Neural Networks: Implementing a Convolutional Neural Network with Keras
Deep Learning in Computer Vision Lab 4

Awet Haileslassie Gebrehiwot, Santiago Herrero Melo

I. INTRODUCTION

In this fourth lab session of *Deep Learning in Computer Vision* we will see how to build a convolutional neural network with the Keras framework in Python. We will use the MNIST dataset, one of the most used datasets in machine learning research, and the CIFAR10 dataset, which consists of color images of 10 different classes. The project will be developed in python language, using Keras in the tensorflow environment.
A Convolution neural network(CNN) composed of the following components:

- *Convolutional layers*
- *Pooling layers*
- *Fully connected layers*
- *A choice of activation function and optimizer*

A. MNIST dataset:

This is the dataset we have been working with in the previous lab sessions. It consists of 70000 grayscale images of handwritten digits (of size 28x28). The images show the digits from 0 to 9, and the program should be able to identify which digit appears in the image.

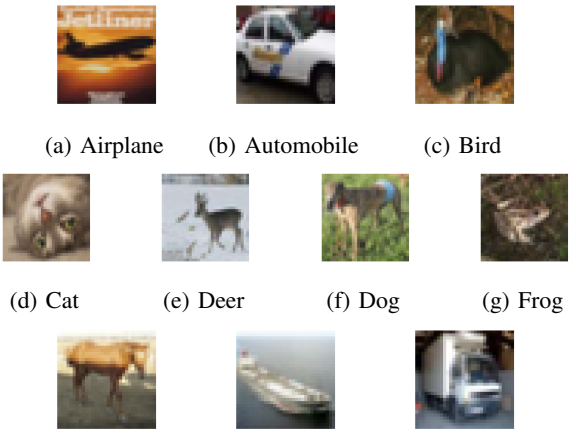
B. CIFAR10 dataset:

This dataset consists of 60000 (50000 for training and 10000 for testing) color images of size 32x32 labelled in ten different classes, which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The program should be able to identify to which of these categories the image corresponds to.

II. IMPLEMENTATION

A. MNIST dataset:

In the first place, we have to import the dataset. Then, we import *Sequential* Keras model in order to create our model. This model is a sequential stack of layers which performs the learning making minimal assumptions on the sequence structure.
We add as many convolutional layers as needed. In each of them we define the *number of outputs*, the *size of the kernel*, the *activation function* and the *size of the input shape*.
We can also add pooling layers in order to choose the best features, and to reduce the dimensions and dropout in order to turn neurons on or off to improve convergence.



(a) Airplane (b) Automobile (c) Bird

(d) Cat (e) Deer (f) Dog (g) Frog

(h) Horse (i) Ship (j) Truck

Fig. 1: Representative images from each class of the CIFAR10 dataset

Once the neural network is created, we have to configure the learning process via the *compile* function. We have to set parameters such as *loss function*, *optimizer* and *metrics*, this last one in order to measure the accuracy.
The following step is to train the model, for which we will use the *fit* function. In this case, we will have to set which are the training data and labels, the number of epochs (100 in this case), the batch size and settle the validation data that we will use, which in our case will be the 20% of the training data.
We will finally evaluate the accuracy of our model with the *evaluate* function, using the test data and labels.

B. CIFAR10 dataset:

The procedure is similar to the previous exercise, but we have to take intro consideration a fundamental aspect of this dataset. As the images are color images, we have to let the model know by pointing out that the size of the input layer is not only the size of the image, but should also be multiplied by 3 as each image has three channels (the color channels).

C. Running the code

For running the code we first have to activate the Tensorflow environment. Once that is done, we access the

folder where the skeleton of the code is saved, and type the following line in the command window:
python lab4_skeleton.py for the MNIST dataset,
python lab4_2_skeleton.py for the CIFAR10 dataset.
In some cases, we have encountered a small issue when changing the computer where we work. The returned parameters of the *fit* function are sometimes called *acc, acc_value* and sometimes *accuracy, accuracy_value*. As this function does not allow indexing, this may need to be modified manually by the user if desiring to obtain the graphs.
In order to obtain the confusion matrix, the *seaborn* and *scikit-learn* packages should be installed. Otherwise, those lines of code should be commented as they can return errors.

