UDC 004.853
DOI: 10.15587/1729-4061.2023.293682

CONSTRUCTION OF A NEURAL NETWORK FOR HANDWRITTEN DIGITS RECOGNITION BASED ON TENSORFLOW LIBRARY APPLYING AN ERROR BACKPROPAGATION ALGORITHM

Tetiana Filimonova
PhD*
Hanna Samoylenko
PhD, Associate Professor*
Anna Selivanova*
Corresponding author
E-mail: ann.selivanova1@gmail.com
Yurii Yurchenko*
Alexei Parashchak
PhD*
*Department of Computer Science and Information Systems
State University of Trade and Economics
Kyoto str., 19, Kyiv, Ukraine, 02156

The object of this study is a neural network for recognizing handwritten digits based on the TensorFlow library using the backpropagation algorithm.

The main problem addressed is the development of an effective model with high recognition accuracy. Working on such a task is important as it allows understanding how algorithms and models can effectively work with real data and helps improve machine learning techniques.

It has been determined that after 20 training epochs, the loss function is 0.105, and the recognition accuracy is 0.976, comparable to human recognition capability. The classification report indicates that the model is effectively trained on training data and demonstrates high accuracy on test data, capable of generalizing information to new examples. Visualization of recognition results confirms that the model correctly recognizes even poorly written digits.

The results can be explained by the peculiarities of the model architecture, optimal selection of hyperparameters, and successful use of the backpropagation algorithm, which was not explicitly specified during model training. TensorFlow provided a convenient toolkit for implementing the neural network and optimizing its parameters. As a result, the model has a fairly high accuracy in image recognition.

A significant feature of the results is the high recognition accuracy achieved through the optimal model architecture, correct choice of hyperparameters, and effective use of the backpropagation algorithm. Unlike models built using Keras and convolutional layers, the research model quickly learns, which is important, and does not compromise on accuracy. This result was made possible by the above features of model construction.

The results could be practically applied in the field of handwritten character recognition, especially in automated document classification systems, in banking recognition systems, and in other areas where the accuracy of handwritten character recognition is essential

Keywords: neural network, loss function, gradient descent, neural network accuracy

Received date 08.10.2023
Accepted date 14.12.2023
Published date 28.12.2023

***How to Cite**: Filimonova, T., Samoylenko, H., Selivanova, A., Yurchenko, Y., Parashchak, A. (2023). Construction of a neural network for handwritten digits recognition based on tensorflow library applying an error backpropagation algorithm. Eastern-European Journal of Enterprise Technologies, 6 (2 (126)), 25–32. doi: https://doi.org/10.15587/1729-4061.2023.293682*

1. Introduction

Image recognition includes many tasks, namely, recognition of handwritten digits, recognition of faces and other physical objects. Image recognition methods are used to create computer vision devices in robotics, to analyze aerospace images, in automated image analysis systems in the medical field, etc. Artificial neural networks are used to recognize objects in images [1]. The most effective are convolutional neural networks [2].

Modern requirements and tasks in various sectors of society require the improvement of pattern recognition methods, which makes it urgent to carry out scientific research in this area. Such research is aimed at improving algorithms, the architecture of artificial neural networks, as well as other methods used to solve pattern recognition tasks.

Scientific research on pattern recognition is important because it contributes to the development and implementation of advanced technologies that determine further advances in computing and engineering.

Handwritten digit recognition is the ability of a computer to recognize numbers written by hand. For the machine, this is not the easiest task because each written number may differ from the reference writing. In the case of recognition, the solution is that the machine is able to recognize the digit in the image. The problem of recognition of handwritten digits is characterized by practical significance due to numerous fields of application and modern requirements of society. This task has applications in automated systems that read and process documents such as checks, tax returns, and sort and process mail. Accurate recognition of handwritten digits in such cases is crucial for the implementation of effective and reliable accounting and control systems. Also of practical importance is the application of recognition of handwritten digits in the field of visual analysis of data from paper documents. The obtained results can be used to transfer information from paper form to digital format, which is important for optimizing work and improving the efficiency of business processes.

Handwritten digit recognition systems are also proving to be an integral element in the field of machine learning and artificial intelligence. They help solve the tasks of classification, identification, and authentication, which are key aspects in a number of modern technological solutions.

Therefore, research to design a neural network for the recognition of handwritten digits, which learns quickly and has high recognition accuracy, is relevant.

2. Literature review and problem statement

To solve the problem of image recognition, classification methods are usually used, which make it possible to assign objects in images to a certain class based on the existing features that characterize such objects. Recognition of handwritten digits is a special case of the problem of pattern recognition. When solving the problem of recognizing handwritten digits, very complex but also more accurate methods are used as classification methods, for example, the method of support vectors [3], as well as various artificial neural networks [4, 5].

In [6, 7] it is stated that the use of neural networks is the most promising direction in the recognition of handwritten symbols, including numbers. But issues related to achieving sufficient accuracy of the model remain unresolved. The results of many studies show that the accuracy of character recognition depends on the nature of model training, on the amount of training samples, and other characteristics [4, 7–9]. A convolutional neural network can be an option to overcome the relevant difficulties. This type of model for recognizing handwritten digits was used in works [6, 7, 10, 11]. Indeed, these models have high accuracy, but the training time is much longer than that of a multilayer neural network using TensorFlow and Keras [12].

