Classification of Written Numbers using 2D Convolutional Neural Network

1st Matúš Ďurovič
Institute of Computer Science and Mathematics
FEI of STU in Bratislava
Bratislava, Slovakia
xdurovicm@stuba.sk

2nd Matúš Vaňo
Institute of Computer Science and Mathematics
FEI of STU in Bratislava
Bratislava, Slovakia
xvanom@stuba.sk

***Abstract*—This article introduces a configurable system for building CNNs designed for grayscale image classification. The network architecture and training settings are defined via a JSON file, allowing full control over the layers and parameters without modifying the code. The system was tested on MNIST dataset using two CNN configurations to classify handwritten numbers. Both achieved satisfactory accuracy. The modular design makes it suitable for rapid prototyping and experimentation in computer vision applications.**

***Index Terms*—neural networks, convolutional neural networks, machine learning, image recognition, handwritten digits, parameter tuning**

I. Introduction

In the last decade, Convolutional Neural Networks (CNN), became a standard approach in the area of computer vision, especially in recognition of objects, image classification, or feature detection. Their ability to extract dependencies in space with filters significantly outperforms other known methods of feature extraction. Usually, these solutions are proposed for the whole spectrum of visible light (RGB), but for many other approaches, like text recognition or medical picture processing, it is necessary to use grayscale images.

In this article, we introduce a parameterizable system for designing your own 2D CNN designated for the classification of grayscale pictures. The main focus is to create a specific model that meets the requirements of a given problem, including setting the size of the convolution and dense layers, the size of the kernels, and the options for optimization of the model. The system was designed with simplicity of deployment and the possibility of analyzing loss development and accuracy of classification for given classes.

The proposed solution was tested on the MNIST open source database, where our results show good accuracy, even on smaller models. We show the impact of architectonic changes on the model results.

In the next parts of this article, we discuss related work in CNNs and grayscale picture classification (Section II), and in Section III we describe the approach and network architecture, we were using. Section IV discusses the results achieved, including the accuracy of a given model reached and the visualization of the loss. We discuss the results in Section V and the limits of the designed system. Finally, in Section VI we conclude our achievements and propose other work that can be done to the system in the future.

II. Related Work

Active research is underway on handwritten text recognition around the world in business or in the academic environment. They are using similar and other methods to classify text.

In [1] were using Adaptive Residual Attention Network (ARAN), where they were finetuning the model with Improved Energy Valley Optimization Algorithm (IEVOA) for enhancing the recognition performance.

Article [2] is studying recognition on Tamil handwriting and shape recognition. The project confronts shape variations and a vast character inventory, utilizing a meticulously categorized dataset of approximately 91,000 samples across 156 distinct character classes developed by MNIST.

Moreover, there are many projects going on with combining neural networks with other approaches like KNN algorithm, SVM, and PCA. In [3] works on classifying the digits using a combination of KNN, PCA, and SVM. They trained their model on the MNIST dataset.

III. Methodology

In this section, we describe the architecture of proposed system, the pre-processing of input data, and training of the model. This system is using dynamic model construction, based on given JSON file, taken from input. As the output, the system returns the best trained model, figures of development of loss and accuracy, with confusion matrix.

A. Dataset and data pre-processing

In the experiment, we used an open source dataset called MNIST, which is a part of the keras and tensorflow libraries. This dataset contains 70000 images, and these are divided into training set (60000 images) and a test set (10000 images). Each image has dimensions of 28x28 pixels and they are grayscale spectrum.

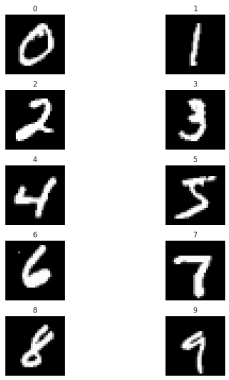

Fig. 1. Examples of images with assigned classes

As a pre-processing method, we used normalization, where each image is in the range [0,1]. The normalization is given by this formula:

$$I_{norm}(x,y) = \frac{I(x,y)}{255} \tag{1}$$

where $I(x,y)$ represents the value of the pixel in the position (x,y).