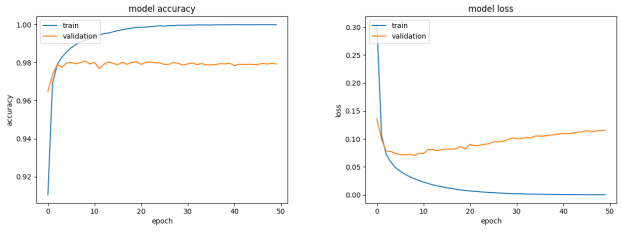
III. RESULTS AND ANALYSIS

A. MNIST dataset:

We have tested different architectures for the MNIST dataset. For all the architectures, the optimizer employed has been the *adaDelta* optimizer, and we have used the categorical cross entropy.

1) First architecture: : The first tested architecture is a very simple one. It consists of a convolutional layer with 64 filters, that uses *ReLu* activation function and a kernel of 3x3 size. Afterwards, a hidden layer with 128 neurons is added between the convolutional layer and the output.

The obtained values for the accuracy and the loss in the test set, after 50 epochs, are *0.9819* and *0.099654*, respectively. The evolution of the training and validation set can be seen in Fig. (2).

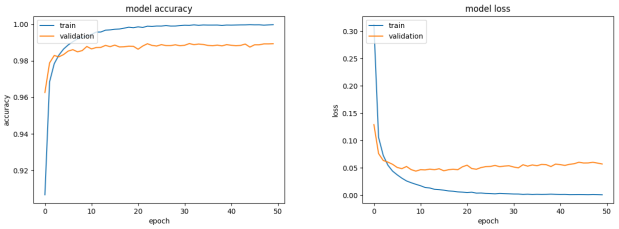


(a) Accuracy (b) Loss function

Fig. 2: Accuracy performance and loss function in the first architecture

2) Second architecture: : We have added a pooling layer and a dropout to the convolutional layer of the previous architecture. The pooling layer is the standard one, (2x2 layer) and the dropout is a 25% dropout. We also flatten the result after the pooling and before the hidden layer.

For the test set, the accuracy of this architecture is *0.9899*, while the loss value is *0.045982*. This results are obtained after performing 50 epochs. Fig. (3) shows the accuracy and loss function for the train and validation set.

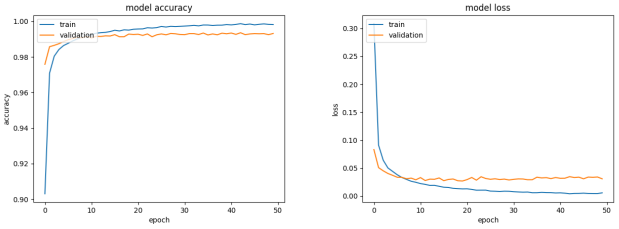


(a) Accuracy (b) Loss function

Fig. 3: Accuracy performance and loss function in the second architecture

3) Third architecture: : In this ocassion, we have added a new convolutional layer, pooling layer and dropout. This new convolutional layer also uses a 3x3 kernel and the *ReLu* activation function, but has 64 filters. The pooling layer is the standard one, while the dropout in this case is of the 30%.

In Fig. (4) we see the evolution of the validation and training set. After 50 epochs, the accuracy of the test set is *0.9932* and the loss is *0.02628717*.



(a) Accuracy (b) Loss function

Fig. 4: Accuracy performance and loss function in the second architecture

This is the architecture with which we obtain our best result. This model is the one that we finally selected because it maximizes the accuracy and minimizes the loss. Out of curiosity, we tried the same architecture with the *sigmoid* activation function, but changing the optimizer to *sgd*. In this case the results were worse, obtaining an accuracy of *0.8675* and much higher loss.

