applied sciences

Article

Comparison of Multilayer Neural Network Models in Terms of Success of Classifications Based on EmguCV, ML.NET and Tensorflow.Net

Martin Magdin *, Juraj Benc, Štefan Koprda, Zoltán Balogh and Daniel Tuček

Department of Informatics, Faculty of Natural Science and Informatics, Constantine the Philosopher University in Nitra, 949 74 Nitra-Chrenová, Slovakia; juraj.benc@student.ukf.sk (J.B.); skoprda@ukf.sk (Š.K.); zbalogh@ukf.sk (Z.B.); daniel.tucek@ukf.sk (D.T.)
* Correspondence: mmagdin@ukf.sk

Abstract: In this paper, we compare three different models of multilayer neural networks in terms of their success in the classification phase. These models were designed for EmguCV, ML.NET and Tensorflow.Net libraries, which are currently among the most widely used libraries in the implementation of an automatic recognition system. Using the EmguCV library, we achieved a success rate in the classification of human faces of 81.95% and with ML.NET, which was based on the pre-trained ResNet50 model using convolution layers, up to 91.15% accuracy. The result of the success of the classification process was influenced by the time required for training and also the time required for the classification itself. The Tensorflow.Net model did not show sufficient classification ability when classifying using vector distances; the highest success rate of classification was only 13.31%. Neural networks were trained on a dataset with 1454 photographs of faces involving 43 people. At a time when neural networks are becoming more and more used for applications of different natures, it is necessary to choose the right model in the classification process that will be able to achieve the required accuracy with the minimum time required for training. The application created by us allows the insertion of images and the creation of their own datasets, on the basis of which the user can train a model with its own parameters. Models can then be saved and integrated into other applications.

Keywords: classification; convolution nets; neural nets; detection; face recognition

Citation: Magdin, M.; Benc, J.; Koprda, Š.; Balogh, Z.; Tuček, D. Comparison of Multilayer Neural Network Models in Terms of Success of Classifications Based on EmguCV, ML.NET and Tensorflow.Net. *Appl. Sci.* **2022**, *12*, 3730. https://doi.org/10.3390/app12083730

Academic Editor: Valentino Santucci

Received: 9 March 2022
Accepted: 6 April 2022
Published: 7 April 2022

1. Introduction

Current automatic recognition systems that contain all three phases (detection, extraction and classification) work with different methods. One of the currently most used methods is the application of neural networks [1,2].

A neural network is a series of algorithms that try to recognize certain relationships between data through a process based on how the human brain works. They can also process and adapt foreign input data, learn to classify them and adapt their own process based on them [3].

The advantage of neural networks is that they can be used not only in one phase but in essentially all three phases. However, they are mostly used in the last phase—classification—when the amount of input data based on a trained set (dataset) can very accurately determine (classify) what the output will be [4–7].

There are several tools that make working with neural networks easier. They provide certain predefined algorithms that allow users to build their own neural network relatively quickly and without deep mathematical knowledge [8–10]. In this paper, we focus on comparing the models created using the most used libraries: EmguCV (a subset of OpenCV), ML.NET and Tensorflow.NET. The Related Work section is focused on an overview of research on the results achieved, depending on the design to the application of neural networks and selected algorithms that multilayer neural networks use in the classification and re-learning phase. The Experiment Preparation section describes the technologies that were used to create the individual models of neural networks. The procedures of data extraction and pre-preparation are described here, together with the creation, training and testing of individual created models of neural networks. In the Discussion section and the evaluation of results from experiments, we present the achieved results and the overall comparison of models.

2. Related Work

The first significant step for artificial intelligence in the use of neural networks occurred almost 80 years ago when the very first neural network model was created using electrical circuits (formal neurons consisting of basic logic operations). However, this model was not able to learn [11]. Nevertheless, it turned out later that even such a neural network model could be trained to be able to recognize and classify objects itself [12,13]. This type of network was called a "perceptron". It is able to work with the so-called number of weighted inputs of any size, not only binary, and can adjust its weights, which allows more accurate classification of results. The problem with these early designs of perceptrons was the fact that they could not be trained to recognize many classes of patterns. This caused stagnation in neural network research.

Multilayer perceptrons, commonly called artificial neural networks in machine learning, are parallel processes with the ability to store different knowledge and learning. By connecting artificial neurons, a neural network is created, and by connecting several neural networks, a so-called multilayer neural network is created. An artificial neuron mimics the process of a biological neuron's function by a mathematical function that receives evaluated data as input, transforms this signal and sends it on to the neurons with which it is associated [14].

Perceptron is the simplest form of single-layered network with forward propagation, which consists of four parts:

- Input values or input layer;
- Weights and optional neuron bias;
- Weighed sum;
- Activation function [15].

The multilayer perceptron is thus an extension of the forward neural network. After each evaluation of the result, the network adjusts the weights, which results in so-called network learning [16].

One of the most widely used algorithms for weight adjustment in forward neural networks is the backpropagation algorithm. The backpropagation algorithm is also considered the simplest and most common in training learning networks assisted by a teacher. This algorithm is based on the principle of approximation of a nonlinear relationship by adjusting the values of internal weights. Backpropagation has two phases, training and testing [17]. With the learning algorithm assisted by the teacher, the neural network (during training) also has access to the correct results, on the basis of which it can retrofit its weights to achieve the desired value in the result and thus train its model, which will be able to predict correct results unknown data. This type of learning is used for classification and regression. Without a teacher, learning algorithm models must look for patterns in the data themselves and group the data into unlabeled data with minimal human intervention. Such a model has only input data available without the required output. It is up to the model to find connections in the input data [18].

A number of proven methods already exist at present for the classification phase [19–23]. However, the two basic methods are used the most, which are based on face recognition using extracted areas of interest and face recognition based on the classification of the whole input image. The idea of classifying users only on the basis of areas of interest (from the features of the face) lies in the speed of the model. The positional coordinates of the areas of interest represent the inputs for the neural network model, which may be able to recognize a specific person based on their values. Normally, there are several points that represent the face (position of the eyes, mouth, eyebrows, nose and other items that as a whole represent the person's face). Khashman [24] described the process of creating a multilayer forward neural network learning with teacher assistance with the purpose of classifying the human face. He modeled this neural network from three layers, an input layer containing 100 neurons, a hidden layer containing 45 neurons and an output layer consisting of 30 neurons. Sigmoid was selected as the activating function of the hidden and output layer. Images of people were used as input data to this neural network. The data on which the neural network was trained consisted of 270 images and 30 people of different ages, ethnicity and gender. The results of this model were measured by classification accuracy on test data, resulting in 83.33% success [25].

For the individual phases of the recognition process (and thus also the classifications), the so-called convolutional neural networks are used [26]. These are used, for example, for object detection, detection and classification of emotions from a person's voice or face, etc.

Several main types of such networks are known, among the best and most important ones are:

- LeNet;
- AlexNet;
- ZfNet;
- VGG;
- GoogleNet [27].

LeNet is the first convolutional neural network ever created. It was created by LeCun, and the most recent version of this network is currently the LeNet-5. It is used to recognize handwritten or typewritten numbers. It consists of five convolutional and pooling layers that alternate, followed by two fully connected layers [28].

AlexNet is a convolutional neural network for object detection. This net consists of eight layers with weights. The first five are convolutional, and the rest are fully connected layers. The network maximizes multinomial logistic regression [29].