The architecture of the model described in this work, in comparison with other types of neural networks, has high accuracy and a much smaller error in recognizing handwritten digits [13]. This method was developed as a generalization of the Withrow-Hoff method for multilayer neural networks with nonlinear differentiated activation functions [14–16]. Its main advantage is the asymptotic convergence of the learning process, which guarantees the potential achievement of the necessary accuracy of the neural network.

All this gives reason to claim that it is appropriate to conduct a study to design a neural network architecture based on TensorFlow mechanisms using the error backpropagation algorithm for recognizing handwritten digits.

3. The aim and objectives of the study

The goal of our research is to design a fully connected neural network using TensorFlow mechanisms, applying the error backpropagation algorithm for recognizing handwritten digits. This will make it possible to build a model with high recognition accuracy.

To achieve the goal, the following tasks are set:

– to design the neural network architecture: determine the number of layers, the number of neurons, the activation function in each layer;

– to train the model according to the algorithm of error backpropagation: using the computational graph, calculate the partial derivatives according to the parameters of the model layers with respect to the cross-entropy function, then use the obtained gradients in the gradient descent algorithm;

– to investigate patterns of loss and accuracy functions for training and test data;

– to evaluate the performance of the model;

– to visualize the work of the constructed model.

4. The study materials and methods

The object of research is the recognition of handwritten digits using TesorFlow mechanisms.

Research hypothesis – the proposed research model, built on the basis of TensorFlow mechanisms, could have high accuracy, comparable to human recognition ability.

Assumptions and simplifications adopted in the research process – the proposed model is based on the assumption that it generalizes regularities from training sets for high-accuracy recognition with no retraining.

A variety of theoretical methods, software, and hardware, as well as specific conditions for experiments and model validation were used in the development of a neural network for recognizing handwritten digits.

Theoretical methods:

1. Neural network architecture: the theory of designing and optimizing the architecture of neural networks is applied, taking into account the principles of the layers, the number of neurons in each layer, and the relationships between them.

2. Optimization of hyperparameters: theoretical methods of optimization of hyperparameters, such as learning rate, number of layers, activation functions, etc., are used.

The following mathematical apparatus was chosen for neural network research:

1. Activation functions and probability distribution:

– the ReLU (Rectified Linear Unit) activation function is a non-linear function that allows the model to learn complex dependences in the data. It is used in neural networks due to its efficiency and simplicity, and has the following form [17]:

$$f(x)=\max(0; x),$$

where x is the input signal, $f(x)$ is the output signal;

– the Softmax activation function is used at the output of the last layer to obtain a probability distribution for classification. Softmax makes it possible to convert the initial values into probabilities for different classes [17]:

$$p_i=\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}},$$

where x_i is the input signal for the ith neuron;

p_i is the probability for each element x_i;

n is the number of neurons.

2. Losses and optimization:

– cross entropy is used as a loss function. The function is used in classification tasks, it makes it possible to measure how accurately the probabilities calculated using Softmax reflect the real distribution of classes;

– to optimize the parameters of the model, the Adam optimizer is used, which effectively adjusts the learning rate for each parameter;

– a certain batch size (*bath_size*) is selected for model training, and the training speed is set. A large batch size can make training faster, but also requires more memory.

Software:

1. TensorFlow: the TensorFlow framework is used to implement the neural network, which provides a convenient interface for constructing and training deep learning models.

2. Jupyter Notebook: an interactive environment used for step-by-step experimentation and visualization of results.

3. Python Libraries: Additional libraries such as NumPy, Matplotlib, and Pandas are used to work with data, visualize, and analyze results.

Hardware:

1. Computer:

– a processor with high speed and support for instructions used for calculations in deep learning (for example, Intel Core i7 or Intel Core i9);

– sufficient amount of RAM (minimum 16 GB).

2. Graphics processor (GPU), modern video cards.

3. Other components, namely: Internet connection for downloading libraries and data, high-resolution monitor.

Conditions for the experiment and validation:

1. Training and test data: The study was conducted on defined training and test data sets to ensure objectivity of the results. The MNIST dataset was used for training, which was created from several datasets of scanned documents from the US National Institute of Standards and Technology (NIST). Hence the name of the data set, the modified NIST data set, or MNIST [18].

Digit images are taken from random scanned documents, size normalized and centered. These transformations make the dataset suitable for model evaluation, allowing the developer to focus on machine learning with minimal data preparation. Each image is a 28 by 28-pixel square (784 pixels in total). A standard dataset is used for model evaluation and comparison, where 60,000 images are used to train the model and a separate set of 10,000 images is used to validate it. So, there are 10 numbers (from 0 to 9), or 10 classes, to predict.

2. Model validation and evaluation:

– accuracy metrics: we used metrics such as accuracy, loss function to evaluate model performance;

– cross-validation: cross-validation was applied to check the stability and generalizability of the model.

3. The behavior of the loss and accuracy functions for the training and test data sets during training was studied.

4. Visualization of results: examples of correct and incorrect recognition are visualized to analyze the model's performance.

5. Results of model construction studies

5. 1. Architecture of the model

The following neural network structure is proposed. Each image with a size of 28×28 pixels is stretched into a vector with a size of 784 elements and transferred to the first layer of 128 neurons. Each neuron in this layer has a ReLU activation function. Next is the output layer with the Softmax activation function.

