HELLENIC REPUBLIC
National and Kapodistrian University of Athens

MACHINE LEARNING

2nd Project

Abstract

This document contains the solution for the 2nd Project of the Machine Learning course taken within the SMARTNET program at the National and Kapodistrian University of Athens. The associated code will be sent via email.

Student: Dan Serbanoiu
ask@danserbanoiu.com

Exercise 1:

A.

In a multi layered perceptron, a cost function is defined. Usually $E = \frac{1}{2}\sum_{k\mu}(T_{k\mu} - {y_{k\mu}}^{(R)})^2$ is used, but it's possible to have other types of cost functions. E is the sum of the differences of the desired outputs from the actual outputs, raised to the power of 2. The goal when training a neural network is to minimize that cost function which is a function that has many dimensions and the way to do that is by iteratively incrementing the weights of the network according to the following relations, calculated by differentiating the chosen cost function:

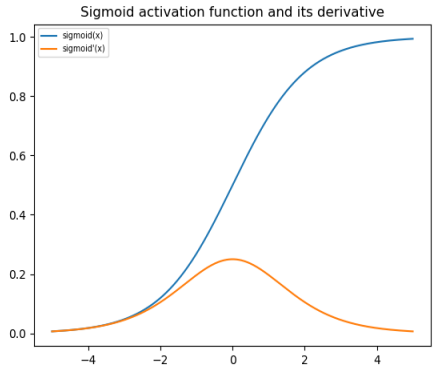
$$\delta {w_{ij}}^{(r)} = -\varepsilon \sum_{\mu} {\delta_{i\mu}}^{(r)} {y_{j\mu}}^{(r-1)}$$

The gradients are calculated with the following formulas, for the last dimension and the r-th dimension respectively:

A) $${\delta_{i\mu}}^{(R)} = f'\left({v_{i\mu}}^{(R)}\right)\left(y_{i\mu} - T_{i\mu}\right)$$

B) $${\delta_{i\mu}}^{(r)} = \sum_{k} {\delta_{k\mu}}^{(r+1)} {w_{ki}}^{(r+1)} f'\left({v_{i\mu}}^{(r)}\right)$$

The previous derivatives are also dependent on the specific activation function chosen to train the network. Let's now look at the general form of the backpropagation equations of MLPs for different activation functions:

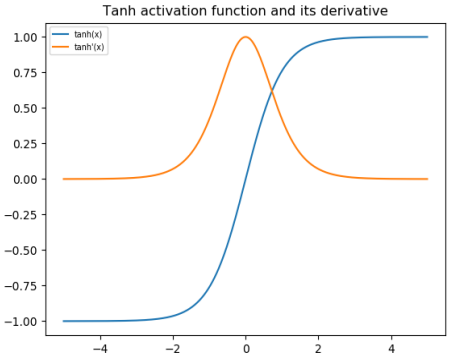


$$f(x) = \frac{1}{1 - e^{-x}}$$
$$f'(x) = f(x)\left(1 - f(x)\right)$$

$$\delta {w_{ij}}^{(R)} = -\varepsilon \sum_{\mu} f({v_{i\mu}}^{(R)})(1 - f\left({v_{i\mu}}^{(R)}\right))\left(y_{i\mu} - T_{i\mu}\right){y_{j\mu}}^{(R-1)}$$

$$\delta {w_{ij}}^{(r)} = -\varepsilon \sum_{\mu}(\sum_{k} {\delta_{k\mu}}^{(r+1)} {w_{ki}}^{(r+1)} f\left({v_{i\mu}}^{(r)}\right)\left(1 - f\left({v_{i\mu}}^{(r)}\right)\right)){y_{j\mu}}^{(r-1)}$$

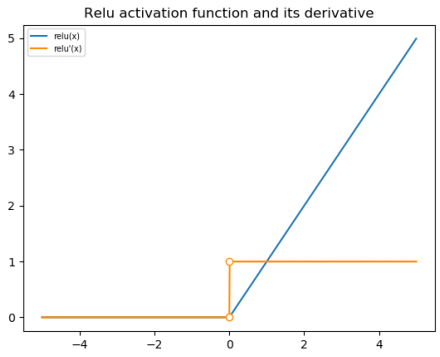
The gradients lie in the interval [0, 1].