ZfNet is used to detect objects, similar to AlexNet. It is a combination of the first two mentioned networks with modifications. Its principle is the use of standard fully controlled convolution models. These networks map a color 2D image. Each layer consists of a convolution of the output of the previous layer. For the first layer, a set of learned filters is used [30].

VGG is a convolutional network used to detect objects. It is based on AlexNet with a different configuration. This network is used on a much larger pool of samples. The network configuration is not fixed, although each of these networks consists of several convolution layers, followed by three fully connected layers. The last layer is the so-called soft-max layer [31].

GoogleNet is another object detection network. It is currently one of the most successful networks, based on the LeNet network. It uses convolutional blocks for classification. It uses average pooling instead of completely connected layers. It consists of a total of 22 layers [32].

Ref. [33] described the development of a convolutional neural network for facial classification. The formed neural network had an input, two convolutional, a coupling and a dense output layer. The model was trained on the "Yale Face Database" containing 165 single-channel images of 15 people. Each person was represented by 11 images. The input images were chosen with a size of 64 × 64 pixels. The first convolution layer contained six characteristic tables with a dimension of 58 × 58, and the individual values in the characteristic tables were calculated from the partial data of the input image with a size of 7 × 7 pixels. The second merging layer contained six characteristic tables with a dimension of 29 × 29, and the partial data were selected with a size of 2 × 2 pixels. The third convolution layer contained 16 characteristic tables with a size of 22 × 22 pixels, the values of which were calculated from partial data with a size of 8 × 8. The fourth merging layer contained 16 characteristic tables of size 11 × 11 with partial data 2 × 2. The output layer is dense and classified into 15 different people. Their resulting successful classification value was 88.88% on training data and 80% on test data. Based on this experiment, it can be stated that both the application of the forward and convolutional networks provide a classification success of over 80%, which is normally considered an effective use of the classification algorithm.

An example of the basic architecture of a convolutional network is shown in Figure 1.

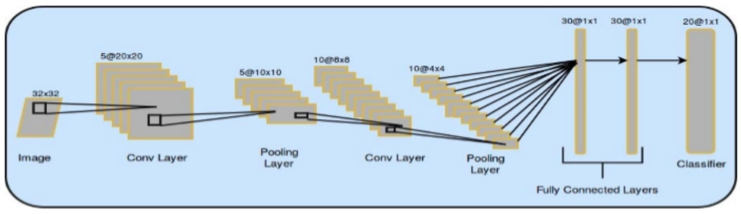


Figure 1. Basic architecture of a convolutional network [34].

The fact that the use of convolutional networks is really very wide is also demonstrated by an experiment carried out by Severyn and Moschitti, who focused on the analysis of tweet sentiment from Twitter. They used stochastic gradient sinking (SGD) and then backpropagation algorithms to train the network. They also chose the Adadelt rule, which automatically tunes and updates the learning rate. They also had to be careful not to overfit the network, which occurs with small and medium-sized datasets. To prevent overfitting, they have extended the cost function with standardization conditions. They also used a technique called dropout, which prevents the network from adapting to the data by setting the values of some parts of the hidden units to zero during the forward phase of training. They followed three main steps in initializing the weights: they provided high-quality inputs to the network, they needed a suitable in-depth learning system when handling a small amount of marked data and they used the parameters obtained in step 2 to initialize the network. They used the Twitter‘13 dataset to validate testing [35].

The biggest known problem with convolutional neural networks is the area of training. In particular, when training a network, a large number of parameters need to be tuned to make the network architecture suitable for use. There are many problems and shortcomings of neural networks, but these are mostly solved in the works themselves [34]. For parameter tuning, it is possible to use, e.g., Bayesian optimization [36]. One of the basic problems is the so-called conflicting picture where the CNN classification is completely different. The problem with such networks is that the network gives the wrong answer with high confidence. This means that even though the network is highly reliable, such an image can cause a malfunction in its classification. This phenomenon is due to the linear nature of neural networks, which means that different networks with different architectures and training on different datasets classify the same example as conflicting. To reduce the malfunction, it is possible to use adversarial network training. However, this method does not solve the problem in the long run [37].

3. Material and Methods

The aim of the experiment was to create and design models of multilayer neural networks in EmguCV, ML.NET, Tensorflow.NET, comparing the results from the models in relation to their ability to successfully classify people's faces (over 80%). The development of the application for testing models was carried out on a computer system with the Windows10 operating system, a 6-core processor with a maximum core frequency of

4.2 GHz and an allocated RAM memory of 4GB. The main logic of the application was programmed in C # (via Visual Studio using the Windows Form library), and the code was compiled by the Roslyn compiler.

EmguCV is a multiplatform image processing library. It is an extension of the OpenCV library [38]. In EmguCV, images are represented in Mat or Matrix formats. They represent data about individual pixels in the image, which are stored in their columns and rows. Unlike a typical data matrix, they contain other essential information needed to work with an image, such as what color channel the image is encoded in. The Mat data type cannot be managed. Data of this type can only be read, loaded or cloned, but not edited. The Matrix is used for managed work with data. This data type is used by the ANN_MLP method, which requires data encoded in this format when creating, training, and testing a neural network. ANN_MLP is an abbreviation for artificial neural networks of multilayer perceptrons and is the main tool for creating multilayer neural networks in EmguCV.

The robust Viola–Jones algorithm can be used for the recognition process phases in EmguCV as well as in OpenCV [39,40]. The Viola–Jones algorithm itself consists of four parts [41–43]:

- Selection of haar attributes;
- Creation of an integral image;
- Adaboost;
- Cascade classificators.

Tensorflow is a software library with publicly available source code developed by Google to facilitate the creation, modeling and training of machine learning models, neural networks and other statistical and predictive analytical tasks. Tensorflow creates graphs of data flows that represent the structure of data on how data are to be processed in a model or series of process nodes. Each node in the graph represents a certain mathematical function, and each connection between the nodes represents a multidimensional array of data represented by the Tensor data type [44].

ML.NET is a free and open source machine learning software library from Microsoft. It allows you to create, train and export machine learning models. In order for a model to be able to classify unknown data, it must be trained on a sufficiently large amount of data. In the experiment, we decided to use the publicly available "Extended Yale Dataset B" [45] dataset (Figure 2). It contains 39 subjects and more than 1200 front-face images shot under various lighting conditions.

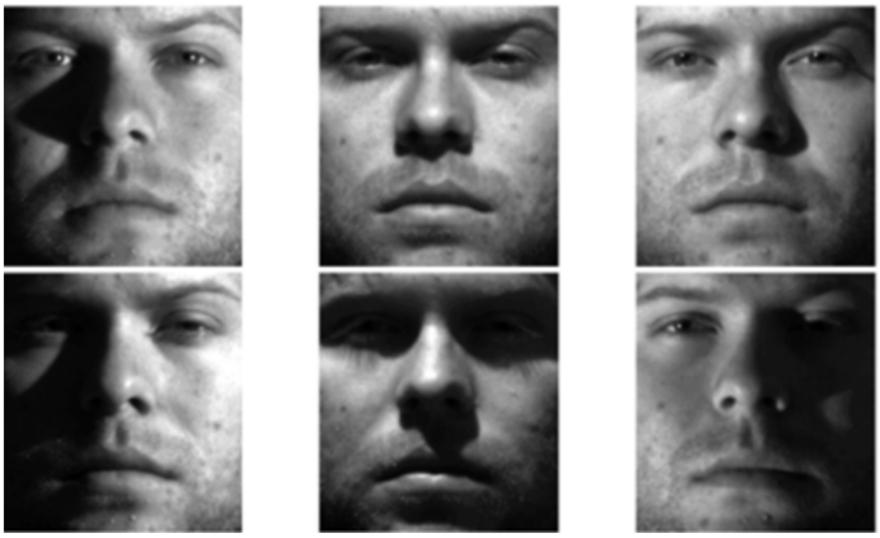


Figure 2. Preview of the Yale Face Dataset B Cropped [45].

