNet-101 deep neural network model, pre-trained on the public ImageNet dataset. The transfer learning technique is used to train the model on its own dataset. The classification accuracy of the image set from the finger-mounted camera reaches 92%. The authors identify confusion issues due to insufficient context or coarse threads and aim to improve the model using additional images from the target domain. The authors have plans for further research, such as studying the effect of the camera position on the user's finger on the accuracy of the system, as well as evaluating the details of the patterns.

The paper [23] describes image processing algorithms rather than neural networks. The authors consider the problem of identifying clothing design classes in images by using new image descriptors. They propose two new descriptors: Completed CENsus Transform hISTogram (cCENTRIST) and Ternary CENsus Transform hISTogram (tCENTRIST), which are based on existing image processing methods such as Completed Local Binary Pattern (CLBP), Local Ternary Pattern (LTP) and CENsus TRanformed hISTogram (CENTRIST). The authors investigated the effectiveness of the proposed methods by comparing them with other well-known methods such as Histogram of Oriented Gradients (HOG), GIST, Local Gradient Pattern (LGP) and the original CENTRIST method. They also compared their methods using pre-trained deep learning models such as CaffeNet. The authors succeeded in improving the accuracy of clothing design classification in images using the proposed methods. For the Clothing Attribute Dataset [24], the cCENTRIST method showed an accuracy of 74.97%, and the highest accuracy of 84.23% was achieved on the Fashion Dataset [25]. It has also been shown that the proposed methods outperform the results obtained using pre-trained deep learning models.

The paper [26] is considered. The main goal of this study is to improve the accuracy of clothing pattern classification. The authors propose their model, which is based on the AlexNet and VGG_S models, where all nine trained layers are used. Compared to AlexNet, a new fully connected layer (FC3) is added to the model, and a data augmentation technique is used to reduce overfitting during the training phase. The images were taken from two publicly available datasets: Fashion [25] and Clothing Attribute [24]. Both datasets were categorized for training and testing to identify clothing design classes. The authors compare the performance of the proposed model with two well-known CNN models (AlexNet and VGGNet) and some advanced manually developed feature extraction methods. The experimental environment for this study was set up using the CaffeNet model and the Ubuntu 12.4 operating system. A GPU was used to speed up the computations in the study. The proposed model achieved an accuracy of 77.8% on the Clothing Attribute Dataset with 6 different classes, and an accuracy of 84.5% on the Fashion Dataset with 5 texture classes. This indicates the success of using deep convolutional neural networks in the classification of clothing texture design and opens up prospects for further research in this area.

Based on the analysis, we can conclude that the accuracy of clothing element patterns classification needs to be improved, so it is worth using modern neural network architectures, selecting hyperparameters to obtain higher classification accuracy than that demonstrated in the reviewed sources [27, 28].

4. Research results

4.1. Preparation of the environment

One personal computer with the Windows operating system was used for development. In order to train the artificial neural network using a graphics processor, we used the Windows subsystem for Linux, which allows us to run Linux as a guest system within the Windows OS. The CUDA toolkit and the cuDNN library are installed in the guest system. To prepare an environment that allows you to use the GPU for computing, the following sequence of actions was performed.

1. the Nvidia GPU driver was installed;
2. download the WSL program to Windows;
3. launch the command prompt and run the command to install the Ubuntu distribution as a guest system:

- wsl –install

4. specify the name of the guest system user and the password of the guest system. After that, the current command line contains the running Linux Ubuntu guest system and corresponds to the command line of the guest system;
5. delete the old GPG key with the command:

- sudo apt-key del 7fa2af80

6. install the CUDA Toolkit 11.8, run a number of commands:

- wget https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/cuda-wsl-ubuntu.pin
- sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-repository-pin-600
- wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda-repo-wsl-ubuntu-11-8-local_11.8.0-1_amd64.deb
- sudo dpkg -i cuda-repo-wsl-ubuntu-11-8-local_11.8.0-1_amd64.deb
- sudo cp /var/cuda-repo-wsl-ubuntu-11-8-local/cuda-*-keyring.gpg /usr/share/keyrings/
- sudo apt-get update
- sudo apt-get -y install cuda

7. check if an environment variable has been added that indicates where to look for the executables of the CUDA 11.8 toolkit, run the command:

- echo $PATH

Added the path to the CUDA executables using the command:

- export PATH="/usr/local/cuda-11.8/bin:$PATH”

8. checked whether the “/usr/local/cuda-11.8” directory contains the “include” and “lib64” directories;
9. downloaded an archive with the CUDA Deep Neural Network library (cuDNN) version 8.7 specifically for CUDA Toolkit 11.8. The archive was moved to the user directory of the guest system;
10. extract the files from the archive containing the cuDNN library and move the corresponding files to the CUDA Toolkit 11.8 location by running the following commands:

- cd /home/YOUR_UBUNTU_USER
- tar -xvf cudnn-linux-x86_64-8.9.6.50_cuda11-archive.tar.xz
- sudo cp cudnn-linux-x86_64-8.9.6.50_cuda11-archive/include/cudnn*.h /usr/local/cuda-11.8/include
- sudo cp -P cudnn-linux-x86_64-8.9.6.50_cuda11-archive/lib/libcudnn* /usr/local/cuda-11.8/lib64
- sudo chmod a+r /usr/local/cuda-11.8/include/cudnn*.h /usr/local/cuda-11.8/lib64/libcudnn*

11. run a command to check if CUDA is installed:

- python3 -c “import tensorflow as tf; print(tf.config.list_physical_devices(‘GPU’))”

The line “[PhysicalDevice(name=‘/physical_device:GPU:0’, device_type=‘GPU’)]” is received, which indicates that the tools and libraries have been successfully configured.

4.2. The general structure of the software

In this paper, software was developed for the classification of patterns of clothing elements. In particular, Figure 1 shows a component diagram that demonstrates the dependencies between system components.

