УДК 004.855.5+004.932.2

D.V. Maliovanyi
I.V. Bogach

INVESTIGATION OF IMPACT OF CONVOLUTIONS ON PREDICTION PROBABILITY ON EXAMPLE OF HUMAN GESTURES LANGUAGE DATASET

Вінницький національний технічний університет

Анотація:
Дану доповідь присвячено дослідженню та порівнянню кореляції ефективності прогнозування нейронної мережі та кількістю шарів згортки та їх структурою. За суб'єкт дослідження взято нейронну мережу, що розпізнає зображення мови жестів.
Ключові слова: нейронна мережа, шар згортки, фільтр, розпізнавання, навчання моделі, датасет, точність на тренувальних даних, точність на перевіркових даних, функція втрат, функція розподілу (класифікації).

Abstract:
This report is dedicated to cover the investigation and comparison of neural network prediction efficiency correlation with convolution layers' quantity and structure. A neural network recognizing gestures language is taken as a subject of this research.
Keywords: neural network, convolution layer, filter, recognition, model training (fitting), dataset, training accuracy, validation accuracy, loss function, distribution (classification) function.

Introduction

The purpose of this paper is to investigate convolutional neural network, which includes deep neural network (DNN further), preceded by convolutional layers, efficiency in image recognition accuracy. Also it is aimed to figure out impact can be caused on input recognition accuracy by different number of convolution layers' amount and its' structure as well. The neural network observation is performed on handles gestures language classification, so it classifies gestures into 24 classes (for number of letters in the English alphabet except J and Z, since they require motion).

Neural Network General Structure

The neural network consists of image preprocessing convolutional layer, as well as pooling layers, deep neural network hidden dense layer and a classification output layer. The dataset used for this task is a MNIST gestures dataset originated from Kaggle competition. It contains around 28'000 training images and slightly more than 7'000 for validation [1]. Data is passed to network via image data generator in batches of 128pc. Every image is grayscale and has dimension 28x28. Some values will be predefined while some will be adjustable.

Convolution Layer

The first section of this is a convolutional layers' sequence. The convolutions are applied to detect features in the image due to filters. In simple words, filter is a matrix of odd numbers, i.e. 3x3, 5x5 etc. There is a plenty of such filters differing in sizes and features they detect, but this paper is not dedicated to get through it, though enough well-written explanations and examples can be found [2]. Also in this chain a pooling layers are present, which condense input data while keep the most essential features. The principle of this process is quite simple. It lays in division of all the image I(MxN) into small segments S_{ij}(K,L), i=1,…, $\frac{M}{K}$, j=1,…, $\frac{N}{L}$. The resulting image will be I*($\frac{M}{K}$, $\frac{N}{L}$), and will consist of the highest values from the segments. E.g. the S_{11} one has such values: $\begin{bmatrix} 15 & 2 \\ 27 & 120 \end{bmatrix}$. As has been mentioned before, the output will be 120 since it's the greatest number. In the resulting matrix this element has (1,1) position. Also it's important to note some essential properties. Firstly, since the data is a raw image matrix, particular values are in 1 to 256 (in most computer languages 0 to 255). If operations are performed on normalized data, the values will be float 0 to 1. Secondly, every convolution will crop image. Convolution of size (3x3) will result in image (50x50) will become (48x48) since the edges doesn't have full neighbour elements matrix (i.e. the filter of size (3x3) on element at (0,0) should contain

$$\begin{bmatrix} (-1,-1) & (-1,0) & (-1,1) \\ (0,-1) & (0,0) & (0,1) \\ (1,-1) & (1,0) & (1,1) \end{bmatrix}$$

And it is obvious elements with negative indices can't be allocated. As it has been mentioned, this layer is a subject to change to obtain different results.