Arch	**loss**			**Accuracy**		
	Train	Val	**Test**	Train	Val	**Test**
First	0.00033	0.1158	0.09965	0.1.000	0.9793	0.9819
Second	0.00097	0.0572	0.046	0.9998	0.9893	0.9899
Third	0.0043	0.03295	**0.027**	0.9984	0.9941	**0.9942**

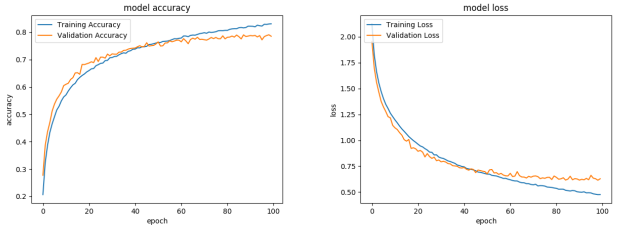
TABLE I: Summery of Experiments Loss and Accuracy values for the tested Architectures of MNST dataset

B. CIFAR10 dataset:

For all the proposed architectures, we are using an optimizer with learning rate 0.005 and momentum 0.9. This proved to achieve better results than simpler optimizers like *adam* or *sgd*.

1) First architecture:: This architecture consist of three blocks of a convolutional layer, a pooling layer and a dropout. The pooling layer is used to select the best and more representative features from the image, and the dropout is called to improve the convergence. Then, the last of these structures will be flattened and connected to a hidden layer of 512 neurons. This last hidden layer will have a dropout of the 20% before connecting to the output layer.
All the convolutional layers will be activated with the *ReLu* function and will have a 3x3 size kernel. The difference stands in the number of filters of the convolutional layers: the first one will have 32 filters, the second one 64 filters and the third one 128 filters.
All of the poolings will be size 2, which is the standard one. The dropouts change value for each layer, being the first one 20%, the second one 25% and the third one 30%.

With this architecture, we obtain the accuracy and loss function shown in Fig. (5). After running 100 epochs, the final accuracy achieved for the test set is *0.7778* and a loss value of *0.65049*.



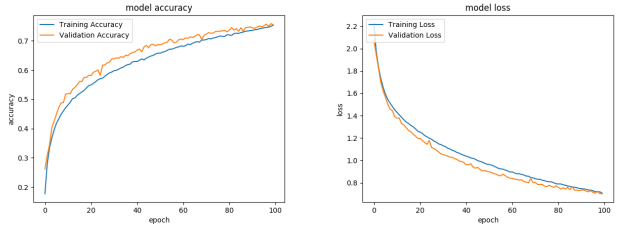
(a) Accuracy (b) Loss function

Fig. 5: Accuracy performance and loss function in the first architecture

2) Second architecture:: The architecture is the same as in the previous case, with the only difference of the initialization of the convolutional layers. In this case, we have added padding for them in order to also work with the pixels located at the edges, and we initialize the kernel as *he_uniform* in order to initialize the weights drawing samples from a truncated normal distribution centered on 0.

The resulting accuracy and loss function are shown in Fig. (6), with final values of *0.7802* and *0.6436* for the test accuracy and the test loss, respectively, after running 100 epochs.

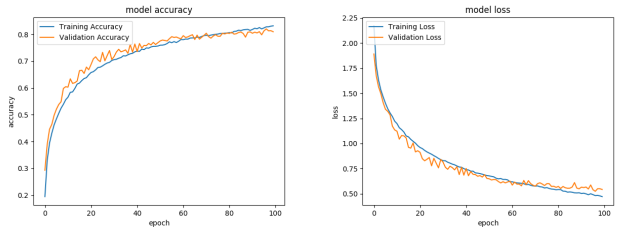
3) Third architecture:: In order to try to improve the accuracy, we have kept the previous architecture, including



(a) Accuracy (b) Loss function

Fig. 6: Accuracy performance and loss function in the second architecture

the new initialization of the convolutional layers, and added a fourth convolutional layer right before the hidden layer, with its corresponding pooling layer and dropout. In this case, the convolutional layer will have 256 filters with the same initialization as the other ones. The pooling layer is the same as in the other cases, and the dropout in this case if 40%.
In Fig. (7) we can see an improvement in the accuracy and loss, being their final values for the test set *0.8065* and *0.5601293*, respectively. This values correspond to running 100 epochs.



(a) Accuracy (b) Loss function

Fig. 7: Accuracy performance and loss function in the third architecture

4) Fourth architecture:: In This last architecture is have tryed to use the advantage of VGG with 4 Blocks type Architecture, we double every convolutional layer that we have made on Third Architecture experimentation ” ”, as shown in Fig. (8). This is the structure with which we achieved our best results.