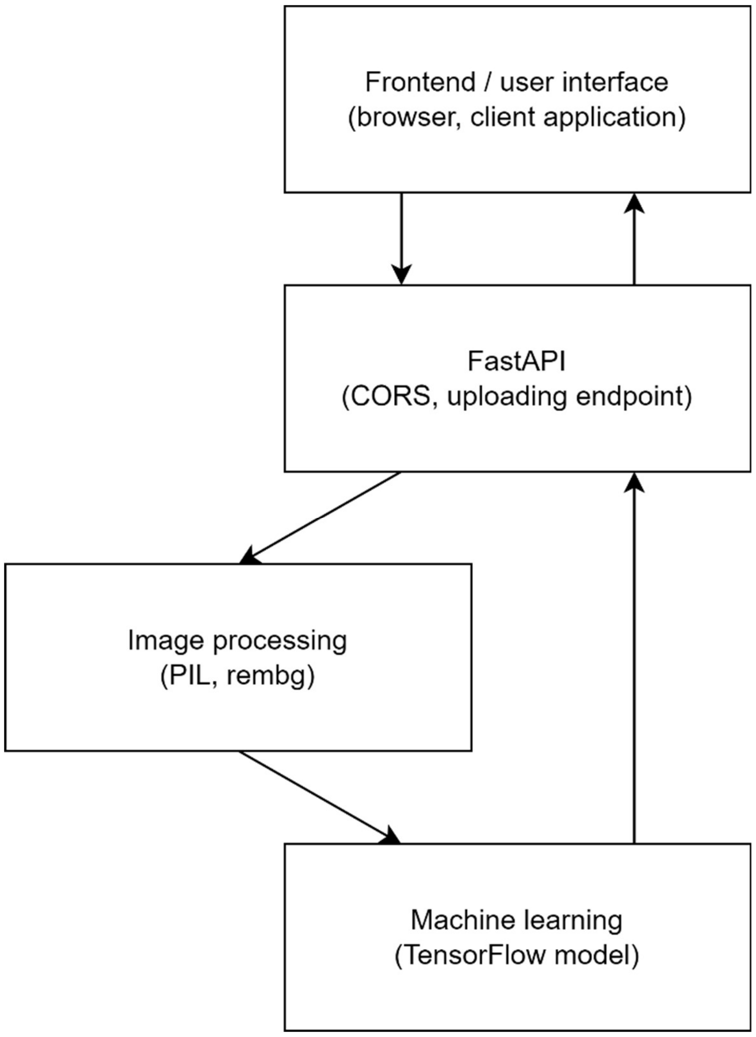

Frontend / user interface
(browser, client application)
FastAPI
(CORS, uploading endpoint)
Image processing
(PIL, rembg)
Machine learning
(TensorFlow model)

Figure 1: Diagram of system components.

The component diagram includes the following components:

- frontend/user interface represents the client side of the web application, where the user interacts with the web application through a browser or other client application;
- FastAPI is responsible for processing HTTP requests and responses. It includes a CORS module that allows you to interact with the application from different sources. Upload Endpoint is an HTTP route that handles file uploads. When files are uploaded, images are processed for further use in the machine learning model;
- image processing includes image processing libraries such as Pillow (PIL, Python Imaging Library) for image manipulation and Rembg library for background removal;
- machine learning includes a TensorFlow model for classifying patterns of clothing items, uses the model for classification, and returns the results to FastAPI;
- FastAPI sends a response to the user or other clients.

The developed software uses a modular principle, which allows for quick modernization of the software tool.

4.3. Structure of the input data set

The study is conducted with one data set, which is divided into three samples: "Training sample", "Validation sample", and "Testing sample"; each sample contains images of a certain class. The structure of the dataset, which is a hierarchy of directories, is shown in Figure 2.