B. Neural network specification

The architecture of the model is given by the input JSON file, which contains the list of layers, with their exact order. Each layer is described by the field *"type"* and other parameters, such as *"filters", "kernel size", "activation"*, etc. Image 2 shows this file and its structure.

```
{
  "layers": [
    {"type": "Conv2D", "filters": 32, "kernel_size": [3, 3], "activation": "relu", "input_shape": [64, 64, 1]},
    {"type": "MaxPooling2D", "pool_size": [2, 2]},
    {"type": "Conv2D", "filters": 16, "kernel_size": [3, 3], "activation": "relu"},
    {"type": "MaxPooling2D", "pool_size": [2, 2]},
    {"type": "Flatten"},
    {"type": "Dense", "units": 64, "activation": "relu"},
    {"type": "Dropout", "rate": 0.4},
    {"type": "Dense", "units": 10, "activation": "softmax"}
  ],
  "optimizer": "Adam",
  "learning_rate": 0.0001,
  "epochs": 15
}
```

Fig. 2. Example of the input JSON file

C. Model construction

After successful loading of the input file, the construction of our model will begin. Each layer is created dynamically on the basis of the type. We use the code shown in Figure 3.

```
model = Sequential()
for layer_config in config["layers"]:
    layer_type = layer_config.pop("type")
    layer_class = getattr(layers, layer_type)
    model.add(layer_class(**layer_config))
```

Fig. 3. Program that constructs the neural network model

The convolutional layer applies a set of filters to the input data. The output feature map O is computed by:

$$O_{i,j} = \sigma(\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} W_{m,n} * I_{i+m,j+n} + b) \tag{2}$$

where W are the filter weights, I is the input feature map, b is the bias term and σ is the activation function.

D. Loss functions

in supervised classification, the loss function determines the difference between the actual labels and the predicted class. For our system, we have chosen the categorical cross-entropy, because it is a standard for classification wit one-hot encoded targets. The categorical cross-entropy loss $\mathcal{L}$ is defined as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i log(\hat{x}) \tag{3}$$

where y_i is the true label and $\hat{x}_i$ is the predicted probability for class i and C is the amount of classes. This loss function penalizes confident but incorrect guesses more strongly and encourages the model to assign high probability to the correct class.

1) Alternative loss functions:

- Binary Cross-Entropy: It is used in binary classification or multi-label tasks. Defined as:

$$\mathcal{L} = -\left[ylog(\hat{y}) + (1-y)log(1-\hat{y})\right] \tag{4}$$

- Mean Squared Error: Mostly used in regression tasks, defined as:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2 \tag{5}$$

- Sparse Cross-Entropy: Variant of categorical cross-entropy used when labels are integer based instead of one-hot.

E. Model optimizer

Optimizers update the model's trainable parameters in order to minimize the loss function. The proposed system uses the Adam optimizer, which is an optimization method that maintains moving averages of squared gradients and gradients.

$$\Theta_{t+1} = \Theta_t - \eta * \frac{\hat{m_i}}{\sqrt{\hat{v_t}} + \epsilon} \tag{6}$$

where Θ_t are model parameters, η is the learning rate, $\hat{v_t}$ and $\hat{m_t}$ are bias-corrected estimates of the gradient's first and second moments.

1) Other optimizers:

- SGD (Stochastic Gradient Descent): A simple and powerful optimizer that can be enhanced with momentum or learning rate schedules.
- RMSprop: Maintains an exponentially decaying average of squared gradients. It is suitable for non-stationary objectives.
- Adagrad: Adapts the learning rate based on the magnitudes of the previous gradients. Useful on sparse data.

F. Model compilation

After the model construction, it is compiling using the optimizer, loss function, and metrics defined in the input configuration.

G. Outputs

After training, the system automatically saves the best-performing model. It also generates visualizations of the training process (loss and accuracy curves) and computes a confusion matrix to evaluate classification performance per class.

H. System advantages

The system offers:

- Modularity: each aspect of the architecture, can be defined separately, outside of the source code
- Reproducibility: experiments can be reproduced thanks to JSON configurations
- Experimental flexibility: change of architecture is easy, and fast. You can change only the configuration file.

IV. Experiments and Results

In this section, we present the experimental setup, evaluation metrics, and the results obtained from training and testing of the dynamically constructed CNN model.

A. Setup of experiments

The experiments are carried out on the MNIST dataset, which consists of 70000 images. The input images were normalized to the range $[0, 1]$ as we described in Section III-A.

The models were initialized using the JSON configuration file, and in both experiments they used Adam optimizer. For evaluation metrics, we used categorical cross-entropy and categorical accuracy.

B. Experiments

1) Experiment 1: In table I is shown the network configuration and in II the training configuration for the first experiment.