As this is a public database, we have provided a test sample of respondents in this way without the need to ask them for permission to work with sensitive data in any way.

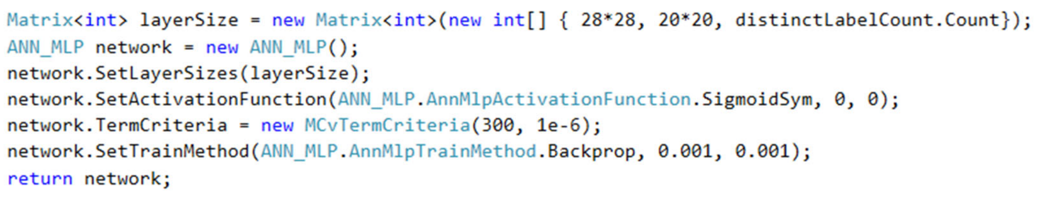
Application creation procedure:

1. Design of a multilayer neural network model in EmguCV focused on human face recognition based on the Viola–Jones algorithm and pixel identification of input images;
2. Design of a model for the Tensorflow.NET framework focused on face recognition based on extracted facial features;
3. Design of a model for the Microsoft framework for machine learning ML.NET for face recognition based on convolutional networks;
4. Testing of created models on individual images of human faces and testing of classification abilities of models.

4. Multilayer Neural Network in EmguCV

In addition to providing imaging tools, EmguCV also enables neural networks, which are often associated with image extraction. Because multilayer perceptrons are a form of the deep neural network, we must first design the layout and size of the layers that the multilayer neural network will contain. It should be noted that the size of the image that this neural network will process will directly affect the training speed and dynamics of the model. If the network were to process images with a resolution of 800 × 600, the input neurons would also be 800 × 600, which, given the large amount of data needed to train the neural network, would be very computationally intensive for many computer systems (this can be solved by PCA—Principal Component Analysis). In our neural network design, we chose the size of the input data of 28 × 28 pixels, i.e., 784 input neurons. The hidden layer contains a smaller number of neurons to reduce the size of the input data entering the output layer but at the same time a sufficient number of neurons to be able to properly represent the data patterns of individual faces. The number of neurons in the hidden layer was chosen to be 400. The output layer does not have a directly determined number of output neurons but is defined during the application during model training. The number of neurons in the output layer is equal to the number determining the number of classification classes, in our case, individual persons.

The content of the method that defines the model of the formed neural network can be described by the following code (Figure 3):



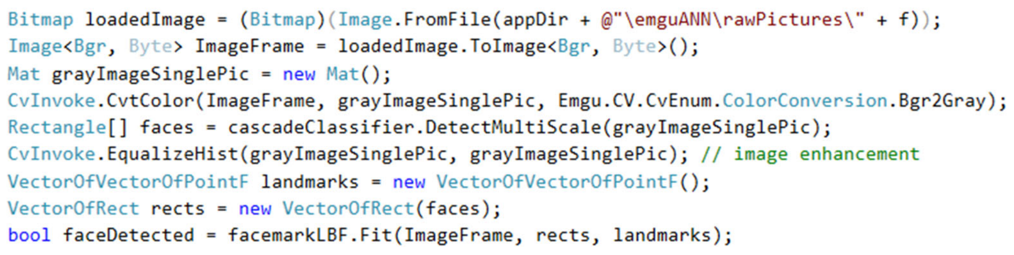
```
Matrix<int> layerSize = new Matrix<int>(new int[] { 28*28, 20*20, distinctLabelCount.Count});
ANN_MLP network = new ANN_MLP();
network.SetLayerSizes(layerSize);
network.SetActivationFunction(ANN_MLP.AnnMlpActivationFunction.SigmoidSym, 0, 0);
network.TermCriteria = new MCvTermCriteria(300, 1e-6);
network.SetTrainMethod(ANN_MLP.AnnMlpTrainMethod.Backprop, 0.001, 0.001);
return network;
```

Figure 3. Code example: CreateNetwork method (own creation).

LayerSize defines a numeric array of values representing the number and size of layers in a multilayer neural network, with the first element representing the input layer, the last output layer, and the values hidden between them. ANN_MLP is a class provided by the EmguCV library, which is defined to instantiate it in program memory. After defining the network, we can start entering parameters into it. The first parameter is the definition of the layers, which we mentioned in the first step of the method. Then, the activation function is selected. In our case, we opted for the SigmoidSym activation function. TermCriteria is a method based on which the training process of the model ends. The first input argument represents the number of iterations after which the training process is to end. The second argument is a value that terminates the process if the network accuracy value changes in the next iteration by a smaller number than that value. The last method, SetTrainMethod, determines the principle on which the neural network will adjust the values of its weights. ANN_MPL offers three learning options, namely the aneal method, resistant back promotion or back promotion. We have chosen the already mentioned method of back promotion in our model. The other two parameters determine the speed at which the neural network adjusts its weights. The higher the value of these parameters, the more the values of the weights are affected when adjusting them. After defining all the elements that the neural network should contain, it uses the "return" method to transmit the resulting model, which is ready for training.

4.1. Data Preparation

We created a simple user interface in the application in order to facilitate work with the neural network model by convenient processing of training and test data and to display classified results. Figure 4 shows a preview of the "Emgu Model" card. The first step is to pre-process the data from the dataset—change the resolution size for each image to the desired shape with a size of 28 × 28 pixels (by pressing the "Crop images" button). The next step is to convert the image to the desired color scale, which is provided by the following part of the code:



```
Bitmap loadedImage = (Bitmap)(Image.FromFile(appDir + @"\emguANN\rawPictures\" + f));
Image<Bgr, Byte> ImageFrame = loadedImage.ToImage<Bgr, Byte>();
Mat grayImageSinglePic = new Mat();
CvInvoke.CvtColor(ImageFrame, grayImageSinglePic, Emgu.CV.CvEnum.ColorConversion.Bgr2Gray);
Rectangle[] faces = cascadeClassifier.DetectMultiScale(grayImageSinglePic);
CvInvoke.EqualizeHist(grayImageSinglePic, grayImageSinglePic); // image enhancement
VectorOfVectorOfPointF landmarks = new VectorOfVectorOfPointF();
VectorOfRect rects = new VectorOfRect(faces);
bool faceDetected = facemarkLBF.Fit(ImageFrame, rects, landmarks);
```

Figure 4. Viola–Jones facial extraction and landmark algorithm (own creation).

In the application on the "Emgu Model" tab, pressing the "Populate dataset" button activates a method that individually goes through all the images of cropped faces, extracts individual pixels and divides them by the number 255, which also normalizes them, and then stores these data in memory (Figure 5). Since the individual pixels contain a value from 0 to 255, dividing these values by 255 gives a value from 0 to 1, which reduces the variance of the input values. A large variance of the input values could have an unwanted effect on adjusting the neural network weights. At the end of this cycle, the stored value field is divided into two parts. Ten percent of the data will be randomly selected to represent the test part of the "test.csv" model, and the remaining 90% will be used to train the "train.csv" model. One line in the csv file represents one face. In addition to the extracted pixel data, it also contains the name of the person whose face belongs to it and the identification number required by the EmguCV neural network.