Hidden DNN Layer

This layer plays an important role in model improvement. Generally speaking, the designation of any neural network is to define the target function, which would for every argument x set in relation target value y. In this case x is represented by a feature vector so it is better to display it as

$$\overrightarrow{X_n} = \begin{bmatrix} 1 \\ \dots \\ n \end{bmatrix}. \tag{1}$$

It is also called feature vector, where *n* stands for *feature number*. The most simple function can be written in form

$$y = b_1x_1 + b_2x_2 + \dots + b_nx_n + b_k, \tag{2}$$

or

$$y = \sum_{i=1}^{n} b_ix_i + b_k; \tag{3}$$

It is also called linear regression and shows correlation between feature vector and target value on defined definite spaces of both features and targets [5]. So let the E(k,n) be the feature vector space and D(m) be the target space, that is all possible target values. Both spaces can be represented as sets. Every X_n must be put in relation to some y. It is easy to prove that most cases won't have strict, i.e. functional relations, and every value would have certain individual deviation from the regression line. To handle this issues, loss function is applied. It shows the mean deviation among the sample and helps to understand how to adjust b coefficients in regression equation. It is clear the less the loss the better the target function so the aim is to minimize it. The most common loss function are Mean Squared Error (MSE):

$$MSE = \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{m}, \tag{4}$$

where y_i is a predicted value, and $\hat{y}_i$ is a target value, $\hat{y}_i \in D(m)$; and Cross Entropy Loss (CEL):

$$CEL = -[y_i \ln(\hat{y}_i) + (1 - y_i)\ln(1 - \hat{y}_i)]. \tag{5}$$

There is quite a variety of loss functions as well as target functions, and one of the important things is to define which one is of best use in particular case.

This hidden layer is designated to minimize the loss function. It is should be noted the hidden layer consists in this case of single dense layer of neurons with 512 neurons into it. The loss function applied in this case is decided to be the CEL, since the input data is normalized, and this function is very good at normalized data. Also, classification to be done is among 24 possible classes, and CEL is great in penalizing probabilities which are confident enough but wrong [3].

Output Layer

Once the target function is calculated, it is passed to output layer, which defines which of the given 24 classes suits the most for the particular feature vector. Since neural network operates solely in the probabilistic field, and the output should be in this case represented by only one class, the classification function is applied. In particular, the output layer will be dense layer of 24 neurons, accordingly to class number. The activation, or, in other words, classification function there is a Softmax, also known as normalized exponential function will be applied. It is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. It is used in this case to represent categorical distribution of *k* probabilities $p_1, p_2, \dots, p_i, \dots, p_k$ [5] meanwhile

$$\sum_{i=1}^{k} p_i = 1; \tag{6}$$

So the function graph will take the form as shown on figure 1.

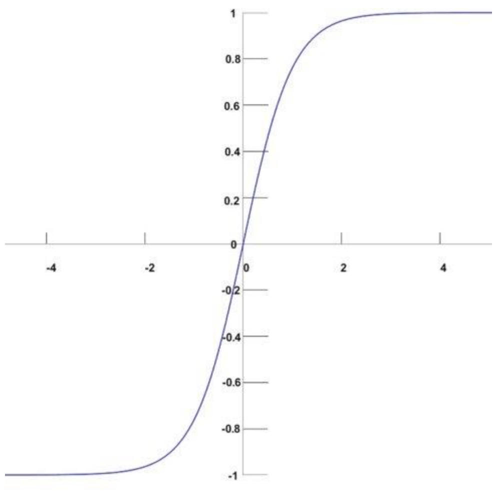


Figure 1. Softmax activation function in the Euclidean plane

Practical Research

All the parameters have been divided into static and adjustable. To static were included epochs number, batch size, optimizer class, activation and loss functions, amount of DNN layers and neurons in them and steps per epoch. There, epochs number stands for amount of cycles fill be performed on model training and is set to 15; batch size is a number of image samples will be processed and their features included into loss function before an actual adjustment is done and further processing is done with adjusted parameters and is set to 128; steps per epoch means amount of batches to be processed during each epoch and is set to 215, since training dataset contains 27455 images, and 215*128=27520, which covers all the dataset so dataset fully processed on each epoch; optimizer decided to use for target function adjustment is a RMSprop with learning rate 0,001 [4]; activation function is softmax, loss function is the CEL; there will be one dense layer with 512 units in it.