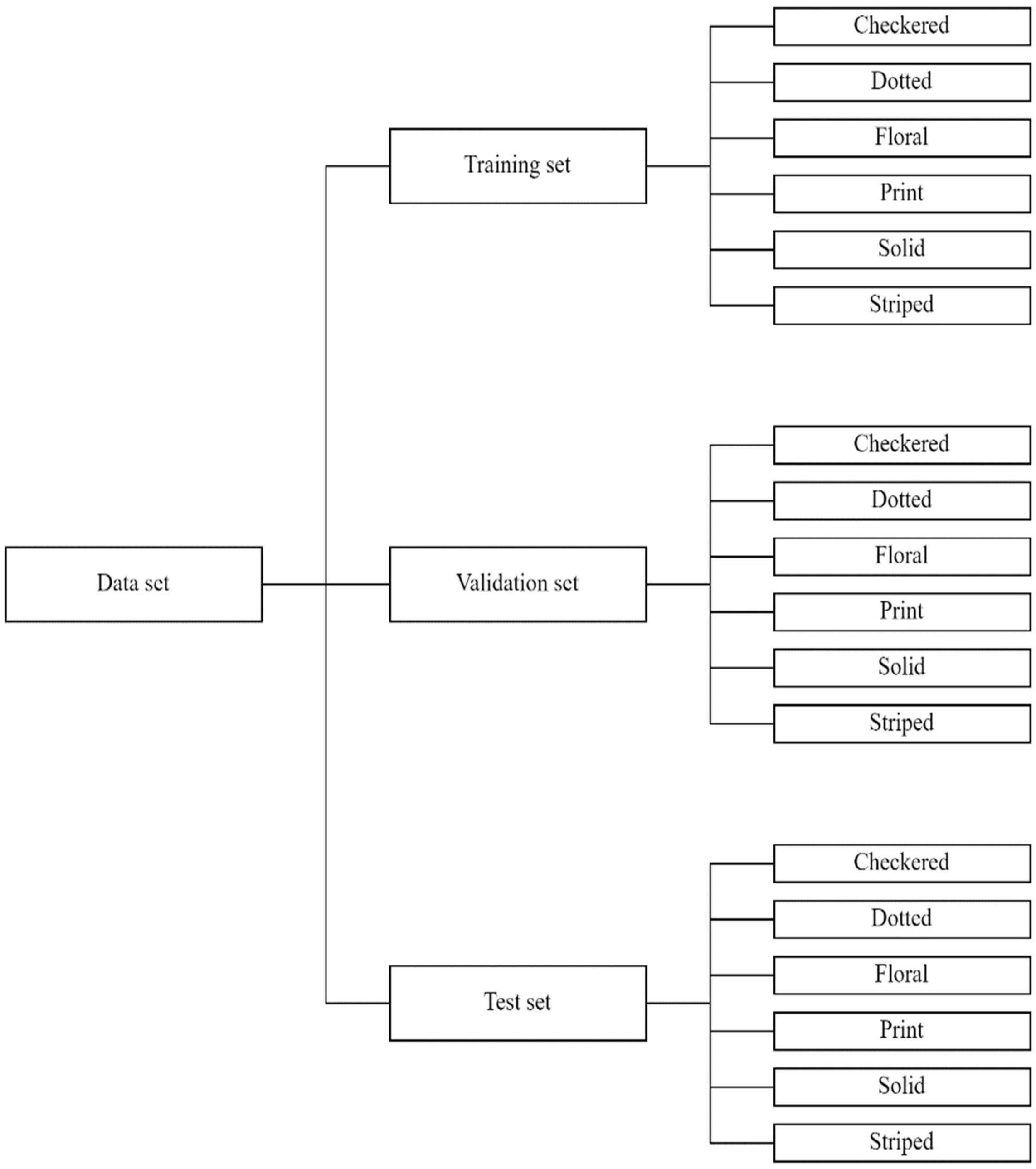


Figure 2: The structure of the dataset.

The classification included the following patterns: “checkered”, “dotted”, “vegetation/floral”, “print”, “solid”, and “striped”. For the task, 100 images were manually chosen for each class, resulting in a total of 600 images. The images are used for training, validation, and testing the model by using generators that select a mini-sample (a “batch”) of images from the corresponding directory during the epoch step, apply augmentation (if specified), and then transmit these images with the corresponding labels to the network. The labels are directories, which denote classes. Generators allow you to work with mini-samples (batches) instead of loading all images into memory.

4.4. Architecture of the applied neural network

Figure 3 shows the architecture of the convolutional neural network.