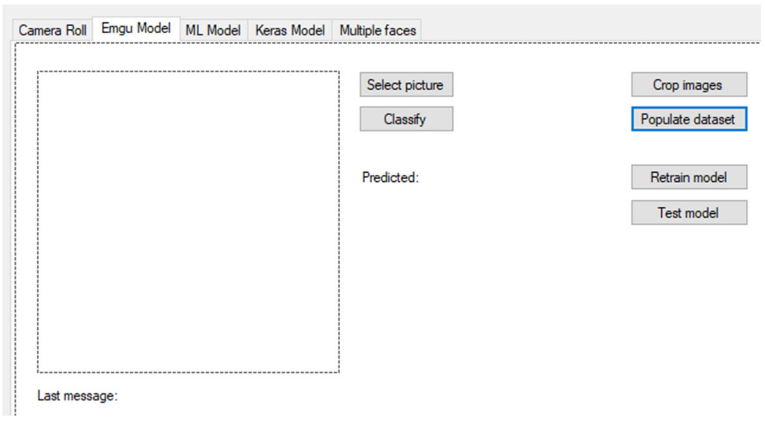


Figure 5. Example of the Emgu Model user interface (own creation).

4.2. Model Training

Model training is the most important part of creating a model. The success of the model classification directly depends on how the neural network adjusts its internal weights during the training process. For this reason, it is appropriate to repeat this step several times or to identify inaccuracies in the classification of the model and extend the dataset with additional data samples of the incorrectly classified class. By selecting images from the YaleB dataset, we created a dataset with a size of 1300 images and 39 people. Pressing the "Retrain Model" button calls the method starting the training phase. Data are read from a training csv file, where one line represents one datum, and each line is divided into three parts. The first part of the line is the ID, which represents the identification number of the person. The second part of the line is the person's name, and the third part contains the values of 784 normalized pixels. The training time depends on the number of data and the neural network settings. One of the disadvantages of EmguCV ANN_MLP is that it does not allow access to its information during training, so it is not possible to monitor the progress or accuracy of the network during the training phase. After training the network, clicking on the "Test model" button starts the test function, which then starts the test phase of the model. Similarly to the training function, the data are read from the test csv file, divided into three parts and cycled after each test sample. After the prediction is generated, this value is then compared with the ID number of the person under whom the test data are requested, the classification is displayed to the user in the application, and the number of correct predictions is increased by one. This process is repeated until all test samples have been tested, and finally, the percentage success rate is calculated depending on the successful classifications from the total number of test samples.

We trained the model 10 times on a dataset containing 43 people and 1545 images. The average success rate of the model classification from the observed data was 81.95%, with an average time required to train the model at 117.1 s. The model was tested on a data sample with a size of 10% of the total dataset.

5. Multilayer Neural Network in Tensorflow.Net

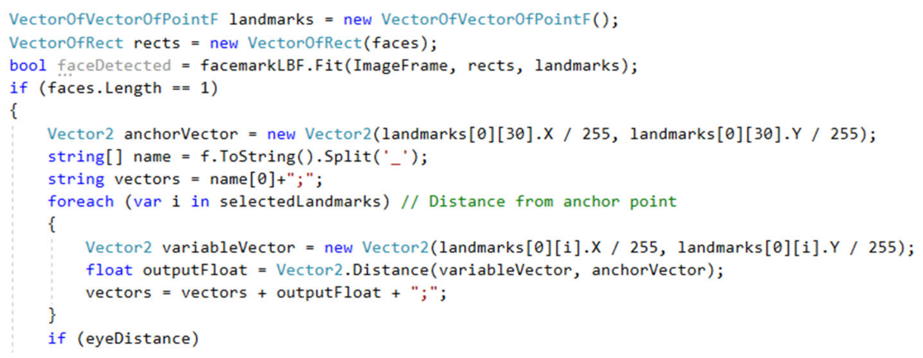
We decided to create a model in the Tensorflow framework due to the fact that it is currently one of the most used tools for the development of neural networks. Its strength is what provides the possibility of more detailed modification of neural network parameters compared to other tools that allow the creation of neural networks.

The main goal in the design of the model was the ability to process a small number of data at the input, compared to other methods for face recognition, and thus speed up the training and testing process of the neural network.

5.1. Data Preparation, Facial Feature Extraction

Similar to the data preparation in EmguCV, we chose the Viola–Jones algorithm in combination with the facial feature extraction algorithm for face extraction. The resolution of the images from which the training data are extracted must be in the same resolution dimension because the model uses the distance between the points of interest on the face. In the application on the "Keras Model" tab, it is possible to call all the functions needed to train and test the model. We pre-process the data by activating the "Resize images" button, which calls the function to convert all images from the dataset to single-channel color depth. This function detects faces and uses the extracted features to trim the face, resize the face to the desired size, and save it to the "Faces" folder located in the Tensorflow model file. We have set the image resolution to 255 × 255 pixels (for easier normalization of pixel values in the single-channel color interface and for eliminating inaccuracies in the detection of facial features in very low-resolution images).

The "Extract landmarks" button activates a function that redetects facial features on individual images of cropped faces, and the data on the detected features are stored in the "VectorOfVectorOfPointF" data type (Figure 6). Each face contains 68 points, which describe its external and internal shapes. It is then verified whether a face has been detected in the cropped face image. This avoids possible inaccuracies that may occur when trimming the face in the previous step or in a situation where the face is too distorted when changing the resolution.



```
VectorOfVectorOfPointF landmarks = new VectorOfVectorOfPointF();
VectorOfRect rects = new VectorOfRect(faces);
bool faceDetected = facemarkLBF.Fit(ImageFrame, rects, landmarks);
if (faces.Length == 1)
{
    Vector2 anchorVector = new Vector2(landmarks[0][30].X / 255, landmarks[0][30].Y / 255);
    string[] name = f.ToString().Split('_');
    string vectors = name[0]+";";
    foreach (var i in selectedLandmarks) // Distance from anchor point
    {
        Vector2 variableVector = new Vector2(landmarks[0][i].X / 255, landmarks[0][i].Y / 255);
        float outputFloat = Vector2.Distance(variableVector, anchorVector);
        vectors = vectors + outputFloat + ";";
    }
    if (eyeDistance)
```

Figure 6. Sample of the function for extracting facial features (own creation).

When calculating the vector distances, the 31st point representing the tip of the nose was chosen as the anchor point (from which the vector distances of the other facial features are calculated (Figure 7). The default face landmarks are then selected in the cycle, and their X and Y coordinates are divided by 255 so that their value is in the range from 0 to 1. A vector is created from them, and the distance between the calculated and anchoring vector is calculated using the "Distance" method.