As the lower bound, the result without any convolution will be set.

```
layers.Conv2D(1,(1,1),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.Flatten(),
layers.Dense(512,activation=tf.nn.relu),
layers.Dense(25,activation=tf.nn.softmax)
```

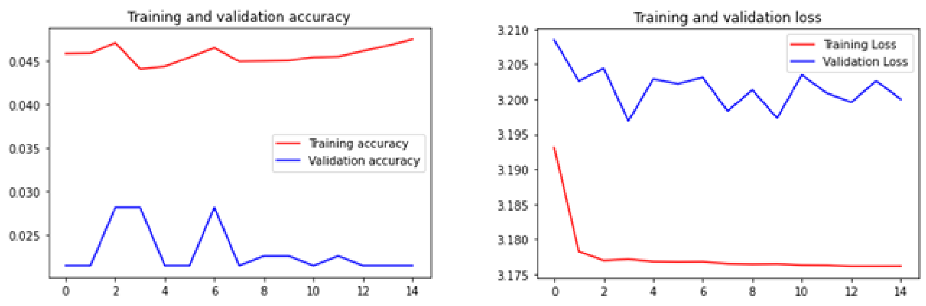


Figure 2. Launch without any convolutions

Then, a set of launches with different settings has been performed:

```
layers.Conv2D(8,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.MaxPooling2D(2,2),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.Conv2D(32,(3,3),activation=tf.nn.relu),
layers.MaxPooling2D(2,2),
layers.Flatten(),
```

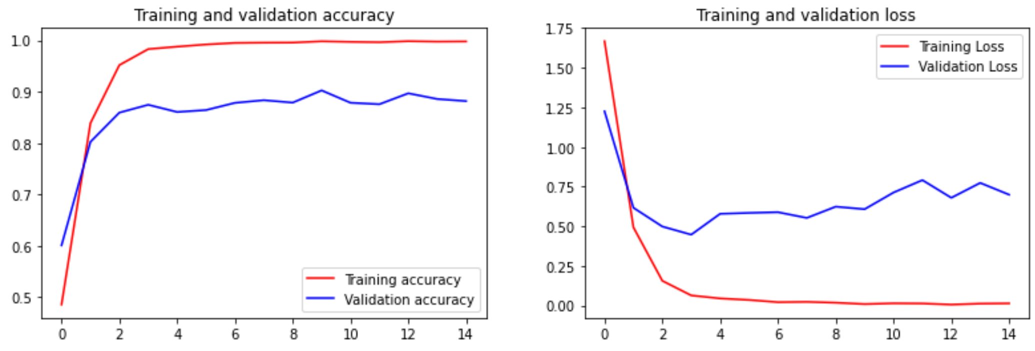


Figure 3. 4 convolution layers and 2 pooling layers

```
layers.Conv2D(8,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.Conv2D(32,(3,3),activation=tf.nn.relu),
layers.Flatten(),
```

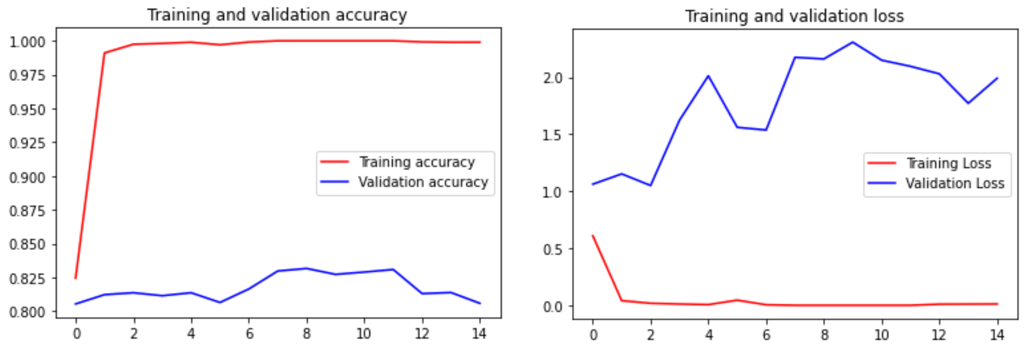


Figure 4. 4 convolutional layers

```
layers.Conv2D(8,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.Conv2D(32,(3,3),activation=tf.nn.relu),
layers.MaxPooling2D(2,2),
```