$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$
$$f'(x) = 1 - f(x)^2$$

$$\delta {w_{ij}}^{(R)} = -\varepsilon \sum_{\mu}(1 - f\left({v_{i\mu}}^{(R)}\right)^2)\left(y_{i\mu} - T_{i\mu}\right){y_{j\mu}}^{(R-1)}$$
$$\delta {w_{ij}}^{(r)} = -\varepsilon \sum_{\mu}(\sum_{k} {\delta_{k\mu}}^{(r+1)} {w_{ki}}^{(r+1)}(1 - f\left({v_{i\mu}}^{(r)}\right)^2))\, {y_{j\mu}}^{(r-1)}$$

The gradients lie in the interval [0, 1].



$$f(x) = \begin{cases} 0, x \leq 0 \\ x, x > 0 \end{cases}$$
$$f'(x) = \begin{cases} 0, x \leq 0 \\ 1, x > 0 \end{cases}$$

For $x > 0$:

$$\delta w_{ij}{}^{(R)} = -\varepsilon \sum_{\mu} (y_{i\mu} - T_{i\mu}) y_{j\mu}{}^{(R-1)}$$
$$\delta w_{ij}{}^{(r)} = -\varepsilon \sum_{\mu} (\sum_{k} \delta_{k\mu}{}^{(r+1)} w_{ki}{}^{(r+1)}) \, y_{j\mu}{}^{(r-1)}$$

For $x \leq 0$

$$\delta w_{ij}{}^{(R)} = 0$$
$$\delta w_{ij}{}^{(r)} = 0$$

The gradients lie in the interval [0, 1].

B.

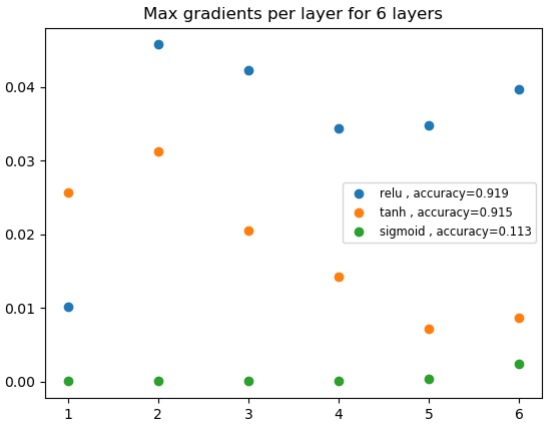
After training a fully connected neural network to recognize handwritten digits with the MNIST dataset, with different activation functions (RELU, Tanh, Sigmoid) and different layers (5, 20, 40) the results are:

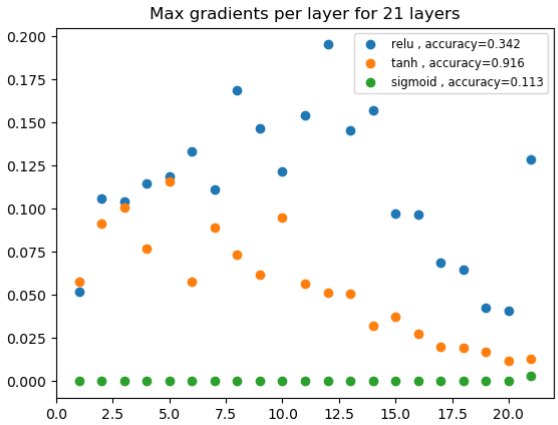
Activation Function	**Layers**	**Accuracy**
RELU	5	93.26%
RELU	20	60.15%
RELU	40	11.35%
Tanh	5	93.27%
Tanh	20	93.73%
Tanh	40	91.58%
Sigmoid	5	11.35%
Sigmoid	20	11.35%
Sigmoid	40	11.35%