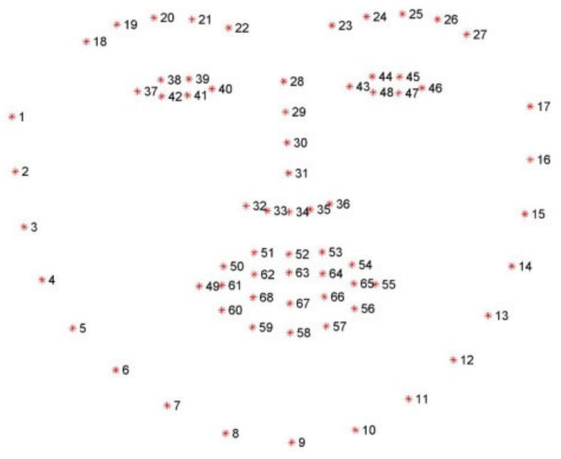
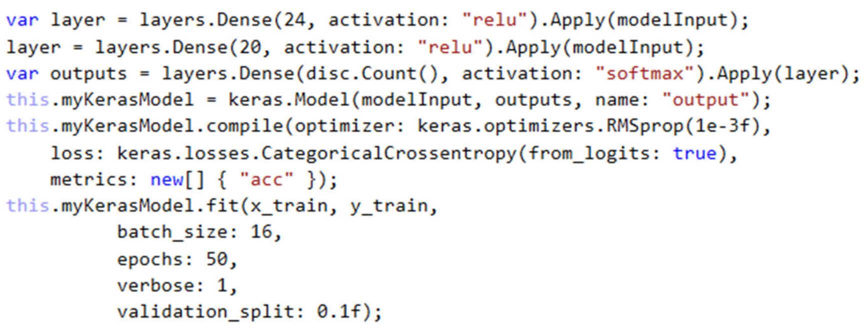


Figure 7. Sketch points of a face (landmarks) (own creation).

5.2. *Tensorflow Model Training*

To work with the Tensorflow framework in our application, we must first import a NuGet package called Tensorflow.NET. It is a wrapper that allows calling Python methods in C #. Although this wrapper does not support all of the methods Tensorflow offers, it is sufficient to create a multilayer neural network. Tensorflow works mainly with the Tensor data type. It is a data type of vectors or matrices of different dimensions and is used to represent all data. Its shape is determined by its data. First, we need to define the model. By using the keras.Input method, we can define the size of the input layer. As a first argument, the method takes a value that specifies the number of neurons. The second argument specifies the name of the layer. Based on it, it will be possible to access it later. We set the number of input neurons to 24, as we decided to represent the individual faces with 24 values of vector distances. In the next step, we will create a variable containing information about the first hidden layer. By calling the layers.Dense method, we create a densely connected, so-called “dense” layer, where each neuron of a given layer contains a connection with each neuron of the previous layer. The first argument of this method specifies the number of neurons that the layer should contain, while the second argument specifies the activation function. We decided to choose the relay activation method based on its speed and the nature of our input data. In the end, the created input layer is applied to this layer, and thus, the information about the layers created so far is stored in one variable. Then, another hidden layer is created. As with the first hidden layer, we chose the dense architecture of the second hidden layer. We set the number of neurons of the second hidden layer to 20 and the activation function on the relay. In the end, all that remains is to define the output layer. By creating the variable outputs, we define the output layer, in which we again selected the dense type of layer. The number of neurons in this layer will determine the number of classification classes. We have selected softmax as the activation function. The softmax activation function converts all classification values into a probability scale and classifies the output class based on the largest value (Figure 8).



```
var layer = layers.Dense(24, activation: "relu").Apply(modelInput);
layer = layers.Dense(20, activation: "relu").Apply(modelInput);
var outputs = layers.Dense(disc.Count(), activation: "softmax").Apply(layer);
this.myKerasModel = keras.Model(modelInput, outputs, name: "output");
this.myKerasModel.compile(optimizer: keras.optimizers.RMSprop(1e-3f),
    loss: keras.losses.CategoricalCrossentropy(from_logits: true),
    metrics: new[] { "acc" });
this.myKerasModel.fit(x_train, y_train,
          batch_size: 16,
          epochs: 50,
          verbose: 1,
          validation_split: 0.1f);
```

Figure 8. Code sample of a Tensorflow model creation (own creation).

After defining the layers, we will define the “myKerasModel” model, based on which we will then classify human faces. The first argument of this layer requires a defined input layer, the second argument a hidden and output layer, and a third neural network name. By calling the compile method on the created Tensorflow model, we specify the parameters of the model in more detail. The first input parameter is specified by the model optimizer. Tensorflow.NET offers three options, Adam, SGD and RMSprop Optimizer. We set the learning speed of the RMSprop optimizer to “1e-3” or “10^{-3}”, which is one-millionth. Another parameter of the compile method is the loss function, in which we chose the categorical cross-entropy function. This loss function is commonly used for classifications with multiple output classes. The third parameter determines how the success of the model classification should be displayed. After defining all the components, the neural network model is ready for the training phase. Calling the fit function on the created model is used to activate the neural network training process. The input parameters of the fit function require data in the form of Tensor, where “x_train” represents data of individual vector distances of the face features in float, and “y_train” represents numeric ID data, based on which the neural network verifies the correct classification during training. Epochs determine how many times to reiterate after input data. The verbose value represents how detailed the training report should be, which is displayed on the console. The last parameter determines the division of data into their validation part.

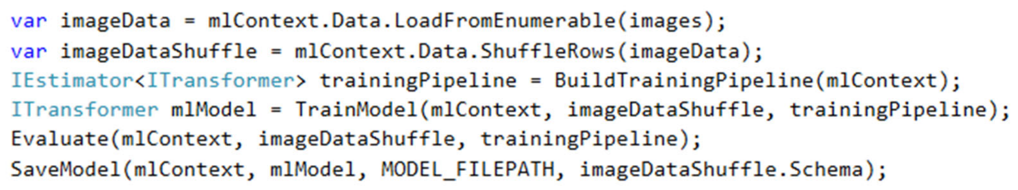
The maximum achieved value of the Tensorflow model prediction was only 15.12%, which makes this model unusable in a real deployment. This small prediction value is caused by inaccuracies in the extraction of facial features from the YaleB dataset and insufficient numbers of elements representing individual faces.

The model was trained 10 times on a dataset containing 43 people and 1545 images. The average success rate of the model classification from the observed data reached the value of 13.31%, with the average time required to train the model at 14.3 s. The model was tested on a data sample with a size of 10% of the total dataset.

6. Multilayer Neural Network in ML.Net

Working with the ML.NET library is directly accessible for all applications created in the Windows Forms framework. We used convolution layers in this model, which transform the input image into certain characteristic tables. Based on them, it is possible to classify which output class it is. The neural network model needs to be taught how it will receive input data, which we implement by creating a new class, "InputImage", with appropriately selected attributes. Defining this class allows you to instantiate each image and train the network. The "OutputImage" class contains the text value of the person's name that the neural network has classified and an array of float values representing the percentage of the model prediction consideration.

By calling the "Data.LoadFromEnumerable" function, we go through all the input images of the "InputImage" type and load them into one variable of the "IDataView" type (Figure 9). It is also the required type for model training. Using the "ShuffleRows" method, we randomly shuffle the input data so that they are not sorted in the order in which they were read.



```
var imageData = mlContext.Data.LoadFromEnumerable(images);
var imageDataShuffle = mlContext.Data.ShuffleRows(imageData);
IEstimator<ITransformer> trainingPipeline = BuildTrainingPipeline(mlContext);
ITransformer mlModel = TrainModel(mlContext, imageDataShuffle, trainingPipeline);
Evaluate(mlContext, imageDataShuffle, trainingPipeline);
SaveModel(mlContext, mlModel, MODEL_FILEPATH, imageDataShuffle.Schema);
```

Figure 9. Sample of the CreateMode method code.