Thus, the output that takes the largest value is the number represented in the image (Fig. 1).

For example, if y_2 takes the largest value, then the number 2 is displayed. If y_2 takes the largest value, then the number 1 is displayed, etc.

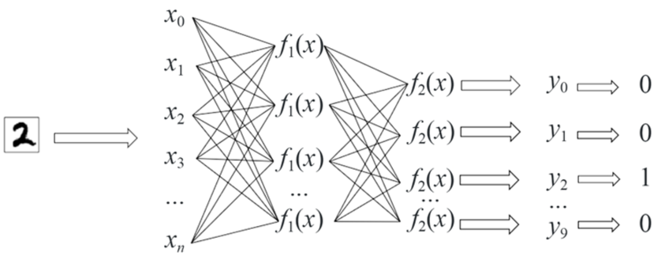
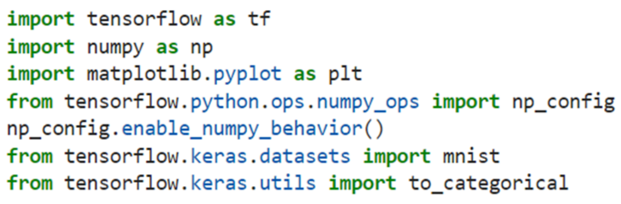


Fig. 1. Neural network architecture

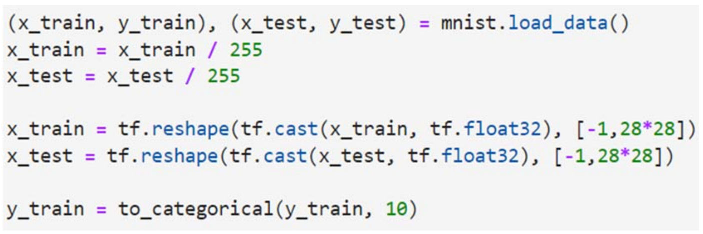
The following procedures are proposed for the software implementation of the presented neural network and its training for recognizing handwritten digits. First, the necessary modules are imported (Fig. 2).



```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.python.ops.numpy_ops import np_config
np_config.enable_numpy_behavior()
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
```

Fig. 2. Imported libraries

TensorFlow is a specialized library for calculations using data flow graphs (graph nodes are mathematical operations, edges are multidimensional data arrays). In this library, parallel calculations on the GPU are available after installing the necessary drivers and setting special parameters. Keras is a library for building deep learning neural networks that provides a high-level application programming interface (API) that uses TensorFlow as a backend. Next, the training and test samples are loaded. *x_train* contains the images that will be fed to the input. The size of each image is 28×28 pixels. Each pixel is represented by a value between 0 and 255. The next step is to normalize these images by representing them as 0 to 1 and converting these images into a single vector. That is, *x_train* will be a training dataset consisting of vectors of length 784 elements. The following transformation is done using the *reshape*() and *to_categorical*() functions. Similar actions were performed for the test sample *x_test* (Fig. 3).



```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255
x_test = x_test / 255

x_train = tf.reshape(tf.cast(x_train, tf.float32), [-1,28*28])
x_test = tf.reshape(tf.cast(x_test, tf.float32), [-1,28*28])

y_train = to_categorical(y_train, 10)
```

Fig. 3. Normalization of input data

It is also necessary to transform the vector *y_train*. Initially, this vector contains only numbers. The numbers are converted to the original data format (Fig. 4).

Fig. 4 shows examples of representing a number in the form of a vector. For example, the number 5 corresponds to a vector consisting of ten elements, in which there is a unity in the corresponding place, and zeros in other places. Similar actions occur for other numbers. The transformation is performed using the function *y_train*=*to_categorical*(*y_train*, 10).

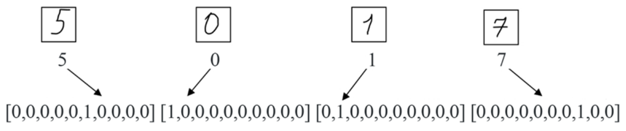


Fig. 4. Representation of vector *y_train*

After data preparation, you can create a model of a fully connected neural network layer. This is implemented using the *DenseNN(tf.Module) class* and the *_call_(self, x)* method. In class *DenseNN(tf.Module)* the function *activate='relu'* is added in the constructor and this type *self.activate=activate* is stored in the local properties. In the *call_(self, x)* method, we form the initial weight coefficients *self.w*, *self.b* depending on the size of the vector *x*. It is executed once. Next, the input sums on each neuron *y=x@self.w+self.b* were calculated, and these sums were passed through the corresponding activation functions. If the parameter *self.activate==‘relu’* then the ReLU function is selected. If the parameter *self.activate==‘softmax’* then the Softmax function is selected. The ReLU and Softmax functions are built-in in TensorFlow, you can access them using the following constructs *tf.nn.relu(y)*, *tf.nn.softmax(y)* (Fig. 5).

The Softmax activation function is used at the output layer to convert the output values into probabilistic values and makes it possible to select one class out of 10 as the output model prediction.

It is worth noting that the number of connections in the first layer is 784×784, in the second layer – 128×10.