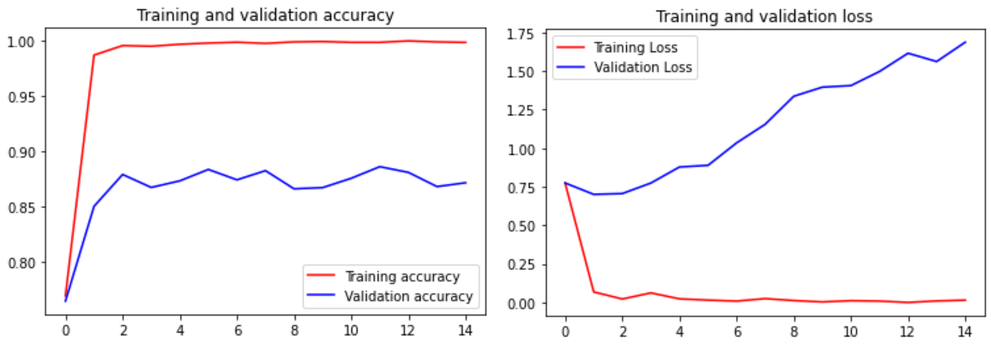
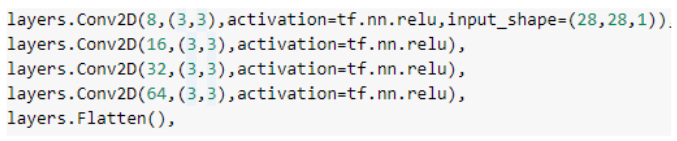


Figure 5. 4 convolution layers and one pooling layer



```
layers.Conv2D(8,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.Conv2D(32,(3,3),activation=tf.nn.relu),
layers.Conv2D(64,(3,3),activation=tf.nn.relu),
layers.Flatten(),
```

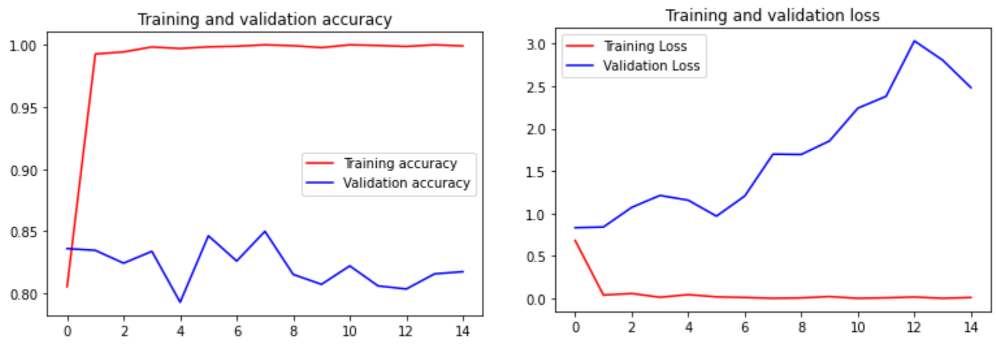
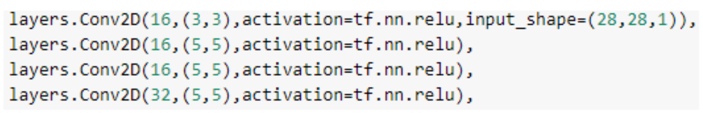


Figure 6. 4 convolution layers, each produces doubled amount convolutions as preceding



```
layers.Conv2D(16,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.Conv2D(16,(5,5),activation=tf.nn.relu),
layers.Conv2D(16,(5,5),activation=tf.nn.relu),
layers.Conv2D(32,(5,5),activation=tf.nn.relu),
```

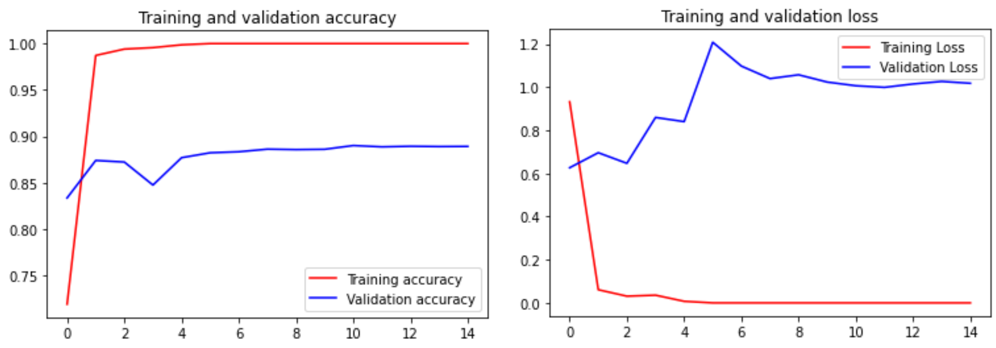
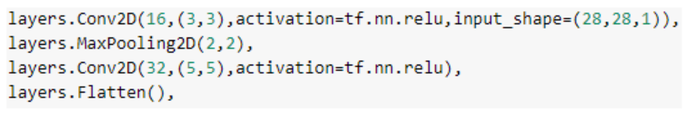