6.1. ML.NET Model Training

By calling the "BuildTrainingPipeline" function, we specify the processing of input data and the form of classification that the model will perform. In the first step of this method, we determine how the neural network model should classify people. In the case of this model, we represent the subjects with their names or with values converted to numeric value IDs. In the multiclassification type of image classification, the epoch value is set to 200, the learning speed to 0.001 and the batch size to 10. The concatenated instructions defined in this way allow the model to be used for training. By calling the "TrainModel" function, the model starts training on the supplied input data. Based on the information specified when creating concatenated instructions, the model begins to test the input data on multiple neural network models that it has specified in advance. The advantage is that the training process is carried out on several pre-prepared models of neural networks, and the one that demonstrates the highest accuracy of correct classifications is chosen. After selecting the most efficient neural network model, the "Evaluate" method is called, which, based on cross-validation, calculates the accuracy of the model in percent and displays them to the user in the application in the form of a dialog box. Finally, the "SaveModel" method is called, which saves the created model in the application folder.

6.2. ML.NET Model Results

ResNet50 was identified as the most suitable model after training in the ML.NET framework. ResNet50 is a model of a multilayer neural network of trained characters, which contains 50 layers composed of convolutional and associative layers. The advantage of pre-trained models is their internal weights, which are assigned to individual objects in the image. Such models are usually trained on a large amount of data, and the training process can take several days.

The model was trained 10 times on a dataset containing 43 people and 1545 images. The average accuracy value for this model is 91.15%, with an average training time of 1441.8 s. We can consider this model a success, as it classifies in any case with an accuracy of over 80%. The model automatically determined the test set.

7. Complex Discussion—Evaluation of Results from Experiments

We tested the ML.NET model by selecting other different predefined pre-trained ResNet models. The model was tested with the integration of ResNet50, ResNet101 and ResNet152. Epochs were tested with values of 50, 100 and 200 and learning speed values of 0.01 and 0.001. The number of samples determined between one forward and reverse learning cycle was tested with values of 10, 15 and 30.

The EmguCV model demonstrated classification capability with an average success rate of 81.95%, and the model created in ML.NET, based on the pre-trained ResNet50 model, demonstrated classification capability with an average of up to 91.15% accuracy. The success rate of both models exceeded 80%, and therefore, we can consider them successful. We created the third model in the Tensorflow.Net software library. The aim of this model was to create a model of a multilayer neural network that would be able to classify the face of a person in an image based on the data of vector distances between individual facial features. Based on the selected network settings and input data type, this model demonstrated the ability to successfully classify with only 13.31% on average. Therefore, we consider this model not applicable for face classification based on vector values of distances of individual areas of interest. However, these data could be suitable as supplementary data to other neural network models, which would serve as extra data describing individuals.

The EmguCV model implements the classification directly from the extracted pixel values of the image with a resolution of 28 × 28 pixels. The ML.NET model converts the input image with a resolution of 224 × 224 pixels by a series of convolutional and associative layers, and classification is then performed based on their outputs. For this reason, the time required to train the model created in EmguCV is significantly less. The average training time for this model is 117.1 s, while the average training time for the ML.NET model is 1441.8 s, as the input dimension of the image is much larger and contains many layers. ML.NET has a higher success rate in classifying faces by 9.2%, and the time required for classification is longer, though. This makes the EmguCV model more suitable for real-time classification. The summary table shows the significant differences between the individual models, both in terms of the time needed for training and the overall accuracy of the classification (Table 1).

Table 1. Overall accuracy of the classification.

	EmguCV		**Tensorflow.Net**		**ML.Net**	
No.	**Accuracy (%)**	**Time (s)**	**Accuracy (%)**	**Time (s)**	**Accuracy (%)**	**Time (s)**
1	84.55	117	14.29	14	92.53	1428
2	83.19	119	15.12	15	91.75	1393
3	84.87	120	14.29	15	92.14	1453
4	81.51	115	11.9	13	91.84	1427
5	80.67	113	12.62	15	89.73	1461
6	83.19	117	13.45	14	90.18	1431
7	79.83	114	12.62	14	90.32	1478
8	80.67	117	14.29	13	89.84	1488
9	79.83	121	11.9	14	91.73	1402
10	81.19	118	12.62	16	91.42	1457
Average	81.95	117.1	13.31	14.3	91.148	1441.8

Our experiments successfully demonstrated that the multilayer neural network model in the EmguCV framework is capable of reliable classification of the faces on which it was trained. EmguCV is a popular library used mainly for face detection and extraction by pre-trained models with the possibility of their modification. Many scientific articles and publications [38], ref. [43] use the EmguCV library for only two phases: detection and extraction. Neural network models are often created in other frameworks, which provide more algorithms and also the possibility of more detailed modification of models when creating neural networks, but the training is implemented on a smaller amount of data. The EmguCV model we designed was trained on a dataset containing 1454 face images belonging to 43 different people. Although our dataset contains a larger number of classification classes compared to them, the model is still able to correctly classify faces with approximately the same accuracy as the described models [38,43].

Provided the EmguCV model was trained with a much larger number of people and facial images, it might be able to achieve even higher accuracy. This is also confirmed by research [46], in which the authors addressed the issue of language translation (from Chinese to English). The translation model has been trained on a huge amount of article data extracted from the Internet. These articles were not modified in any way and also contained a lot of unwanted data, such as links and images. As a result, the translation model learned to recognize certain patterns based on a large amount of data and then used them to translate phrases with great success.

ML.NET achieved a higher classification success by implementing an already predefined (pre-trained) convolutional network, while the differences in successful classifications could be reduced by expanding the dataset.

8. Conclusions

Neural networks are becoming more and more used for applications of various kinds these days. The application designed by us enables the insertion of images and the creation of your own datasets, based on which the user can train his own model with the parameters that we predefined in the work. Models can then be saved and integrated into other applications. The paper describes how to design and test three different models within this application.

In this paper, we used convolutional neural networks in individual models (EmguCV, ML.NET and Tensorflow.Net). The use of convolutional networks makes it possible to train networks with high classification success in various application areas, for example, in the detection of invisible cracks in ceramic materials [47] or the classification of operation cases in electric arc welding machines [48].

The EmguCV library is an extension of the OpenCV. From the available databases of professional articles (e.g., Scopus), we can see that the issue of applying EmguCV is more oriented to its use in the first phase of the recognition process—detection [49,50]. Currently not exist many relevant sources that refer to extraction or classification phases. In this paper, we compared different ways of classification using EmguCV, ML.NET and Tensorflow.Net. The last two mentioned are among the most used. However, as we state in the paper, EmguCV contains cascade classificators very similar to the one used in the application of the Viola–Jones algorithm. In the article, we, therefore, showed that we could successfully use the EmguCV model to train and test the speed also in the classification phase.

We consider the ability of the EmguCV model to train and test the speed and the classification accuracy of the ML.NET model to be the advantages of the created models with classification accuracy above 80%. We consider the inaccuracies of the classification in a significant rotation of the person's face and the inaccuracies of the classification in very low light conditions to be the disadvantages of the created models. These problems could be eliminated by expanding the dataset by adding photos of faces that are tilted or rotated in different lighting conditions. Other disadvantages are the inaccuracies in face extraction with the Viola–Jones algorithm, which is not able to detect a face in very dark images, or the face detected in such situations does not represent the actual face shown in the photo.