5. 2. Model training

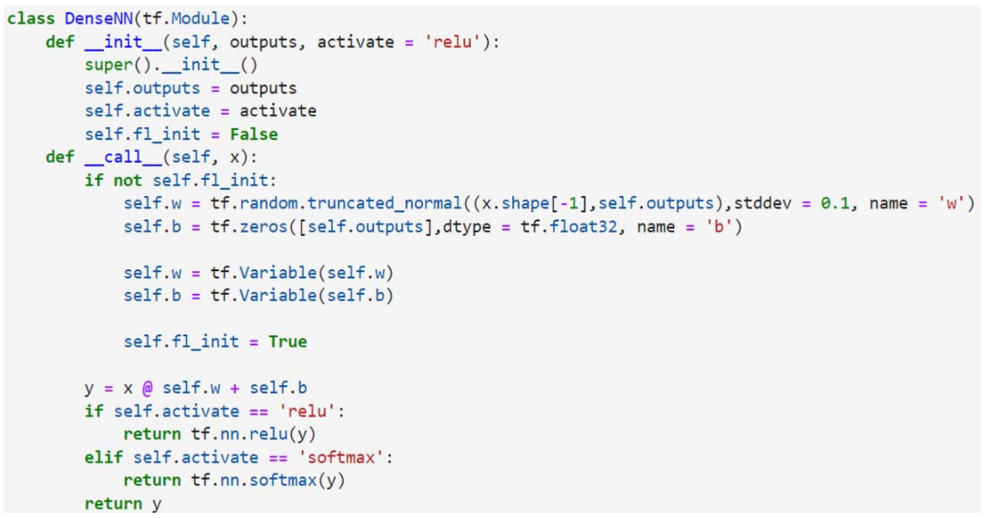
Now you need to train the neural network, that is, find the values of the parameters of the weighting coefficients using the gradient descent algorithm. To do this, you need to set the loss function. In classification tasks with more than two classes, one can apply categorical cross-entropy, which is built into the TensorFlow package. An optimizer for gradient descent should be defined: *tf.optimizers.Adam(learning_rate*=0.001). The set learning step is 0.001 (Fig. 6).

A generic *SequentialModule* class is defined here, containing two fully connected layers of 128 and 10 neurons, respectively. The *_call_* function sequentially calls these layers and returns the result of the model. This representation is convenient because it is possible to work with the neural network as a single model.

It is worth noting that *y_true* and *y_pred* are sets of one-dimensional *batch_size* vectors, that is, the *cross_entropy* function will be applied to the data set as a whole, not to a single observation. Since the goal is to have a single, specific number as the result, all values for each observation in the batch are averaged using the *reduce_mean()* function.

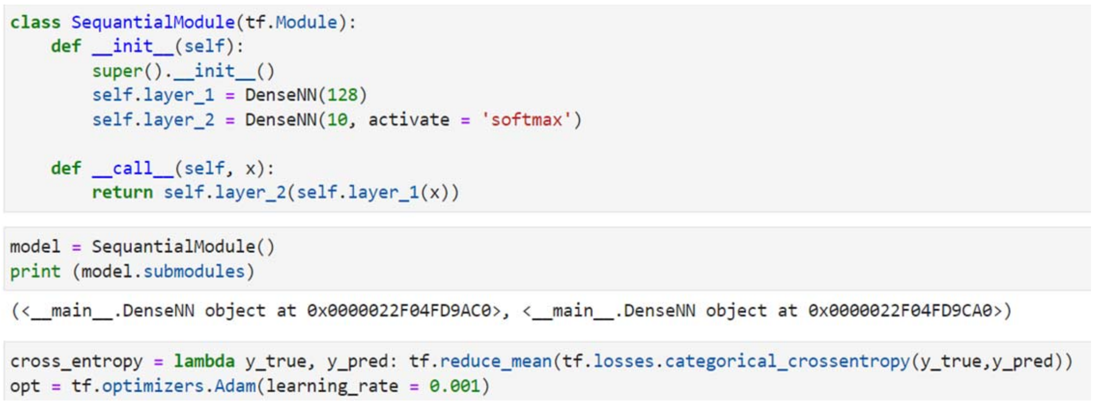
It is necessary to determine the parameters of the model necessary for training (Fig. 7).

After splitting and shuffling the sample, the process of training the model is initiated (Fig. 8).



```
class DenseNN(tf.Module):
    def __init__(self, outputs, activate = 'relu'):
        super().__init__()
        self.outputs = outputs
        self.activate = activate
        self.fl_init = False
    def __call__(self, x):
        if not self.fl_init:
            self.w = tf.random.truncated_normal((x.shape[-1],self.outputs),stddev = 0.1, name = 'w')
            self.b = tf.zeros([self.outputs],dtype = tf.float32, name = 'b')

            self.w = tf.Variable(self.w)
            self.b = tf.Variable(self.b)

            self.fl_init = True

        y = x @ self.w + self.b
        if self.activate == 'relu':
            return tf.nn.relu(y)
        elif self.activate == 'softmax':
            return tf.nn.softmax(y)
        return y
```