TABLE I
CONFIGURATION OF THE CNN MODEL USED IN EXPERIMENT 1

Layer	Type	Parameters
Input	Input Layer	$28 \times 28 \times 1$
1	Conv2D	64 filters, 3×3 kernel, ReLU, L2 regularization $\lambda = 0.005$
2	MaxPooling2D	2×2 pool size
3	Conv2D	32 filters, 3×3 kernel, ReLU, L2 regularization $\lambda = 0.005$
4	MaxPooling2D	2×2 pool size
5	Conv2D	32 filters, 3×3 kernel, ReLU, L2 regularization $\lambda = 0.005$
6	BatchNormalization	—
7	Activation	ReLU
8	Flatten	—
9	Dense	128 units, ReLU
10	Dropout	rate = 0.2
11	Dense	128 units, ReLU
12	Dropout	rate = 0.2
Output	Dense	10 units, softmax

TABLE II
TRAINING PARAMETERS USED IN EXPERIMENT 1

Parameter	Value
Optimizer	Adam
Learning rate	0.0001
Epochs	70
Early stopping	Patience = 5
Loss function	Categorical cross-entropy

Table III shows the loss and accuracy of Experiment 1, Figures 4 show its training process, and Figure 5 shows the confusion matrices for training and test sets in Experiment 1.

TABLE III
LOSS AND ACCURACY RESULTS OF FIRST EXPERIMENT

Training loss	Training accuracy	Testing loss	Testing accuracy
0.036	0.993	0.061	0.988

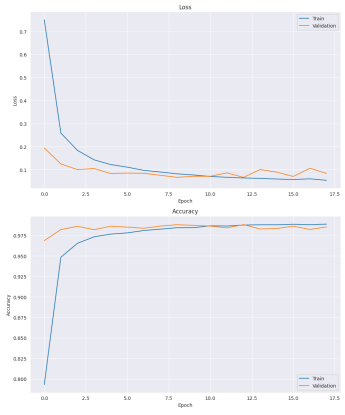

Fig. 4. Development of loss and accuracy during training of experiment 1

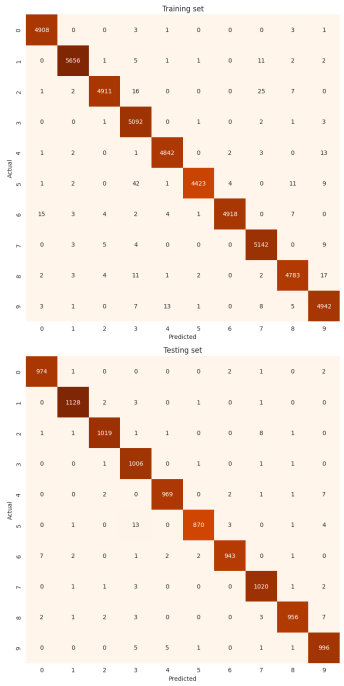

Fig. 5. Confusion matrix for the in experiment 1

2) Experiment 2: Table IV shows the network configuration and table V shows the training configuration for the second experiment.

TABLE IV
CONFIGURATION OF THE CNN MODEL USED IN EXPERIMENT 2

Layer	Type	Parameters
Input	Input Layer	$28 \times 28 \times 1$ (grayscale image)
1	Conv2D	64 filters, 3×3 kernel, ReLU, L2 regularization $\lambda = 0.005$
2	BatchNormalization	—
3	Activation	ReLU
4	MaxPooling2D	2×2 pool size
5	Conv2D	32 filters, 3×3 kernel, ReLU, L2 regularization $\lambda = 0.005$
6	BatchNormalization	—
7	Activation	ReLU
8	MaxPooling2D	2×2 pool size
9	Conv2D	32 filters, 3×3 kernel, ReLU, L2 regularization $\lambda = 0.005$
10	BatchNormalization	—
11	Activation	ReLU
12	MaxPooling2D	2×2 pool size
13	Flatten	—
14	Dense	128 units, ReLU
15	Dropout	rate = 0.3
16	Dense	64 units, ReLU
17	Dropout	rate = 0.3
Output	Dense	10 units, softmax

TABLE V
TRAINING PARAMETERS USED IN EXPERIMENT 2

Parameter	Value
Optimizer	Adam
Learning rate	0.0005
Epochs	70
Loss function	Categorical cross-entropy
Callbacks	EarlyStopping (patience=5), ReduceLROnPlateau (patience=3, factor=0.5)

In table VI we can see the loss and accuracy of Experiment 2, Figure 6 shows its training process, and Figure 7 shows the confusion matrices for training and test sets in Experiment 2.