Figure 7. Another example of 4-layer convolution configuration



```
layers.Conv2D(16,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.MaxPooling2D(2,2),
layers.Conv2D(32,(5,5),activation=tf.nn.relu),
layers.Flatten(),
```

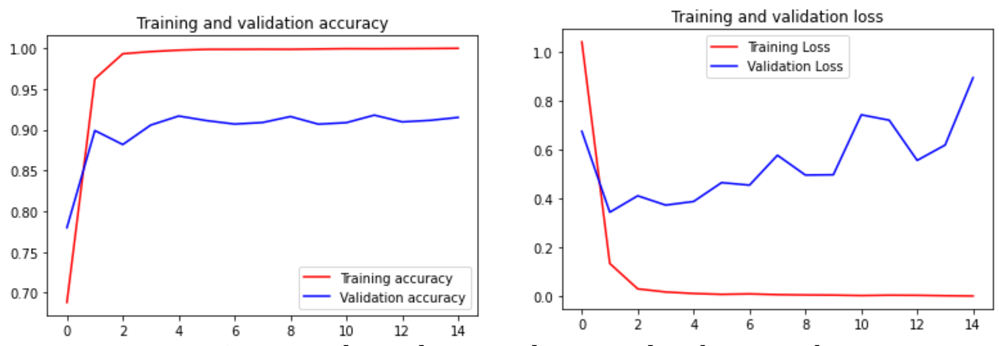
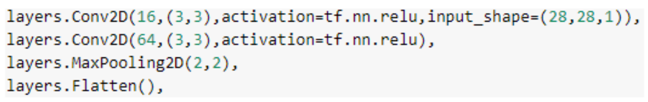


Figure 8. 2 convolution layers and max pooling between them



```
layers.Conv2D(16,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.Conv2D(64,(3,3),activation=tf.nn.relu),
layers.MaxPooling2D(2,2),
layers.Flatten(),
```

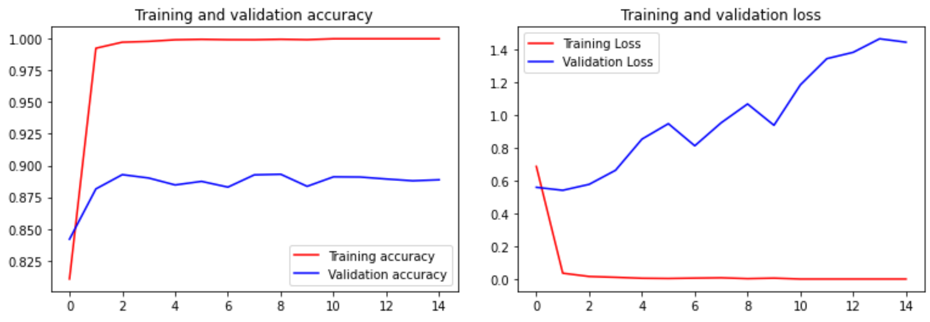
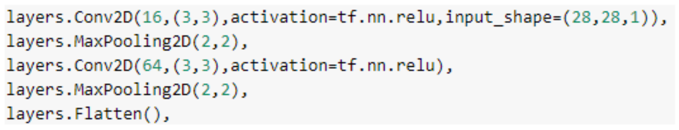


Figure 9. 2 convolution layers followed by max pooling one



```
layers.Conv2D(16,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.MaxPooling2D(2,2),
layers.Conv2D(64,(3,3),activation=tf.nn.relu),
layers.MaxPooling2D(2,2),
layers.Flatten(),
```