Fig. 5. Model of a fully connected layer of a neural network



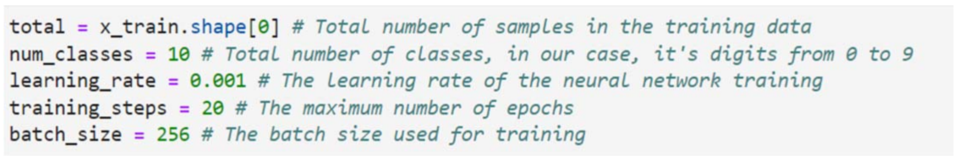
```
class SequantialModule(tf.Module):
    def __init__(self):
        super().__init__()
        self.layer_1 = DenseNN(128)
        self.layer_2 = DenseNN(10, activate = 'softmax')

    def __call__(self, x):
        return self.layer_2(self.layer_1(x))
```

```
model = SequantialModule()
print (model.submodules)
```

```
(<__main__.DenseNN object at 0x0000022F04FD9AC0>, <__main__.DenseNN object at 0x0000022F04FD9CA0>)
```

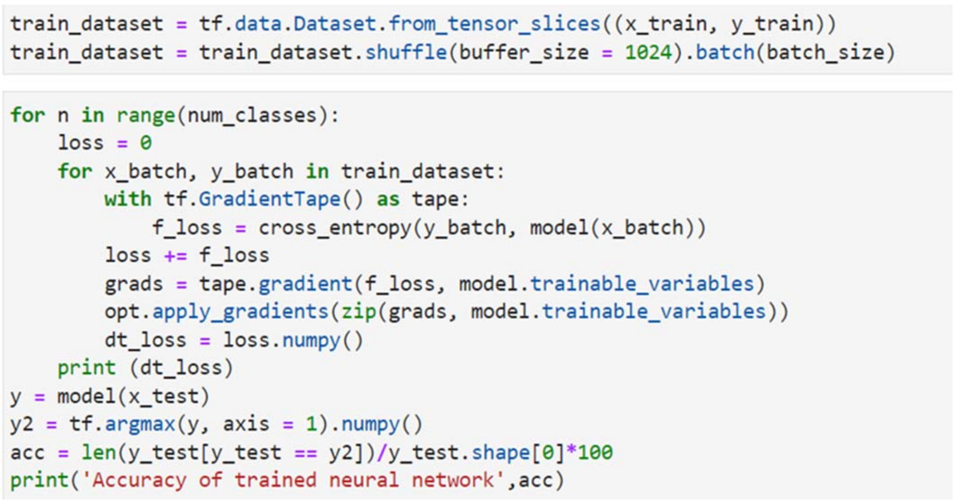
```
cross_entropy = lambda y_true, y_pred: tf.reduce_mean(tf.losses.categorical_crossentropy(y_true,y_pred))
opt = tf.optimizers.Adam(learning_rate = 0.001)
```

Fig. 6. Applications of categorical entropy



```
total = x_train.shape[0] # Total number of samples in the training data
num_classes = 10 # Total number of classes, in our case, it's digits from 0 to 9
learning_rate = 0.001 # The learning rate of the neural network training
training_steps = 20 # The maximum number of epochs
batch_size = 256 # The batch size used for training
```

Fig. 7. Auxiliary parameters for building a neural network



```
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = train_dataset.shuffle(buffer_size = 1024).batch(batch_size)
```

```
for n in range(num_classes):
    loss = 0
    for x_batch, y_batch in train_dataset:
        with tf.GradientTape() as tape:
            f_loss = cross_entropy(y_batch, model(x_batch))
        loss += f_loss
        grads = tape.gradient(f_loss, model.trainable_variables)
        opt.apply_gradients(zip(grads, model.trainable_variables))
        dt_loss = loss.numpy()
    print (dt_loss)
y = model(x_test)
y2 = tf.argmax(y, axis = 1).numpy()
acc = len(y_test[y_test == y2])/y_test.shape[0]*100
print('Accuracy of trained neural network',acc)
```

Fig. 8. Model training cycle

The first cycle is over epochs, and the second over samples of *batch_size* from the random observations of the training sample. The gradients for the weights were then calculated and modified using *apply_gradients*() for the parameters of both model layers. As a result, the loss function goes to zero.

5. 3. Study of patterns of loss and accuracy functions

We calculated values of loss and accuracy functions for training and test data sets. After analyzing the results, it can be concluded that the accuracy function for both sets increases during training, and the loss function for these sets decreases. There is no effect of model retraining (Fig. 9, 10).

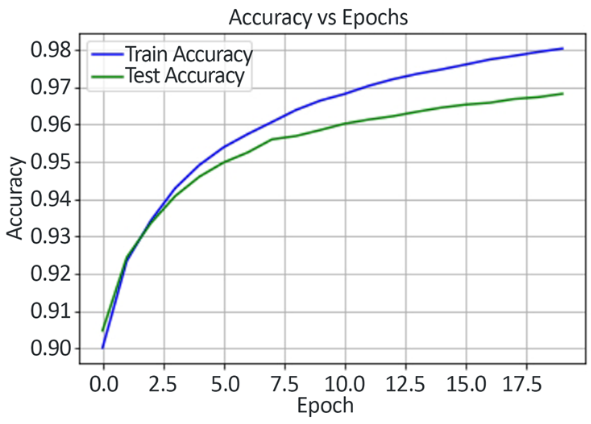


Fig. 9. Precision plot for training and test sampling

After 20 training epochs, the loss function is 0.105 and the image recognition accuracy is 0.976, which is comparable to human recognition ability.