TABLE VI
LOSS AND ACCURACY RESULTS OF SECOND EXPERIMENT

Training loss	Training accuracy	Testing loss	Testing accuracy
0.036	0.998	0.079	0.989

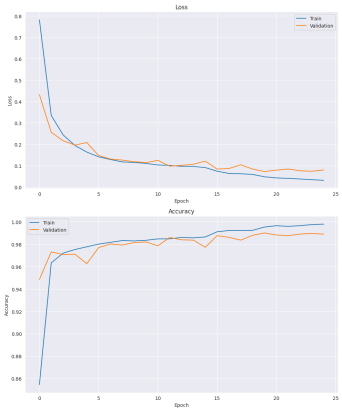

Fig. 6. Development of loss and accuracy during training of Experiment 2

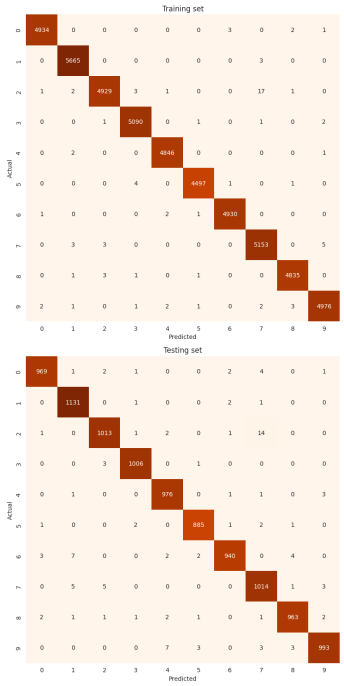

Fig. 7. Confusion matrix for the in Experiment 2

V. Discussion

The experiments carried out demonstrate the effectiveness of dynamically constructed CNNs for grayscale classification. Both tested architectures achieved high accuracy on the MNIST dataset, confirming the sustainability of the proposed system for such tasks.

Compared with our experiments, Experiment 2 (Section IV-B2) outperformed 0.1% the baseline configuration of Experiment 1 (Section IV-B1). For Experiment IV-B1 In Figure 5, we see well-colored diagonals of the confusion matrices, which indicate good predictions. There are no signs of overfitting or underfitting. We can see that there was a problem that the model falsely predicted class 3, when it was class 5 in 13 cases. This can be caused by the same position of the lower arch in these numbers. In the second experiment, we do not see this phenomenon to occur, but in both cases we see present the false predicament of class 2 when class 7 was given.

The inclusion of dropout layers in both experiments helped to build a better generalization ability of our models. It is indicated by smooth loss and accuracy curves.

We observe that even after adding one more convolutional layer, the accuracy of the model did not increase that much. For a more accurate model, for example with accuracy in the high 99% we would need a lot of more trainable parameters than we have in our model. In our case, the better model has 125400 parameters and the training lasted about 1 hour and 15 minutes, while the simpler model has 99400 and the training lasted about a half hour. This shows how increasing the complexity of the network significantly increases the training time.

In general, the results validate the flexibility, modularity, and effectiveness of the proposed dynamic CNN construction approach for grayscale image classification.

VI. Conclusion and Future Work

In this paper, we presented the flexible and dynamically configurable system for building convolutional neural networks, tailored for grayscale classification tasks. The system enables users to define the entire network architecture, including the convolutional, pooling, normalization, dense, and dropout layers, through an external JSON configuration file. This modular approach ensures high reproducibility and ease of experimentation without the need to alter the source code. The system also supports the specification of training parameters, such as optimizer, loss function, learning rates, and callbacks. This further increases flexibility and opens the door to efficient hyperparameter tuning.

We evaluated the proposed solution on the MNIST dataset using two different CNN configurations. Both experiments achieved high classification accuracy, with the second, deeper model reaching a test accuracy of $98,9\%$.

In future work, we plan to extend the capabilities of the system by:

- Adding support to more types of layer, such as LSTM and GlobalAveragePooling.
- Including data augmentation techniques to improve generalization on more complex datasets
- Support of logging and monitoring of training process via web interface.

In summary, the system offers a solid and extensible base for conducting experiments in deep learning, especially in educational and research contexts, where rapid prototyping is essential.

References

[1] S. Rao N, N. Kennedy Babu C, “Adaptive Residual Attention Network for Handwritten Character and Digits Recognition with Improved Energy Valley Optimizer Algorithm” 2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS).

[2] C. Varshini, S. Yogeshwaran and V. Mekala, ”Tamil and English Handwritten Character Segmentation and Recognition Using Deep Learning,”, 2024 International Conference on Communication, Computing and Internet of Things (IC3IoT). Chennai, India, 2024, pp. 1-5.

[3] M. Pan and J. Lin, ”Research on Handwritten Digit Recognition Based on Intelligent Algorithms,” 2024 6th International Conference on Internet of Things, Automation and Artificial Intelligence (IoTAAI), Guangzhou, China, 2024, pp. 614-617

This page is intentionally left blank.