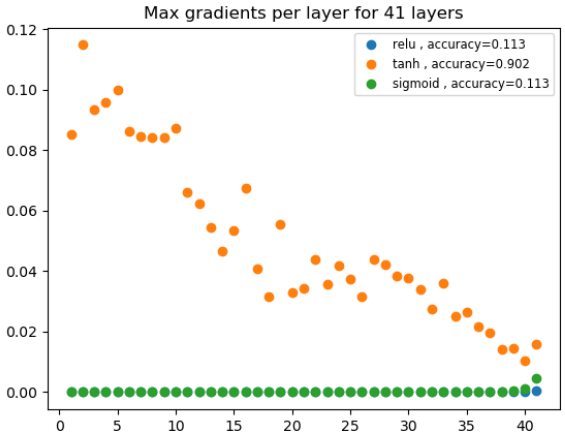
It can be observed that the accuracy of a model depends on the complexity of the network, that is the number of layers, and the activation function that has been used. Increasing the complexity doesn't necessarily result in better accuracy. For example, if RELU is used, increasing the number of layers leads to a worse accuracy. A Sigmoid activation function performs poorly independently of the number of layers used. On the contrary, the tanh activation performs excellently no matter the number of layers.

C.

The following figures show the max gradient for each layer and activation function considering the model described in (B). The value of the gradient was calculated on a batch of size 64. The output layer was included in the calculations:







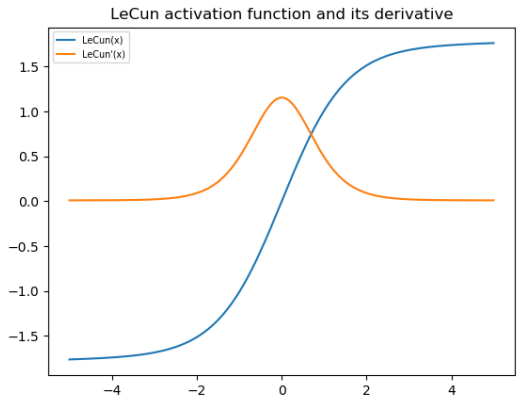
It can be observed that for the sigmoid activation function, which performs poorly no matter how many layers are used, the gradients change very little, which means that the backpropagation algorithm is converging very slowly to a solution. 3 epochs are not enough to obtain a good accuracy. If we consider RELU it can be noted that the distribution of the maximum gradients per layer is more and more unpredictable as the number of layers increase, suggesting the possible presence of gradient overflow or underflow events, which is consistent with the results that were obtained from the experiments. Lastly, the gradients for the Tanh activation function behave as expected, with greater steps towards the beginning of the network and smaller steps towards the output of the network. This is also compatible with the results obtained from the experiments. Tanh has a good accuracy no matter how many layers are used.

D.

Same as (B) but using the activation functions (LeCun, Tanh). The results are:

Activation Function	Layers	Accuracy
LeCun	5	94.34%
LeCun	20	94.72%
LeCun	40	92.89%
Tanh	5	93.48%
Tanh	20	92.35%
Tanh	40	91.82%