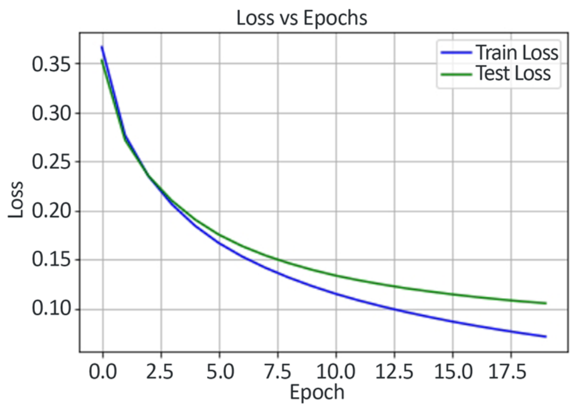
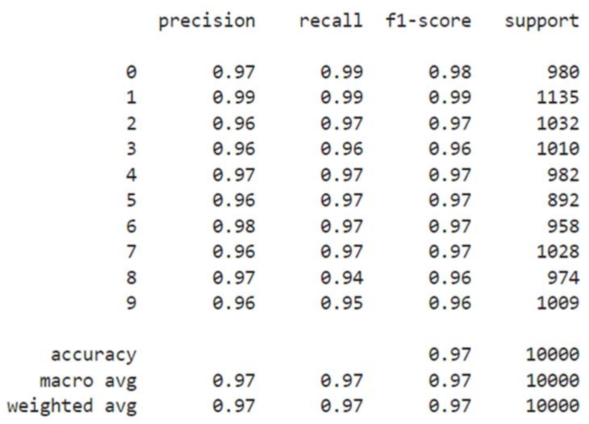


Fig. 10. Plot of the loss function of the training and test sample

5. 4. Assessment of model performance

The assessment was carried out using a classification report (Fig. 11).