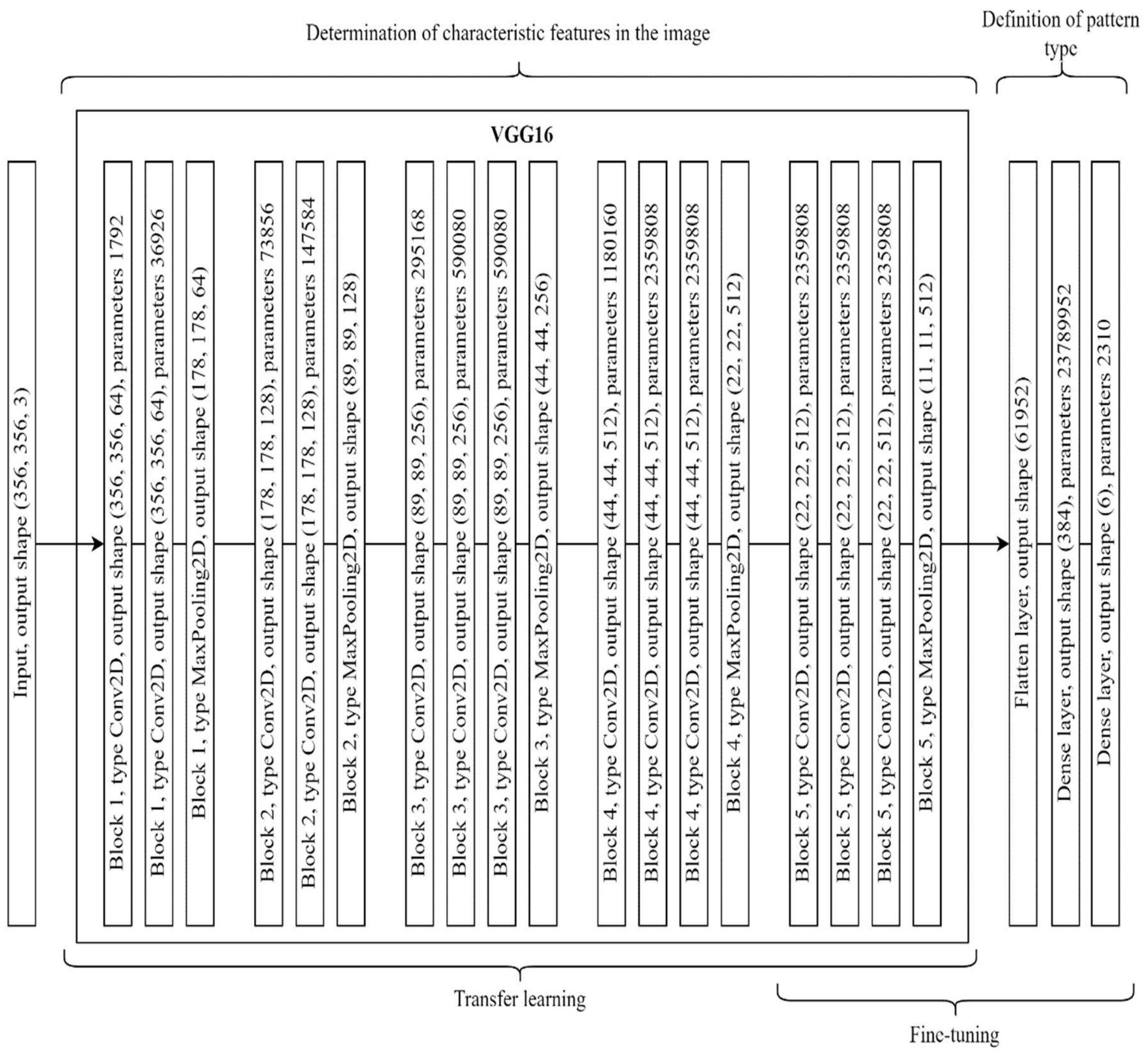

Determination of characteristic features in the image
Definition of pattern type
VGG16
Input, output shape (356, 356, 3)
Block 1, type Conv2D, output shape (356, 356, 64), parameters 1792
Block 1, type Conv2D, output shape (356, 356, 64), parameters 36926
Block 1, type MaxPooling2D, output shape (178, 178, 64)
Block 2, type Conv2D, output shape (178, 178, 128), parameters 73856
Block 2, type Conv2D, output shape (178, 178, 128), parameters 147584
Block 2, type MaxPooling2D, output shape (89, 89, 128)
Block 3, type Conv2D, output shape (89, 89, 256), parameters 295168
Block 3, type Conv2D, output shape (89, 89, 256), parameters 590080
Block 3, type Conv2D, output shape (89, 89, 256), parameters 590080
Block 3, type MaxPooling2D, output shape (44, 44, 256)
Block 4, type Conv2D, output shape (44, 44, 512), parameters 1180160
Block 4, type Conv2D, output shape (44, 44, 512), parameters 2359808
Block 4, type Conv2D, output shape (44, 44, 512), parameters 2359808
Block 4, type MaxPooling2D, output shape (22, 22, 512)
Block 5, type Conv2D, output shape (22, 22, 512), parameters 2359808
Block 5, type Conv2D, output shape (22, 22, 512), parameters 2359808
Block 5, type Conv2D, output shape (22, 22, 512), parameters 2359808
Block 5, type MaxPooling2D, output shape (11, 11, 512)
Flatten layer, output shape (61952)
Dense layer, output shape (384), parameters 23789952
Dense layer, output shape (6), parameters 2310
Transfer learning
Fine-tuning

Figure 3: Network architecture for pattern classification of clothing elements.

Description of the network architecture:

1. The input is an image of 356 by 356 pixels in three channels (RGB).
2. The convolutional part of the VGG16 model:

- block 1: convolutional layer of Conv2D type, filters 64, filter size 3x3, step 1, Same padding, ReLU activation function, output shape (356, 356, 64), number of parameters 1792;
- block 1: convolutional layer of Conv2D type, filters 64, filter size 3x3, step 1, Same padding, ReLU activation function, output shape (356, 356, 64), number of parameters 36926;
- block 1: pooling layer of MaxPooling2D type, filters 64, filter size 2x2, step 2, output shape (178, 178, 64);
- block 2: convolutional layer of Conv2D type, 128 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (178, 178, 128), number of parameters 73856;
- block 2: convolutional layer of Conv2D type, 128 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (356, 356, 128), number of parameters 147584;
- block 2: pooling layer of MaxPooling2D type, 128 filters, 2x2 filter size, step 2, output shape (89, 89, 128);
- block 3: convolutional layer of Conv2D type, 256 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (89, 89, 128), number of parameters 73856;
- block 3: convolutional layer of Conv2D type, 256 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (89, 89, 128), number of parameters 590080;

- block 3: convolutional layer of Conv2D type, 256 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (89, 89, 128), number of parameters 590080;
- block 3: pooling layer of MaxPooling2D type, 256 filters, 2x2 filter size, step 2, output shape (44, 44, 256);
- block 4: convolutional layer of Conv2D type, 512 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (44, 44, 512), number of parameters 1180160;
- block 4: convolutional layer of Conv2D type, 512 filters, filter size 3x3, step 1, Same padding, ReLU activation function, output shape (44, 44, 512), number of parameters 2359808;
- block 4: convolutional layer of Conv2D type, 512 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (44, 44, 512), number of parameters 2359808;
- block 4: pooling layer of MaxPooling2D type, 512 filters, 2x2 filter size, step 2, output shape (22, 22, 512);
- block 5: convolutional layer of Conv2D type, 512 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (22, 22, 512), number of parameters 2359808;
- block 5: convolutional layer of Conv2D type, 512 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (22, 22, 512), number of parameters 2359808;
- block 5: convolutional layer of Conv2D type, 512 filters, 3x3 filter size, step 1, Same padding, ReLU activation function, output shape (22, 22, 512), number of parameters 2359808;
- block 5: pooling layer of MaxPooling2D type, 512 filters, filter size 2x2, step 2, output shape (11, 11, 512).

3. Part for classification:

- flatten layer, output shape (61952);
- fully connected layer, Sigmoid activation function, dropout 0.5, output shape (384);
- fully connected layer, SoftMax activation function, output shape (6).

4.5. Experiments

The dataset is split as follows: 80% for the training set, 10% for the validation set, and 10% for the test set. The batch size is configured to 32, and the input image dimensions are 356 by 356 pixels.

First, hyperparameters are manually selected to determine the model that will be used in further research, with the comparison based on training and testing accuracy.

The VGG16 [29], VGG19 [30], InceptionV3 [31], Xception [32], and ResNet50 [33] models were employed for testing. For each model, pre-trained weights from the ImageNet dataset were applied, the classification layers were removed, and all layers of the network were set to non-trainable. Using TensorFlow, a sequential architecture was built, incorporating the feature extraction layers of VGG16, VGG19, InceptionV3, Xception, and ResNet50. This was followed by a flattening layer, a dense layer containing 256 neurons with a ReLU activation function, and a dropout rate of 0.5. The final output layer consisted of 6 neurons with a SoftMax activation function. The model was configured with 'categorical cross-entropy' as the loss function, and the Adam optimizer with a learning rate of 0.001 was applied over a training duration of 15 epochs.

The results for the specified hyperparameters are presented in Table 1.