The final loss for the test set is *0.62597*, which is a bit higher than in the previous case. On the other hand, the accuracy has increased up to *0.8341* for test dataset, which is very high improvement from all Experiments we have tried. For the training and validation set, we can see the evolution in Fig. (9).

We have proposed the Forth CNN Architecture (Fig. (8) for better classification of CIFAR10 Dataset as this architecture is fast to train and results with the best Maximum Accuracy on test dataset.

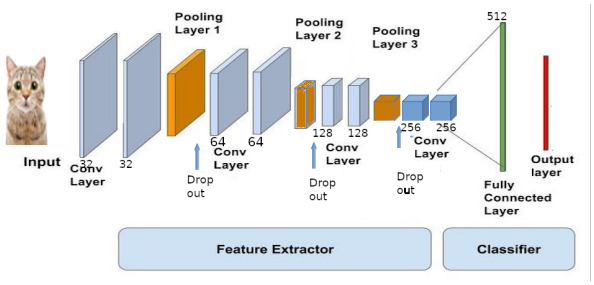
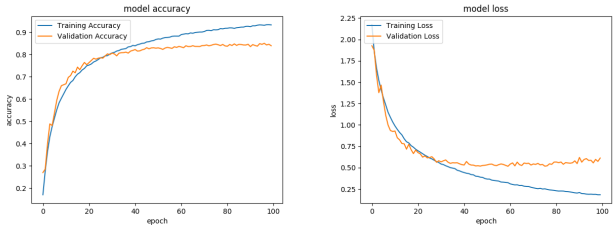


Fig. 8: Proposed CNN Architecture for better classification CIFA10 dataset



(a) Accuracy (b) Loss function

Fig. 9: Accuracy performance and loss function in the fourth architecture

As we can observe from Confusion Matrix shown on Fig. (10), for the proposed CNN architecture, we can conclude that the CNN Classification Accuracy shows a Promising result, except for the class category of **Cat** and **Dog** as they are similar in the way they look(have similar appearance) even for Humans(People) the CNN also confused to distinguish between them. Summery of All the experimented CNN Architectures with detailed Loss and Accuracy values are Shown on Fig.(II).

Arch	**loss**			**Accuracy**		
	Train	Val	**Test**	Train	Val	**Test**
First	0.4762	0.6285	0.65049	0.8308	0.7854	0.7778
Second	0.4693	0.6198	0.64368	0.8314	0.7908	0.7802
Third	0.4723	0.5434	0.560129	0.8319	0.8099	0.8065
Fourth	0.1841	0.6126	0.62597	0.9929	0.8394	**0.8341**

TABLE II: Summery of Experiments Loss and Accuracy values for the tested Architectures of CIFAR dataset

C. 10 Worst Classified Images from CIFAR Dataset

We have programmed our Algorithm to give us 10 Worst Classified Images, As a result the generated output can be shown on Fig.(11)

IV. Conclusion

As we have observed from our experiments, making the network deeper and deeper does not guaranty improvement in the Accuracy, where as we have to carefully assess the parameters that we have used for the type of problem we wanted to solve. We also believe that by doing(adding) some Data Augmentation techniques, We can achieve more accurate and better Classification algorithm as this can be

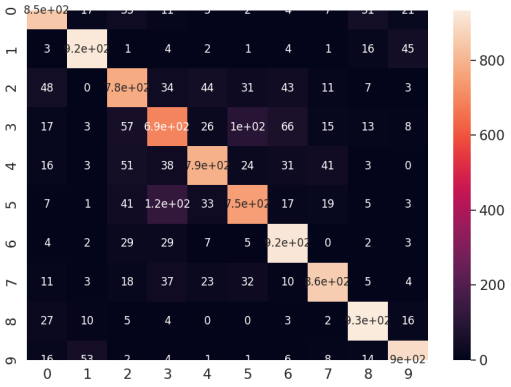


Fig. 10: Confusion matrix for CIFAR10 dataset obtained using best architecture (forth architecture)

Fig. 11: 10 Worst Classified Images for CIFAR10 dataset obtained using best architecture (forth architecture)

used to artificially expand the size of a training dataset by creating modified versions of images with better discrimination property that can support CNN feature extraction.