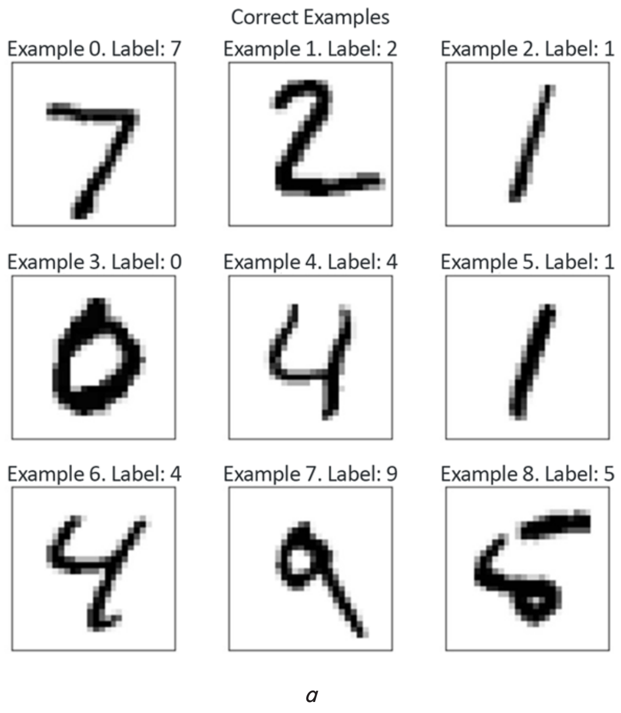
```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.97      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.97      0.97       982
           5       0.96      0.97      0.97       892
           6       0.98      0.97      0.97       958
           7       0.96      0.97      0.97      1028
           8       0.97      0.94      0.96       974
           9       0.96      0.95      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

Fig. 11. Model performance report

The model has high accuracy (precision), completeness (recall) for all classes. Individual and average F1 values are also high, indicating balanced precision and recall. Both the macro avg and weighted avg are high, indicating that the model is stable across the entire data set. The overall accuracy of the model is 0.97, which is also a high value.

5. 5. Visualization of results

The results are visualized by outputting correctly recognized images. There are also images that are recognized with an error (Fig. 12).



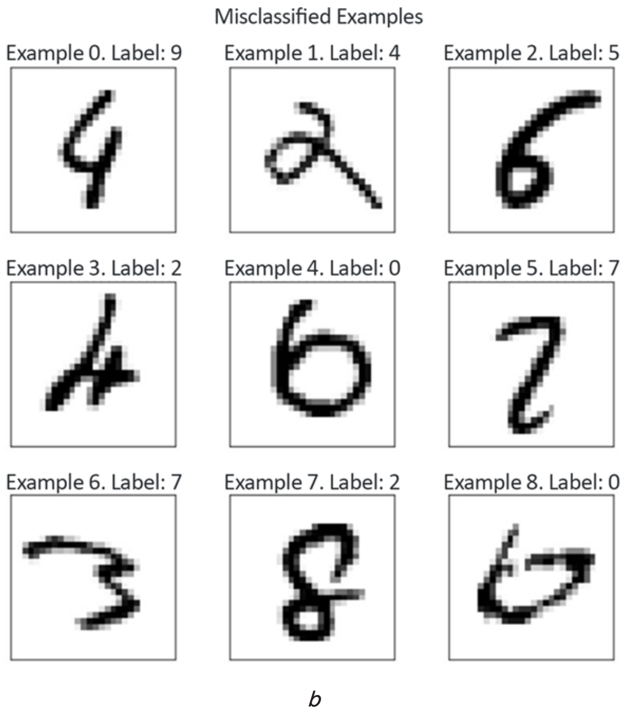


Fig. 12. Results of image recognition: *a* — correctly recognized numbers; *b* — erroneously recognized numbers

Fig. 12 shows that even vaguely written numbers are recognized correctly. There are also falsely recognized images.

6. Discussion of results of the model construction research

As a result of the study, a model was built using TensorFlow mechanisms. After 20 training epochs, the accuracy for the training data set is 0.9808 and for the test data set is 0.976. In addition, the loss function for the training data set is 0.0704, and for the test data set – 0.105 (Fig. 9, 10).

According to the classification report, the model has high precision for all classes from 0.94 to 0.99, recall for all classes from 0.94 to 0.99. Individual and average *F*1 values are also high, indicating balanced precision and recall. The average values of *macro avg* and *weighted avg* are high, which indicates the stability of the model in the entire data set (Fig. 11). In general, the model performs very well for all classes, and it can be considered that it effectively solves the problem of digit classification.

These results indicate that the model was effectively trained on the training data and shows high accuracy on the test data, suggesting its ability to generalize information on new examples.

Visualization of the recognition results confirms that the model correctly recognizes even vaguely written numbers (Fig. 12).

Our results can be explained by the peculiarity of the model architecture, the optimal selection of hyperparameters, and the use of an error backpropagation algorithm, which was not explicitly prescribed. It can be concluded that machine learning algorithms are easily implemented through the computational graph in TensorFlow. As a result, the model has a fairly high accuracy of image recognition (Fig. 11).

The features of the proposed method and our results are as follows. Unlike models built using the Keras library and convolutional layers and therefore have high accuracy, for example, as in [11, 13], the research model is built on the basis of TensorFlow mechanisms and has an accuracy comparable to human recognition ability. At the same time, the model learns quickly, which is important. This result became possible thanks to the features of the model construction listed above. Unlike the model in [12], which has an accuracy of approximately 0.98 and a loss function of 0.026, this neural network has a clear architecture, high accuracy, and recall for all classes from 0.94 to 0.99. The model in [12] also has signs of overtraining. The features of the research model, in contrast to the results reported in [19], where an autoencoder was built for recognizing handwritten digits using Keras, are in the construction of the original architecture using TensorFlow mechanisms, the selection of hyperparameters, and, as a result, obtaining high recognition accuracy.

This study has certain limitations that may affect its applicability, reproducibility, and stability of results. Some potential limitations include:

1. The recognition of handwritten digits is limited by the fact that the model was trained on a specific data set (MNIST). Extending the application to other types of handwriting may require additional training on relevant data.

2. The performance of the model may depend on the input conditions, such as the quality of scanning or the quality of handwritten digits. The model may not give optimal results under unknown or non-standard conditions.

3. Reproducibility of results can be a problem if all training parameters are not specified, such as random initialization of weights and limited resources to reproduce exactly the same experiment.

4. The model may be sensitive to changes in the input data, such as large changes in handwriting style or the addition of noise.

Understanding these limitations makes it possible to adequately use and improve the model, taking them into account when applying them to real conditions and tasks.

Disadvantages of this study:

1. Homogeneity of data. The model is trained exclusively on handwritten digits from the MNIST dataset, which may make it less effective at recognizing other writing styles. To eliminate it, it is possible to expand the data set by adding other sources of handwritten digits or applying the previous techniques on different data sources.

2. Using TensorFlow can lead to high resource consumption, especially with large amounts of data. To eliminate possible optimization of the code, the use of distributed learning on clusters of graphics processing unit (GPU) or hardware acceleration to optimize calculations.

3. The model is limited to digit recognition and its application is limited in other scenarios. It is possible to extend the tasks of the model to the classification of other objects or the recognition of other languages.

The development of this research can be as follows:

1. Extending the work of the model to other types of data other than numbers, for example, handwritten text.

2. Multiclass classification. Extending the classification to more than one class (numbers).

3. Development of methods for automatic hyperparameter selection and model optimization.

Difficulties you may encounter along the way:

1. Adapting the model architecture to work with more diverse data may require new training techniques and corrections to avoid overtraining.