Table 1
Accuracy of models with manual selection of hyperparameters

№	The name of the architecture	Training accuracy	Testing accuracy
1	VGG16	0,9646	0,8667
2	VGG19	0,8534	0,6833
3	InceptionV3	0,7933	0,6333
4	Xception	0,8436	0,4667
5	ResNet50	0,3259	0,1833

Further research focused on evaluating the VGG16 model, which achieved the highest performance after manual hyperparameter tuning

At this point, Keras-Tuner [34] was applied with the Bayesian Optimization algorithm to search for optimal parameters. The VGG16 network's weights were loaded from the ImageNet dataset, with its classification layers removed, and all network layers were set as non-trainable. A sequential model was created in TensorFlow, starting with the feature extraction section of the VGG16 network, followed by a flattening layer. The number of dense layers was set to vary between 1 and 2, with each layer containing between 32 and 512 neurons, increasing in steps of 32. Activation functions for these layers included ReLU, Sigmoid, Tanh, ELU, and SELU, and a dropout rate of 0.5 was applied to each dense layer. The final layer was a dense layer with 6 neurons using the SoftMax activation function. The optimization algorithms considered included Adam, RMSprop, and SGD, with metrics such as Accuracy, Precision, and Recall measured.

Keras-Tuner performed 10 rounds of network training, adjusting hyperparameters through Bayesian Optimization as defined in the model-building function. The optimization targeted a reduction in the loss function value on the validation dataset. The three best hyperparameter configurations from the 10 trials are shown in Table 2.

Table 2
Hyperparameter sets obtained using Keras-Tuner

№	Activation function	Number of layers	The number of neurons in a layer	Optimizer	The value of the loss function on the validation set during training
1	Sigmoid	1	384	RMSprop	0,36
2	ELU	2	1 – 512, 2 – 320	SGD	0,39
3	SGD	1	192	Adam	0,42

A model is constructed using the hyperparameters that yielded the lowest loss function value. The weights of the VGG16 network from the ImageNet dataset were loaded, with fully connected classification layers removed and training disabled for all network components. Using TensorFlow, a sequential network was created, starting with the part of the VGG16 network responsible for extracting characteristic features from the images. A flatten layer was added, followed by a fully connected layer containing 384 neurons and a Sigmoid activation function. A dropout rate of 0.5 was applied, along with a fully connected layer of 6 neurons utilizing the SoftMax activation function. The loss function is defined as "categorical cross-entropy", and the RMSprop optimizer is used with a learning rate of 0.001 over 50 epochs. Additionally, the TensorFlow library includes a tool called ModelCheckpoint, which saves the best model based on a specified metric, namely the loss function on the validation set during training.

Table 3 displays the metrics of the model created using Keras-Tuner hyperparameters, which achieved the highest performance.

Table 3
Metrics of the model built by Keras-Tuner hyperparameters

Sample	Loss	Accuracy	Precision	Recall
Training	0,0755	0,9917	0,9916	0,9812
Validation	0,2432	0,9375	0,9355	0,9062
Test	0,3711	0,9333	0,9310	0,9000

Figure 4 illustrates the accuracy and loss function plots for the model developed using Keras-Tuner hyperparameters. It is evident that overfitting starts after the 30th epoch, with the selected model resulting from the 29th epoch.

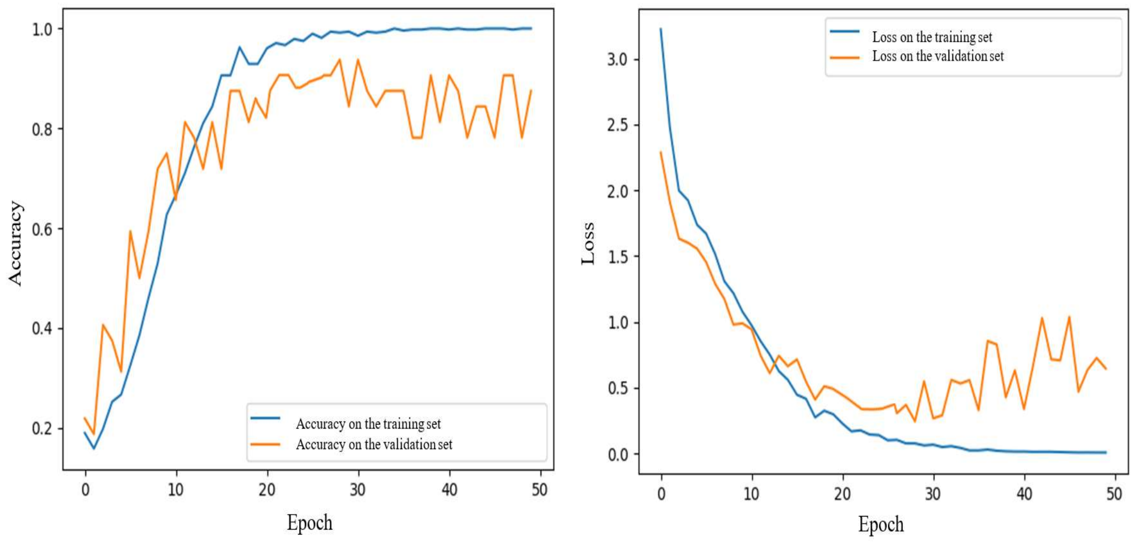


Figure 4: Plots of accuracy and error functions for the Keras-Tuner hyperparameterized model.

In the next stage, fine-tuning is implemented by "unfreezing" the layers of the 5th block of the VGG16 neural network, which are responsible for identifying characteristic features in the image. Unfreezing these layers will enable the model to better highlight the distinct features of the objects present in the applied dataset. Fine-tuning can only be conducted if the classification component of the network has already been trained. If the classification part is not trained, the weights are initially set randomly, leading to significant errors at the beginning of the training process and indicating a high gradient of weight change. According to the error backpropagation algorithm, the signal from the classifier will propagate to the convolutional layers. As a result, in the "unfrozen" convolutional layers, the weights will experience substantial changes, which could degrade the accuracy of the pre-trained network utilized through transfer learning. Therefore, at this stage, work is conducted with the trained model, using hyperparameters derived from Keras-Tuner. The TensorFlow-trained model is loaded into memory. A loop is defined for the layer containing the convolutional part of the VGG16 model to "unfreeze" the layers of the 5th block, specifically three convolutional layers and one pooling layer. The loss function is set to "categorical cross-entropy", and the RMSprop optimizer is employed with a learning rate of 0.00001, over a total of 50 epochs. Additionally, the TensorFlow library includes a tool called ModelCheckpoint, which saves the best model based on the specified metric, specifically the loss function on the validation set during training.