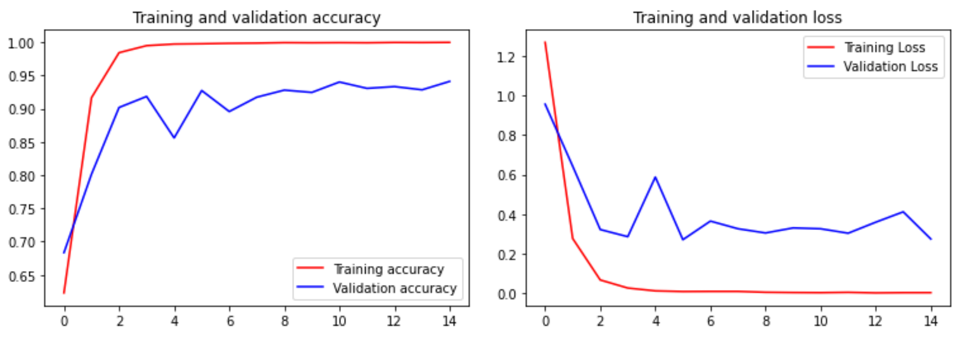
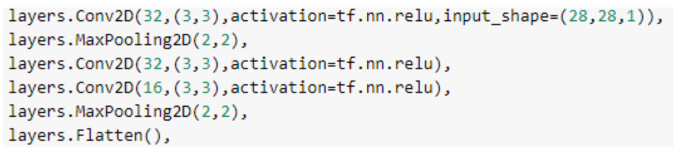


Figure 10. Modification of 2 convolution and 2 pooling layers interchanging each other



```
layers.Conv2D(32,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.MaxPooling2D(2,2),
layers.Conv2D(32,(3,3),activation=tf.nn.relu),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.MaxPooling2D(2,2),
layers.Flatten(),
```

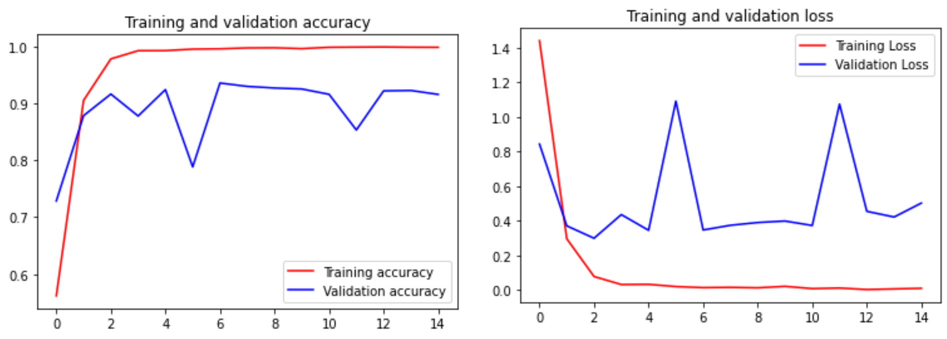


Figure 11. 3 convolution layers and 2 pooling ones

```
layers.Conv2D(32,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.MaxPooling2D(2,2),
layers.Conv2D(32,(3,3),activation=tf.nn.relu),
layers.Conv2D(32,(3,3),activation=tf.nn.relu),
layers.MaxPooling2D(2,2),
layers.Flatten(),
```

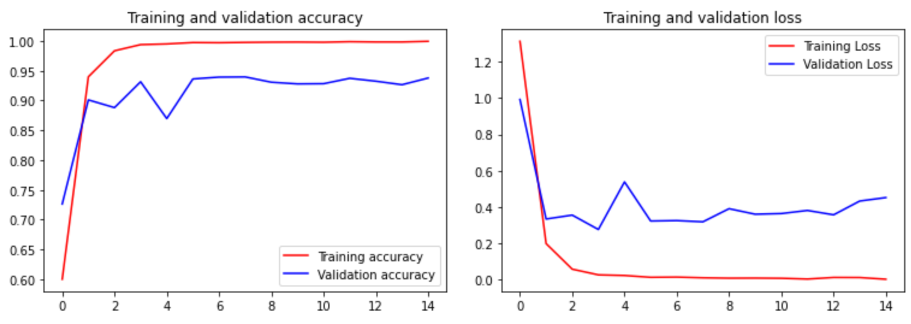


Figure 12. Adjusted previous model

```
layers.Conv2D(16,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.MaxPooling2D(2,2),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.MaxPooling2D(2,2),
layers.Flatten(),
```