Author Contributions: Methodology, M.M., S.K. and Z.B.; software, J.B.; validation, D.T., M.M. and J.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hajek, P.; Barushka, A.; Munk, M. Neural Networks with Emotion Associations, Topic Modeling and Supervised Term Weighting for Sentiment Analysis. *Int. J. Neural Syst.* **2021**, *31*, 2150013. https://doi.org/10.1142/S0129065721500131.
2. Devi, M.K.; Prabhu, K. Face Emotion Classification using AMSER with Artificial Neural Networks. In *Proceedings of the 6th International Conference on Advanced Computing and Communication Systems, ICACCS 2020, Coimbatore, India, 6–7 March 2020*; IEEE: Piscataway, NJ, USA, 2020; pp. 148–154. https://doi.org/10.1109/ICACCS48705.2020.9074348.
3. Chen, R.; Zhang, H.; Liu, J. Multi-Attention augmented network for single image super-resolution. *Pattern Recognit.* **2022**, *122*, 108349. https://doi.org/10.1016/J.PATCOG.2021.108349.
4. Tsai, C.C.; Cheng, W.C.; Taur, J.S.; Tao, C.W. Face detection using eigenface and neural network. In *Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics, Taipei, Taiwan, 8–11 October 2006*; IEEE: Piscataway, NJ, USA, 2006; Volume 5, pp. 4343–4347. https://doi.org/10.1109/ICSMC.2006.384817.
5. Šaloun, P.; Stonawski, J.; Zelinka, I. Recommending new links in social networks using face recognition. In *Proceedings of the 8th International Workshop on Semantic and Social Media Adaptation and Personalization, SMAP 2013, Bayonne, France, 12–13 December 2013*; IEEE: Piscataway, NJ, USA, 2013; pp. 89–94. https://doi.org/10.1109/SMAP.2013.13.
6. Shanmuganathan, M.; Nalini, C. Face Detection based on Extraction by K-NNC and NMC Framework. In Proceedings of the 3rd International Conference on Intelligent Sustainable Systems, ICISS 2020, Thoothukudi, India, 3–5 December 2020; pp. 1006–1011. https://doi.org/10.1109/ICISS49785.2020.9315955.
7. Sumalakshmi, C.H.; Vasuki, P. Facial Expression Recognition using Feature Extraction with Hybrid KFDA and CLBP. *Int. J. Adv. Sci. Technol.* **2020**, *29*, 2102–2112.
8. Miao, Y.; Wang, X.; Chen, J.; Zhang, X.; Lee, Y.T. Single Image Based Interactive Modeling for Modular Architectures Using Imaging Consistency. *Jisuanji Fuzhu Sheji Yu Tuxingxue Xuebao/J. Comput. Aided Des. Comput. Graph.* **2018**, *30*, 2001–2010. https://doi.org/10.3724/SP.J.1089.2018.17033.
9. Kajita, T.; Yuge, R.; Sultoni, S.; Abdullah, A.G. Real Time Facial Recognition Using Principal Component Analysis (PCA) And EmguCV. *IOP Conf. Ser. Mater. Sci. Eng.* **2018**, *384*, 012079. https://doi.org/10.1088/1757-899X/384/1/012079.
10. Fadhil, A.T.; Abbar, K.A.; Qusay, A.M. Computer Vision-Based System for Classification and Sorting Color Objects. *IOP Conf. Ser. Mater. Sci. Eng.* **2020**, *745*, 012030. https://doi.org/10.1088/1757-899X/745/1/012030.
11. Beohar, D.; Rasool, A. Handwritten digit recognition of MNIST dataset using deep learning state-of-the-art artificial neural network (ANN) and Convolutional Neural Network (CNN). In *Proceedings of the 2021 International Conference on Emerging Smart Computing and Informatics, ESCI 2021, Pune, India, 5–7 March 2021*; IEEE: Piscataway, NJ, USA, 2021; pp. 542–548. https://doi.org/10.1109/ESCI50559.2021.9396870.
12. Yin, Y.; Juan, C.; Chakraborty, J.; McGuire, M.P. Classification of Eye Tracking Data Using a Convolutional Neural Network. In *Proceedings of the 17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA, 17–20 December 2018*; IEEE: Piscataway, NJ, USA, 2019; pp. 530–535. https://doi.org/10.1109/ICMLA.2018.00085.
13. Huang, X.; Lei, X.; Pan, H.; Li, D. Algorithm of image classification based on Atrous-CNN. In *Proceedings of the 31st Chinese Control and Decision Conference, CCDC 2019, Nanchang, China, 3–5 June 2019*; IEEE: Piscataway, NJ, USA, 2019; pp. 5324–5328. https://doi.org/10.1109/CCDC.2019.8833250.
14. Eluyode, O.; Journal, E.; Eluyode, O.S.; Akomolafe, D.T. Scholars Research Library Comparative study of biological and artificial neural networks. *Appl. Eng. Sci. Res.* **2013**, *2*, 36–46.
15. Schrimpf, M.; Kubilius, J.; Hong, H.; Majaj, N.J.; Rajalingham, R.; Issa, E.B.; Kar, K.; Bashivan, P.; Prescott-Roy, J.; Geiger, F.; et al. Brain-Score: Which Artificial Neural Network for Object Recognition is most Brain-Like? *bioRxiv* **2018**, 407007. https://doi.org/10.1101/407007.
16. Tekkali, C.G.; Vijaya, J. A Survey: Methodologies used for Fraud Detection in Digital Transactions. In *Proceedings of the 2nd International Conference on Electronics and Sustainable Communication Systems, ICESC 2021, Coimbatore, India, 4–6 August 2021*; IEEE: Piscataway, NJ, USA, 2021; pp. 1758–1765. https://doi.org/10.1109/ICESC51422.2021.9532915.
17. Siregar, S.P.; Wanto, A.; Tunas, S.; Pematangsiantar, B. Analysis of Artificial Neural Network Accuracy Using Backpropagation Algorithm in Predicting Process (Forecasting). *IJISTECH Int. J. Inf. Syst. Technol.* **2017**, *1*, 34–42. https://doi.org/10.30645/IJISTECH.V1I1.4.
18. Saravanan, R.; Sujatha, P. A State of Art Techniques on Machine Learning Algorithms: A Perspective of Supervised Learning Approaches in Data Classification. In *Proceedings of the 2nd International Conference on Intelligent Computing and Control Systems, ICICCS 2018, Madurai, India, 14–15 June 2018*; IEEE: Piscataway, NJ, USA, 2019; pp. 945–949. https://doi.org/10.1109/ICCONS.2018.8663155.