Table 4 presents the metrics of the model with the "unfreezing" of the layers in the 5th block of the VGG16 convolutional part.

Table 4
Metrics of the model with "unfreezing" of the layers of the 5th block of the convolutional part of the VGG16 layer

Sample	Loss	Accuracy	Precision	Recall
Training	0,0404	0,9999	0,9999	0,9999
Validation	0,1000	0,9999	0,9999	0,9688
Test	0,2078	0,9500	0,9655	0,9333

Figure 5 displays the graphs of accuracy and loss functions for the model with the "unfreezing" of the layers in the 5th block of the VGG16 convolutional part. It can be observed that overfitting begins after the 5th epoch, with the selected model derived from the 5th epoch.

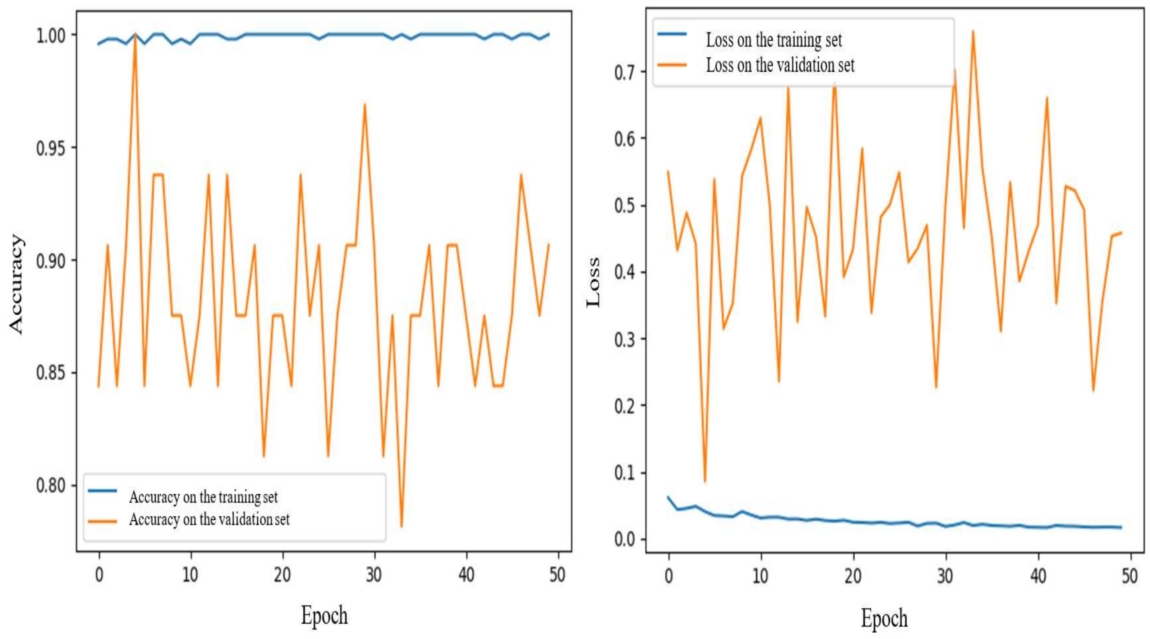
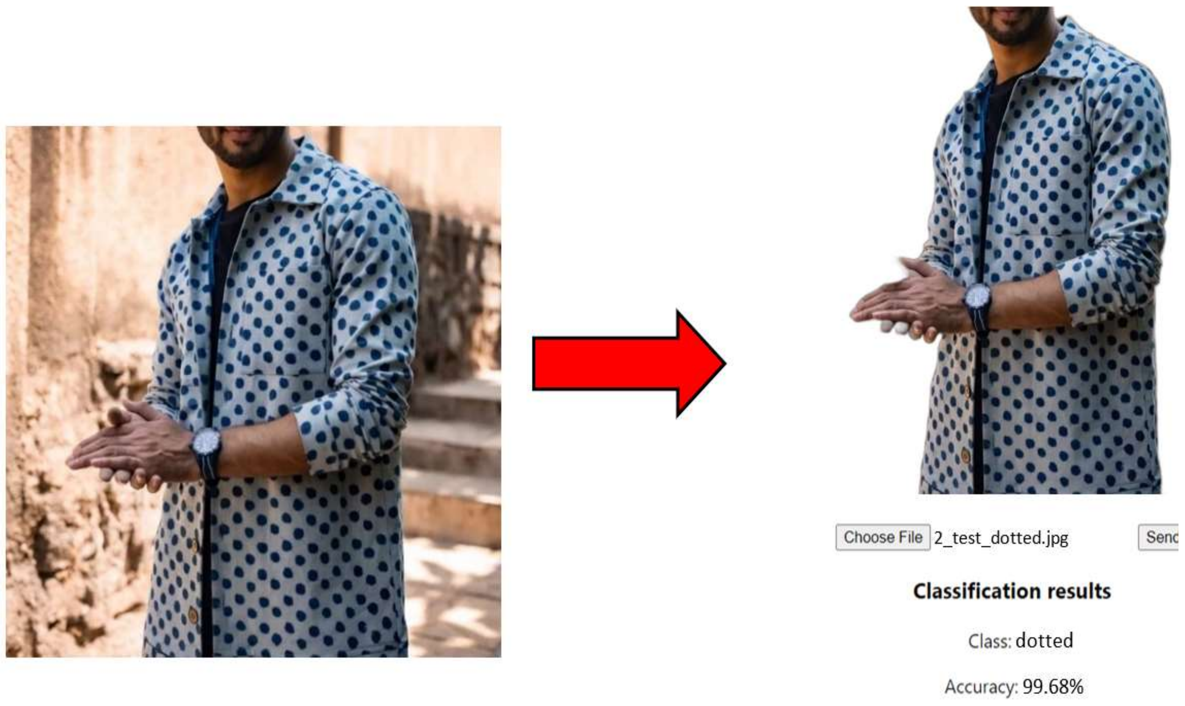


Figure 5: Graphs of accuracy and loss functions for the model with the 5th block of the VGG16 convolutional layers unfrozen.