LeCun and Tanh activation both have a very good accuracy independently of the number of layers. Following is the general form of the backpropagation equations of MLPs for LeCun:



$$f(x) = 1.7159 tanh\left(\frac{2}{3}x\right) + 0.01x$$

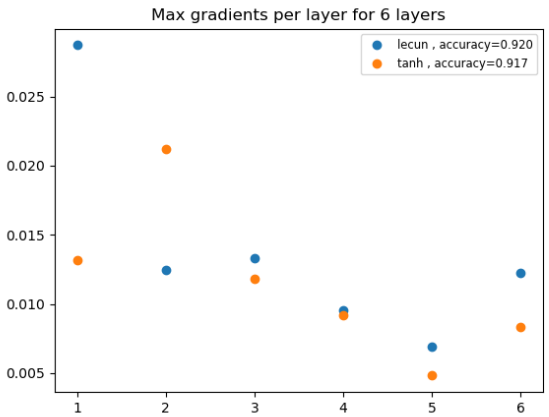
$$f'(x) = \left(1.7159\frac{2}{3}(1 - tanh(x)^2) + 0.01\right)$$

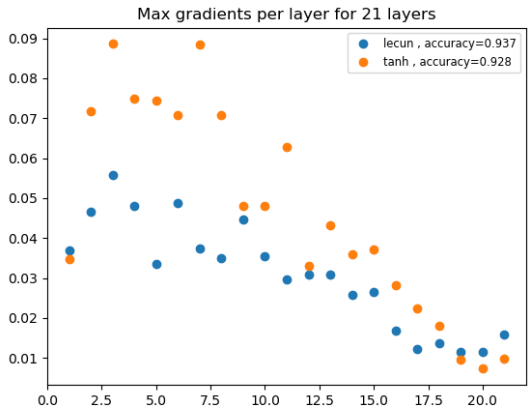
$$\delta w_{ij}{}^{(R)} = -\varepsilon \sum_{\mu} \left(1.7159\frac{2}{3}\left(1 - tanh\left(v_{i\mu}{}^{(R)}\right)^2\right) + 0.01\right)\left(y_{i\mu} - T_{i\mu}\right) y_{j\mu}{}^{(R-1)}$$

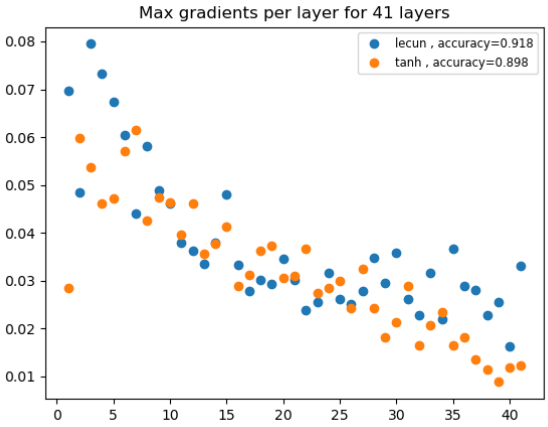
$$\delta w_{ij}{}^{(r)} = -\varepsilon \sum_{\mu} \left(\sum_{k} \delta_{k\mu}{}^{(r+1)} \left(1.7159\frac{2}{3}\left(1 - tanh\left(v_{i\mu}{}^{(r)}\right)^2\right) + 0.01\right)\right) y_{j\mu}{}^{(r-1)}$$

The gradients lie in the interval [0, 1].

The following figures show the max gradient for each layer and activation function for LeCun and Tanh:





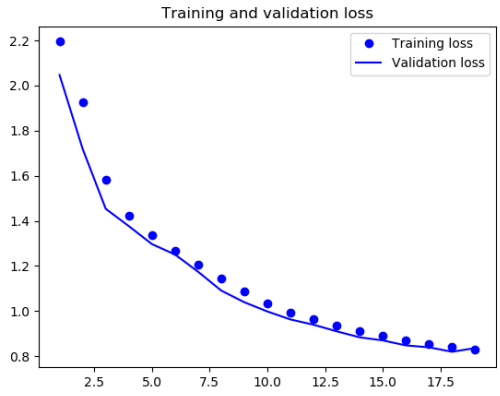


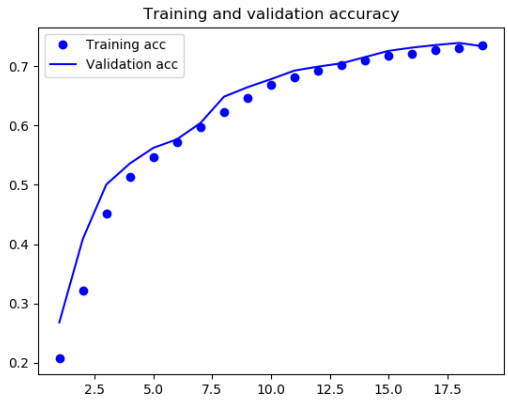
It can be observed that because the accuracy of both activation functions is excellent, the gradients behave consistently as the number of layers increase.

Exercise 2:

A.

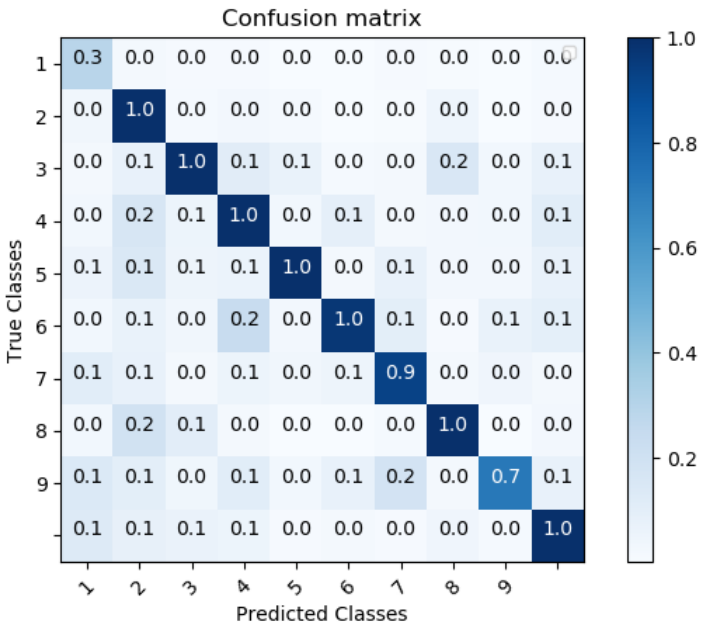
When training neural networks is difficult to choose the number of training epochs to use. Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that consists in specifying an arbitrary large number of training epochs and stop training once the model performance stops improving on a holdout validation data. The training will be stopped when the loss starts to increase and the validation accuracy starts to decrease.





After running the experiment with the requested parameters, the number of epochs where this happens is more or less 18 or 19. The accuracy for the test set is 73%.

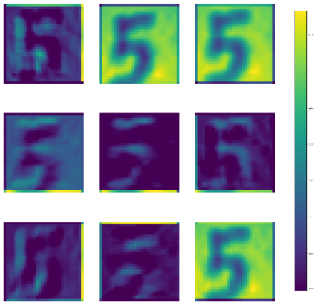
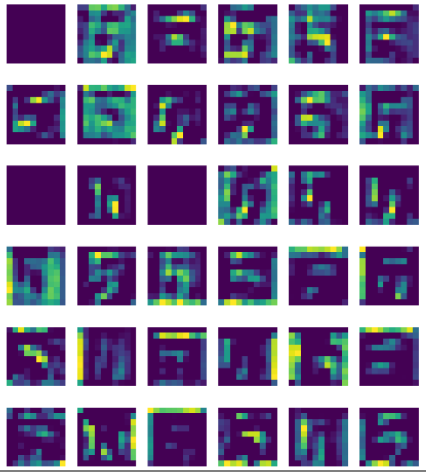
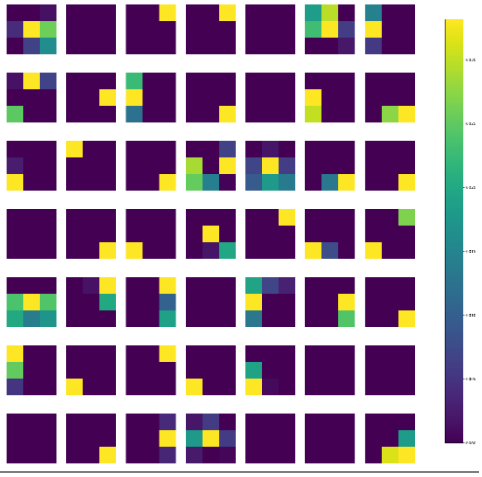
More insight can be gained by looking at the table of confusion, that reports the number of false positives, false negatives, true positives, and true negatives:

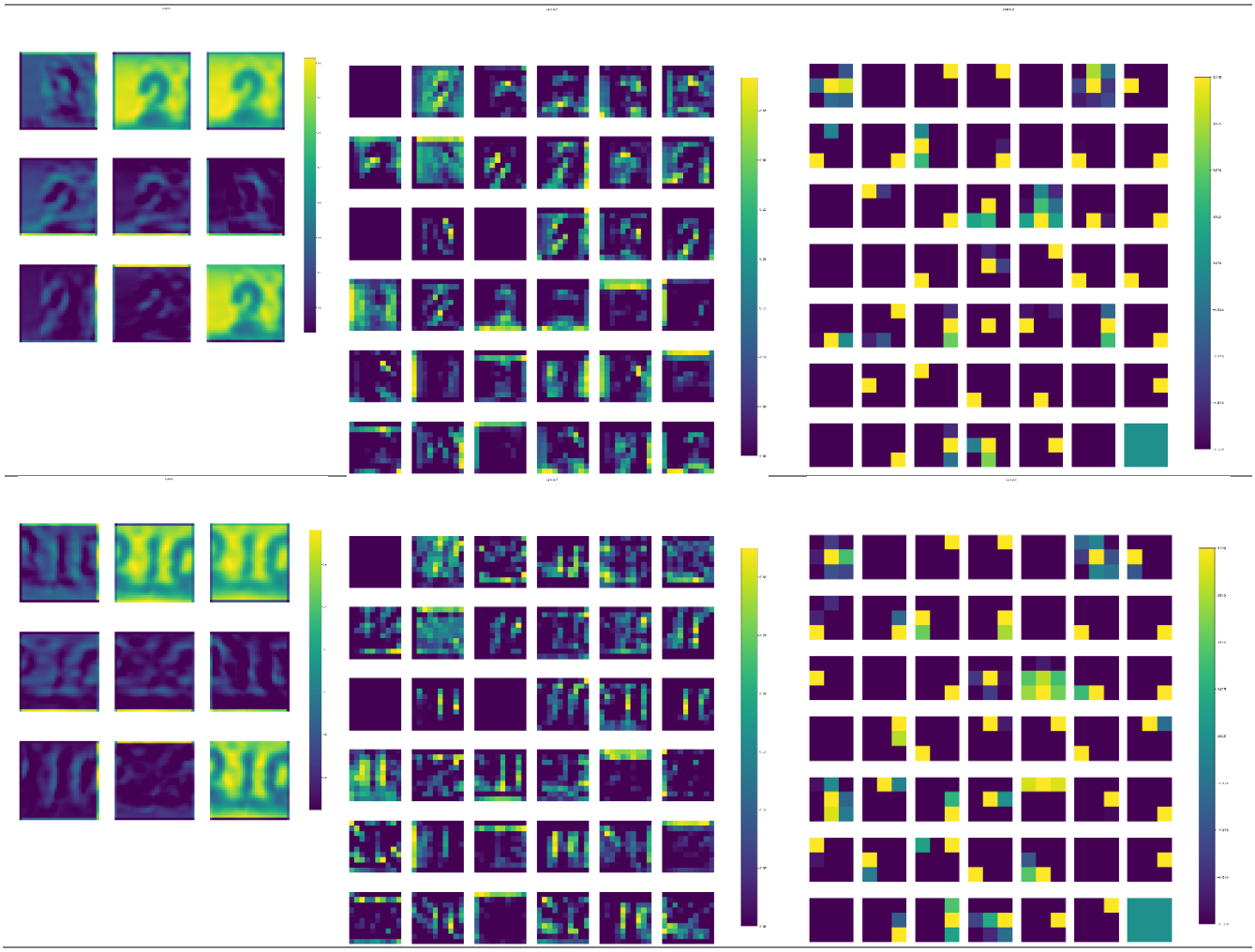


The results depicted in the confusion matrix are consistent with the accuracy of the model, showing a very good ability in classifying correctly the images to their class/classes, with few cases of misclassification.

B.

The following images represent the (A) model's output for each convolutional filter for some test samples (5, 2, 210):

1st convolution	2nd convolution	3rd convolution
		



It can be observed that through back propagation, the images tune themselves to become blobs of colored pieces and edges. At every convolutional layer, the filters are doing dot products to the input of the previous convolution layers. So, they are taking the smaller colored pieces or edges and making larger pieces out of them. They are extracting more and more basic features of the original image.