2. The impossibility of predicting how the model will react to the classification of other objects, the question of evaluating the results in a multiclass context.

3. Devising an effective automated process can be challenging, as each task may require a unique approach.

7. Conclusions

1. As a result of our research, an original neural network architecture was designed using TensorFlow mechanisms, consisting of two layers, the first layer contains 128 neurons, the second layer – 10 neurons, with ReLU and Softmax activation functions, respectively.

2. The model was trained using the error backpropagation algorithm, which was not explicitly specified, with an overall accuracy of 0.97.

3. The values of the loss and accuracy functions for the training and test data sets were calculated. It is concluded that the accuracy function for both sets increases during training, and the loss function for these sets decreases. There is no effect of model retraining. Plots of precision and loss functions for the training and test samples were constructed. After 20 training epochs, the accuracy for the training data set is 0.9808 and for the test data set is 0.976. Also, the loss function for the training data set is 0.0704 and for the test data set is 0.105.

4. The performance of the model was evaluated. The report confirms that the model is effectively trained on the training sample and has high accuracy on the test sample. The model has high accuracy for all classes from 0.94 to 0.99, completeness for all classes from 0.94 to 0.99. Individual and average $F1$ values are also high, indicating balanced precision and recall. The average values of *macro avg* and *weighted avg* are high, indicating the stability of the model across the entire data set.

5. The results of the model are visualized; they demonstrate that the model correctly recognizes even vaguely written numbers.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Funding

The study was conducted without financial support.

Data availability

The manuscript has associated data in the data warehouse.

Use of artificial intelligence

The authors used artificial intelligence technologies within acceptable limits to provide their own verified data, which is described in the research methodology section.

References

1. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A. (2014). Return of the Devil in the Details: Delving Deep into Convolutional Nets. Proceedings of the British Machine Vision Conference 2014. doi: https://doi.org/10.5244/c.28.6
2. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S. et al. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115 (3), 211–252. doi: https://doi.org/10.1007/s11263-015-0816-y
3. Tuba, E., Tuba, M., Simian, D. (2016). Handwritten digit recognition by support vector machine optimized by bat algorithm. WSCG 2016 - 24th Conference on Computer Graphics, Visualization and Computer Vision 2016, 369–376. Available at: https://dspace5.zcu.cz/bitstream/11025/29725/1/Tuba.pdf

4. Elleuch, M., Maalej, R., Kherallah, M. (2016). A New Design Based-SVM of the CNN Classifier Architecture with Dropout for Offline Arabic Handwritten Recognition. Procedia Computer Science, 80, 1712–1723. doi: https://doi.org/10.1016/j.procs.2016.05.512
5. Reshma, A. J., James, J. J., Kavya, M., Saravanan, M. (2016). An overview of character recognition focused on offline handwriting. ARPN Journal of Engineering and Applied Sciences, 11 (15), 9372–9378. Available at: http://www.arpnjournals.org/jeas/research_papers/rp_2016/jeas_0816_4774.pdf
6. Alom, M. Z., Sidike, P., Taha, T. M., Asari, V. K. (2017). Handwritten bangla digit recognition using deep learning. arXiv. doi: https://doi.org/10.48550/arXiv.1705.02680
7. Maitra, D. S., Bhattacharya, U., Parui, S. K. (2015). CNN based common approach to handwritten character recognition of multiple scripts. 2015 13th International Conference on Document Analysis and Recognition (ICDAR). doi: https://doi.org/10.1109/icdar.2015.7333916
8. Glauner, P. O. (2015). Comparison of training methods for deep neural networks. arXiv. doi: https://doi.org/10.48550/arXiv.1504.06825
9. Guerra, L., McGarry, L. M., Robles, V., Bielza, C., Larra aga, P., Yuste, R. (2010). Comparison between supervised and unsupervised classifications of neuronal cell types: A case study. Developmental Neurobiology, 71 (1), 71–82. doi: https://doi.org/10.1002/dneu.20809
10. Stenin, A., Pasko, V., Soldatova, M., Drozdovich, I. (2022). Recognition of handwritten numbers on the basis of convolutional neural networks. Adaptive systems of automatic control, 2 (41), 39–44. Available at: http://asac.kpi.ua/article/view/271337/
11. Korotun, O. V., Marchuk, H. V., Marchuk, D. K., Talaver, O. V. (2020). Systema rozpiznavannia rukopysnykh tsyfr z otsinkoiu yakosti. Tekhnichna inzheneriya, 1 (85), 135–146. doi: https://doi.org/10.26642/ten-2020-1(85)-135-146
12. Kraskovska, A. O., Filimonova, T. O. (2023). Rozrobka arkhitektury zghortkovoi neironnoi merezhi dlia rozpiznavannia rukopysnykh tsyfr. Zbirnyk tez XX mizhnarodnoi naukovo-praktychnoi konferentsiyi «Matematychne na prohramne zabezpechennia intelektualnykh system. MPZIS – 2023». Dnipro, 165–167.
13. El-Sawy, A., EL-Bakry, H., Loey, M. (2016). CNN for Handwritten Arabic Digits Recognition Based on LeNet-5. Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2016, 566–575. doi: https://doi.org/10.1007/978-3-319-48308-5_54
14. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86 (11), 2278–2324. doi: https://doi.org/10.1109/5.726791
15. Whittington, J. C. R., Bogacz, R. (2019). Theories of Error Back-Propagation in the Brain. Trends in Cognitive Sciences, 23 (3), 235–250. doi: https://doi.org/10.1016/j.tics.2018.12.005
16. Whittington, J. C. R., Bogacz, R. (2017). An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. Neural Computation, 29 (5), 1229–1262. doi: https://doi.org/10.1162/neco_a_00949
17. Layer activation functions. Available at: https://keras.io/api/layers/activations/
18. Datasets. MNIST. TensorFlow. Available at: https://www.tensorflow.org/datasets/catalog/mnist
19. Podoliak, B. Yu., Filimonova, T. O. (2023). Rozrobka avtokoduvalnyka dlia rozpiznavannia rukopysnykh tsyfr. Zbirnyk tez XX mizhnarodnoi naukovo-praktychnoi konferentsiyi «Matematychne na prohramne zabezpechennia intelektualnykh system. MPZIS – 2023». Dnipro, 243–244.