The trained neural network was evaluated using new images, with examples provided for two patterns. Initially, a test image featuring a dotted pattern was used, resulting in a correct classification with an accuracy of “99.68%”, as illustrated in Figure 6.



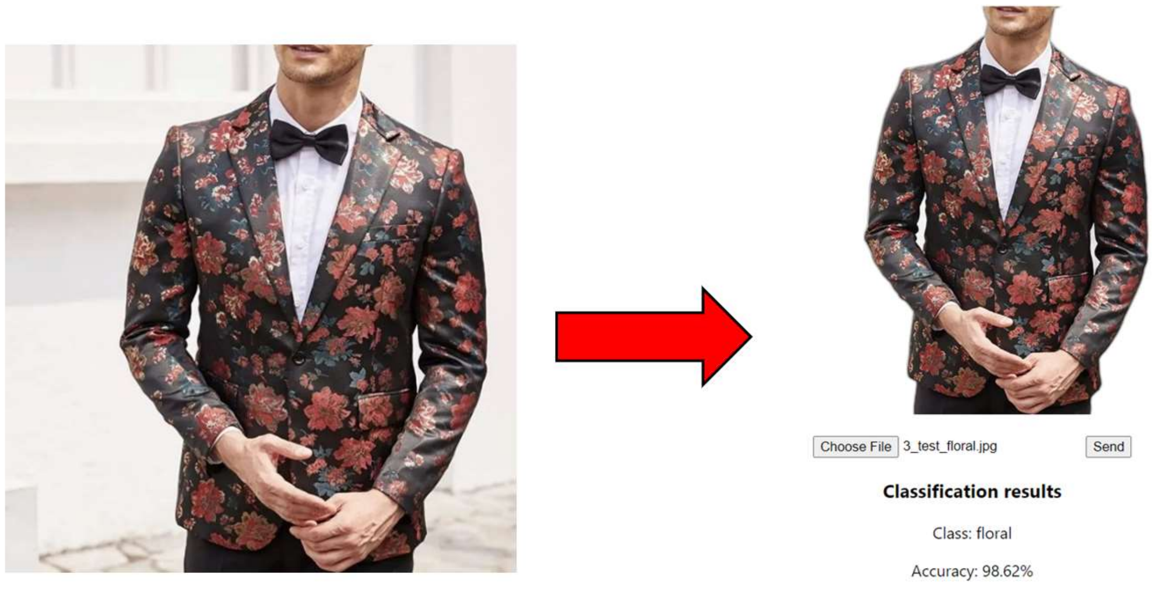
Classification results

Class: dotted

Accuracy: 99.68%

Figure 6: Evaluating the user interface and the trained model by inputting an image with a “dotted” pattern.

Additionally, a test image displaying a “floral” pattern was utilized, achieving a correct classification result with an accuracy of “98.62%”, as demonstrated in Figure 7.



Classification results

Class: floral

Accuracy: 98.62%

Figure 7: Evaluating the user interface and the trained model by inputting an image featuring a "floral" pattern.

5. Discussion of the obtained results

A convolutional neural network was created for classifying clothing patterns using transfer learning, achieving a training accuracy of 99.17%, a validation accuracy of 93.75%, and a testing accuracy of 93.33%. Subsequently, through fine-tuning, the final version of this network was further optimized, reaching a training accuracy of 99.99%, a validation accuracy of 99.99%, and a testing accuracy of 95%.

The research's scientific contribution consists in advancing a model for classifying clothing patterns by leveraging machine learning techniques, specifically through the application of transfer learning and fine-tuning methods.

The practical value of the research lies in the implementation of machine learning for clothing pattern classification, with developed software, defined system architecture, and an established mathematical foundation. These elements are backed by the findings presented in the study.

6. Conclusions

The study explores and implements models and tools aimed at improving the effectiveness of clothing pattern classification through machine learning. The findings allow users to recognize clothing patterns within images accurately.

The study showed that the proposed model achieved greater efficiency in clothing pattern classification.

The project can be improved and expanded in the following ways:

- increase the volume and diversity of the data set, which will increase the robustness of the artificial neural network;
- apply augmentation to increase the data set;
- add other classes of patterns;
- continue to optimize the artificial neural network by selecting hyperparameters and making changes to the architecture; optimize the speed of forecasting;
- organize a system of anonymization of people in images;
- to develop means of exporting the results of classification of patterns of clothing elements.

The results of this study can be used for integration into complex computer vision systems used for processing clothing images.

References

[1] V. Zakaryan, AI Clothing Detection: Use Cases for Fashion and E-commerce, 2022. URL: https://postindustria.com/ai-clothing-detection-use-cases-for-fashion-and-e-commerce/.

[2] H. Wang, Rule-free sewing pattern adjustment with precision and efficiency, ACM Trans. Graph. 37 (2018) 1–13. doi:10.1145/3197517.3201320.

[3] L. Liu, X. Xu, Z. Lin, J. Liang, S. Yan, Towards Garment Sewing Pattern Reconstruction from a Single Image, ACM Trans. Graph. 42 (2023). doi:10.1145/3618319.

[4] Y. Shen, J. Liang, M. Lin, GAN-Based Garment Generation Using Sewing Pattern Images, 2020. doi:10.1007/978-3-030-58523-5_14.