Exercise 3:

After reading the approach described here https://www.sciencedirect.com/science/article/pii/S0031320311004006
I decided to use implement a model similar to a hybrid CNN-SVN classifier:

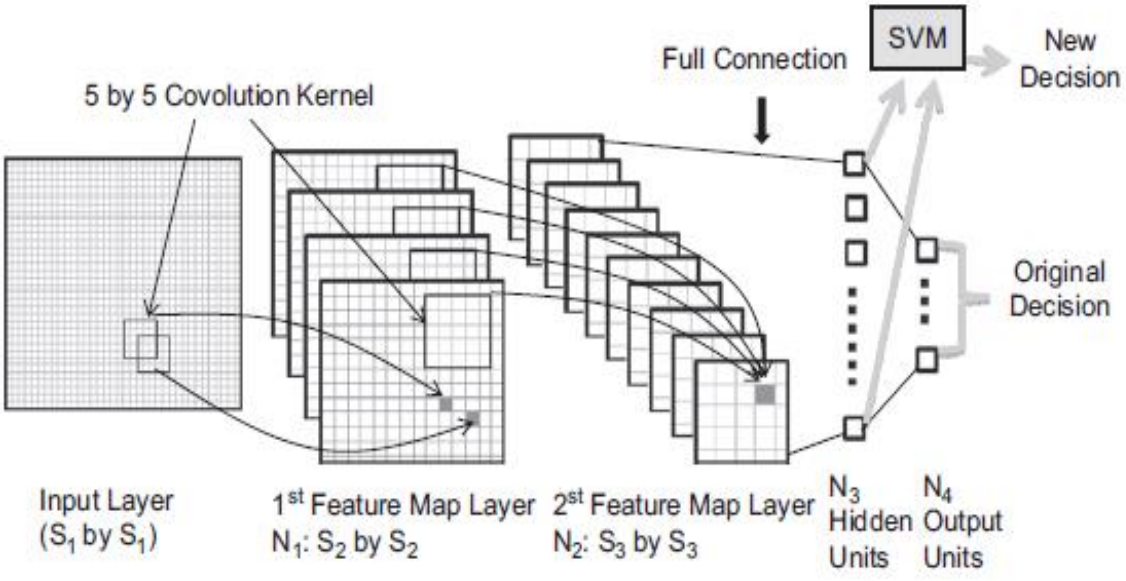


Fig. 2. Structure of the hybrid CNN–SVM model.

My model differs from the one described in the paper in the fact that it does not implement an RBF as the kernel of the SVM layer at the end of the network. The MNIST dataset was used to train the model and some transformations were operated on the images to generate more training and validation data. Unfortunately, my model has not shown a good accuracy with test images taken from a source other than MNIST.