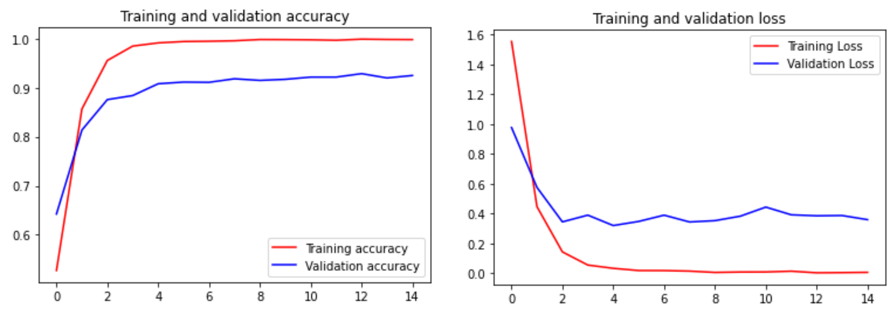


Figure 13. Slightly more adjustments

```
layers.Conv2D(16,(3,3),activation=tf.nn.relu,input_shape=(28,28,1)),
layers.MaxPooling2D(2,2),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.MaxPooling2D(2,2),
layers.Conv2D(16,(3,3),activation=tf.nn.relu),
layers.Flatten(),
```

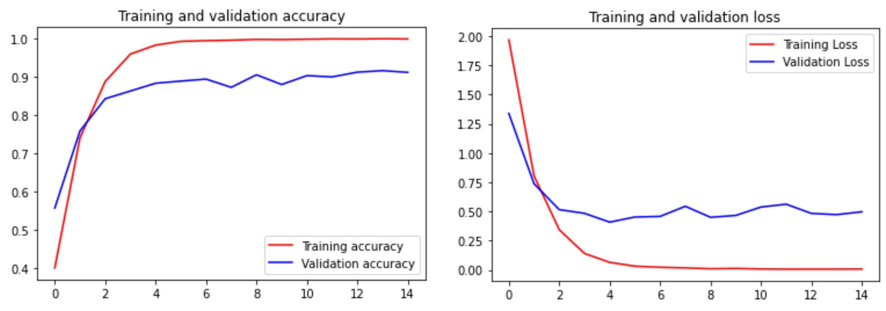


Figure 14. Modification of model from figure 13

As can be observed, more doesn't always mean better in computer vision and machine learning. So acquiring really worth model really requires some effort on tries and errors. On the other hand, there wasn't much space on image preprocessing via convolutions and pooling since input size was only 28x28, and as every convolution or pooling crops a part of image, there were some limitations in possible modification (the input to hidden layer should have contained at least 3x3 pixels). Also it should be noted, the layer called Flatten is used to transform matrix (m, n) into feature vector (m*n). It is present in every experiment and stands for transformation the data into format can be processed by Dense layer. Also an interesting case can be spotted when comparing figures 13 and 14. On the first sight, there is an impression that model from figure 14 performs better. Yes, the accuracy line does slightly better than one from figure 13. But when looking on the loss function comparison, the one from figure 13 is better since it is slightly lower. Generally speaking, it can be concluded that overfitting is performed on most models so probably 15 is too much in this case. On the other hand, the smoother the training, the less it likely to overfit and the more likely to get better results.

USED LITERATURE LIST

1. Michigan Institute of Technology, Drop-In Replacement for MNIST for Hand Gesture Recognition Tasks [Electronic resource] – Electronic data. – Mode of access: https://www.kaggle.com/datamunge/sign-language-mnist/tasks – Title from screen.
2. Lode Vandevenne, Image Filtering. [Electronic resource] – Electronic data. – Mode of access: https://lodev.org/cgtutor/filtering.html – Title from the screen.
3. Common loss functions in machine learning [Electronic resource] – Electronic data. – Mode of access: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23 – Title from screen.
4. A Look at Gradient Descent and RMSprop Optimizers [Electronic resource] – Electronic data. – Mode of access: https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b – Title from the screen.
5. Machine Learning, Tom Mitchell, McGraw Hill, 1997. ISBN 0070428077.

Богач Ілона Віталіївна, кандидат технічних наук, доцент кафедри автоматизації та інтелектуальних інформаційних технологій, Вінницький національний технічний університет, ilona.bogach@gmail.com.
Мальований Дмитро Вадимович, студент групи 1ІСТ-18б, Факультет комп’ютерних систем та автоматики, Вінницький національний технічний університет, dmytro.maliovanyi@gmail.com.

Bogach Ilona Vitaliivna, PhD, Associate Professor of the department of automation and intelligent information technologies, Vinnytsia National Technical University, ilona.bogach@gmail.com.
Maliovanyi Dmytro Vadymovych, the student of 1IST-18B, the faculty of computer systems and automation, Vinnytsia National Technical University, dmytro.maliovanyi@gmail.com.