[5] K. Mehta, S. P. Panda, Sentiment Analysis on E-Commerce Apparels using Convolutional Neural Network, Int. J. Comput. 21 (2022) 234–241. doi:10.47839/ijc.21.2.2592.

[6] M. M. A. El-Nahas, The Impact of Augmented Reality on Fashion and Textile Design Education, Int. Des. J. 11 (2021) Article 3.

[7] O. Jadhav, A. Patil, J. Sam, M. Kiruthika, Virtual Dressing using Augmented Reality, ITM Web Conf. 40 (2021) 03028. doi:10.1051/itmconf/20214003028.

[8] M. A. I. Hussain, B. Khan, Z. Wang, S. Ding, Woven Fabric Pattern Recognition and Classification Based on Deep Convolutional Neural Networks, Electronics 9 (2020) 1048. doi:10.3390/electronics9061048.

[9] A. Birjuk, Unseen and unheard: The power of anti-surveillance clothing, 2023. URL: https://medium.com/@alinabirjuk/unseen-and-unheard-the-power-of-anti-surveillance-clothing-156570fefb0e.

[10] L. Sikora, R. Martsyshyn, Y. Miyushkovych, N. Lysa and B. Yakymchuk (2015) "Systems approaches of providing the guaranteed functioning of technological structures on the basis of expert coordination o f local strategies," Xth International Scientific and Technical Conference "Computer Sciences and Information Technologies" (CSIT), Lviv, 2015, pp. 166-168. doi: 10.1109/STC-CSIT.2015.7325458.

[11] K. E. Rajasekhar, Convolutional Neural Network, 2020. URL: https://developersbreach.com/convolution-neural-network-deep-learning/

[12] JetBrains, PyCharm – The Python IDE for Professional Developers, n.d. URL: https://www.jetbrains.com/pycharm/.

[13] TensorFlow, Create production-grade machine learning models with TensorFlow, n.d. URL: https://www.tensorflow.org/.

[14] Keras, Keras – deep learning API, n.d. URL: https://keras.io/.

[15] Nvidia, CUDA Toolkit, n.d. URL: https://developer.nvidia.com/cuda-toolkit.

[16] Nvidia, CUDA Deep Neural Network library, n.d. URL: https://developer.nvidia.com/cudnn.

[17] C. Loewen, M. Wojciakowski, et al., Windows Subsystem for Linux, 2023. URL: https://learn.microsoft.com/en-gb/windows/wsl/install.

[18] S. Ramírez, FastAPI framework, high performance, easy to learn, fast to code, ready for production, n.d. URL: https://fastapi.tiangolo.com/.

[19] D. Gatis, Rembg – a tool to remove images background, 2020. URL: https://github.com/danielgatis/rembg.

[20] J. D. Hunter, Matplotlib: A 2D graphics environment, Comput. Sci. Eng. 9 (2007) 90–95. doi:10.1109/MCSE.2007.55.

[21] V. M. Teslyuk, S. S. Ivasiv, System for recognizing clothing items and their colors in an image, Ukr. J. Inf. Technol. 5 (2023) 25–32. doi:10.23939/ujit2023.02.025.

[22] L. Stearns, L. Findlater, J. E. Froehlich, Applying Transfer Learning to Recognize Clothing Patterns Using a Finger-Mounted Camera, in: Proc. 20th Int. ACM SIGACCESS Conf. Comput. Access., ASSETS '18, ACM, 2018, pp. 349–351. doi:10.1145/3234695.3241015.

[23] E. Dey, M. N. A. Tawhid, M. Shoyaib, An Automated System for Garment Texture Design Class Identification, Comput. 4 (2015) 265–282. doi:10.3390/computers4030265.

[24] H. Chen, A. Gallagher, B. Girod, Describing Clothing by Semantic Attributes, in: Proc. 2012 IEEE Conf. Comput. Vis. Pattern Recognit., IEEE, 2012, pp. 609–623. doi:10.1007/978-3-642-33712-3_44.

[25] M. Manfredi, C. Grana, S. Calderara, et al., A complete system for garment segmentation and color classification, Mach. Vis. Appl. 25 (2014) 955–969. doi:10.1007/s00138-013-0580-3.

[26] S. S. Islam, E. K. Dey, M. N. A. Tawhid, B. M. M. Hossain, A CNN Based Approach for Garments Texture Design Classification, Adv. Technol. Innov. 2 (2017) 119–125. URL: https://ojs.imeti.org/index.php/AITI/article/view/366.

[27] V. Kotsovsky, A. Batyuk and V. Voityshyn, "On the Size of Weights for Bithreshold Neurons and Networks," *2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT)*, LVIV, Ukraine, 2021, pp. 13-16, doi: 10.1109/CSIT52700.2021.9648833.

[28] Kotsovsky, V., Batyuk, A., Mykoriak, I.: The computation power and capacity of bithreshold neurons. In: 2020 IEEE 15th International Conference on Computer Sciences and Information Technologies, CSIT 2020, in press (2020) https://doi.org/10.1109/CSIT49958.2020.9322014 .

[29] Datagen, Understanding VGG16: Concepts, Architecture, and Performance, n.d. URL: https://datagen.tech/guides/computer-vision/vgg16/.

[30] Dr. Info Sec., VGG-19 Convolutional Neural Network, 2021. URL: https://blog.techcraft.org/vgg-19-convolutional-neural-network/.

[31] A. T. Narein, Inception V3 Model Architecture, 2021. URL: https://iq.opengenus.org/inception-v3-model-architecture/.

[32] F. Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, in: Proc. 2017 IEEE Conf. Comput. Vis. Pattern Recognit., IEEE, 2017, pp. 1800–1807. doi:10.1109/CVPR.2017.195.

[33] Datagen, ResNet-50: The Basics and a Quick Tutorial, n.d. URL: https://datagen.tech/guides/computer-vision/resnet-50/.

[34] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, et al., KerasTuner, 2019. URL: https://github.com/keras-team/keras-tuner.