19. Osisanwo, F.Y.; Akinsola, J.E.T.; Awodele, O.; Hinmikaiye, J.O.; Olakanmi, O.; Akinjobi, J. Supervised Machine Learning Algorithms: Classification and Comparison. *Int. J. Comput. Trends Technol.* **2017**, *48*, 128–138. https://doi.org/10.14445/22312803/IJCTT-V48P126.
20. Kowsari, K.; Meimandi, K.J.; Heidarysafa, M.; Mendu, S.; Barnes, L.; Brown, D. Text Classification Algorithms: A Survey. *Information* **2019**, *10*, 150. https://doi.org/10.3390/INFO10040150.
21. Maxwell, A.E.; Warner, T.A.; Fang, F. Implementation of machine-learning classification in remote sensing: An applied review. *Int. J. Remote Sens.* **2018**, *39*, 2784–2817. https://doi.org/10.1080/01431161.2018.1433343/SUPPL_FILE/TRES_A_1433343_SM5998.ZIP.
22. Xu, Y.; Li, Z.; Yang, J.; Zhang, D. A Survey of Dictionary Learning Algorithms for Face Recognition. *IEEE Access* **2017**, *5*, 8502–8514. https://doi.org/10.1109/ACCESS.2017.2695239.
23. Yan, K.; Huang, S.; Song, Y.; Liu, W.; Fan, N. Face recognition based on convolution neural network. *Chin. Control. Conf. CCC* **2017**, 4077–4081. https://doi.org/10.23919/CHICC.2017.8027997.
24. Khashman, A. Application of an emotional neural network to facial recognition. *Neural Comput. Appl.* **2009**, *18*, 309–320. https://doi.org/10.1007/S00521-008-0212-4/TABLES/2.
25. Oyedotun, O.K.; Khashman, A. Prototype-incorporated emotional neural network. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 3560–3572. https://doi.org/10.1109/TNNLS.2017.2730179.
26. Albawi, S.; Mohammed, T.A.; Al-Zawi, S. Understanding of a convolutional neural network. In *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017, Antalya, Turkey, 21–23 August 2017*; IEEE: Piscataway, NJ, USA, 2018; Volume 2018-January, pp. 1–6. https://doi.org/10.1109/ICENGTECHNOL.2017.8308186.
27. Khan, A.; Sohail, A.; Zahoora, U.; Qureshi, A.S. A survey of the recent architectures of deep convolutional neural networks. *Artif. Intell. Rev.* **2020**, *53*, 5455–5516. https://doi.org/10.1007/S10462-020-09825-6.
28. Kuo, C.C.J. Understanding convolutional neural networks with a mathematical model. *J. Vis. Commun. Image Represent.* **2016**, *41*, 406–413. https://doi.org/10.1016/J.JVCIR.2016.11.003.
29. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. https://doi.org/10.1145/3065386.
30. Zeiler, M.D.; Fergus, R. Visualizing and Understanding Convolutional Networks. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). In *Computer Vision—ECCV 2014*; Springer: Cham, Switzerland, 2014; Volume 8689 LNCS, pp. 818–833. https://doi.org/10.1007/978-3-319-10590-1_53.
31. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015.
32. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 07–12 June 2015*; IEEE: Piscataway, NJ, USA, 2015; pp. 1–9. https://doi.org/10.1109/CVPR.2015.7298594.
33. Khalajzadeh, H.; Mansouri, M.; Teshnehlab, M. Face Recognition Using Convolutional Neural Network and Simple Logistic Classifier. *Adv. Intell. Syst. Comput.* **2014**, *223*, 197–207. https://doi.org/10.1007/978-3-319-00930-8_18.
34. Aloysius, N.; Geetha, M. A review on deep convolutional neural networks. In *Proceedings of the 2017 IEEE International Conference on Communication and Signal Processing, ICCSP 2017, Chennai, India, 6–8 April 2017*; IEEE: Piscataway, NJ, USA, 2018; Volume 2018, pp. 588–592. https://doi.org/10.1109/ICCSP.2017.8286426.
35. Severyn, A.; Moschitti, A. UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification. *Assoc. Comput. Linguist.* **2015**, 464–469.
36. Snoek, J.; Larochelle, H.; Adams, R.P. Practical Bayesian Optimization of Machine Learning Algorithms. *Adv. Neural Inf. Processing Syst.* **2012**, *2*, 2951–2959.
37. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and Harnessing Adversarial Examples. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015.
38. Balodis, A.; Grabusts, P. Oject Recognition Methods in Cimputer Vision Using Image Processing Library Emgu CV. In *Human Environment Technologies;* Rezekne Academy of Technologies: Rezekne, Latvia, 2017; pp. 41–46. https://doi.org/10.17770/HET2017.21.3582.
39. Wang, Y.-Q. An Analysis of the Viola-Jones Face Detection Algorithm. *Image Processing Line* **2014**, *4*, 128–148. https://doi.org/10.5201/IPOL.2014.104.
40. Ghosh, M.; Sarkar, T.; Chokhani, D.; Dey, A. Face Detection and Extraction Using Viola–Jones Algorithm. *Lect. Notes Electr. Eng.* **2022**, *786*, 93–107. https://doi.org/10.1007/978-981-16-4035-3_9.
41. Soo, S. Object detection using Haar-cascade Classifier. *Inst. Comput. Sci.* **2014**, *2*, 1–12.
42. Gangopadhyay, I.; Chatterjee, A.; Das, I. Face Detection and Expression Recognition Using Haar Cascade Classifier and Fisherface Algorithm. In *Advances in Intelligent Systems and Computing*; Springer: Cham, Switzerland, 2019; Volume 922, pp. 1–11. https://doi.org/10.1007/978-981-13-6783-0_1.
43. Alankar, B.; Ammar, M.S.; Kaur, H.; Alankar, B.; Ammar, Á.M.S.; Kaur, Á.H. Facial Emotion Detection Using Deep Learning and Haar Cascade Face Identification Algorithm. In *Lecture Notes in Networks and Systems*; Springer: Cham, Switzerland, 2021; Volume 202 LNNS, pp. 163–180. https://doi.org/10.1007/978-981-16-0695-3_17.

44. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. {TensorFlow}: A System for {Large-Scale} Machine Learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’16), Savannah, GA, USA, 2–4 November 2016; Volume 10. ISBN 978-1-931971-33-1.
45. Yale Face Database. Available online: http://cvc.cs.yale.edu/cvc/projects/yalefacesB/yalefacesB.html (accessed on 19 January 2022).
46. Halevy, A.; Norvig, P.; Pereira, F. The unreasonable effectiveness of data. *IEEE Intell. Syst.* **2009**, *24*, 8–12. https://doi.org/10.1109/MIS.2009.36.
47. Nogay, H.S.; Akinci, T.C.; Yilmaz, M. Detection of invisible cracks in ceramic materials using by pre-trained deep convolutional neural network. *Neural Comput. Appl.* **2022**, *34*, 1423–1432. https://doi.org/10.1007/S00521-021-06652-W/FIGURES/6.
48. Nogay, H.S.; Akinci, T.C. Classification of operation cases in electric arc welding wachine by using deep convolutional neural networks. *Neural Comput. Appl.* **2021**, *33*, 6657–6670. https://doi.org/10.1007/S00521-020-05436-Y/FIGURES/12.
49. Wicaksana, B.A.; Sari, R.F. Implementing text information display of detected color for partially color blinded person using .NET platform and EmguCV library. In *Proceedings of the 2011 International Conference on Information Technology and Multimedia: "Ubiquitous ICT for Sustainable and Green Living", ICIM 2011, Kuala Lumpur, Malaysia, 14–16 November 2011*; IEEE: Piscataway, NJ, USA, 2011. https://doi.org/10.1109/ICIMU.2011.6122760.
50. Culler, D.; Long, J. A Prototype Smart Materials Warehouse Application Implemented Using Custom Mobile Robots and Open Source Vision Technology Developed Using EmguCV. *Procedia Manuf.* **2016**, *5*, 1092–1106. https://doi.org/10.1016/J.PROMFG.2016.08.080.