MSc in Data Science

Course: Deep Learning

Assignment 1

MLPs and CNN for Image Classification
Fashion item recognition

Professor: Prodromos Malakasiotis

Team Members

Adamidi Eleni: **p3352002 - eadamidi@aueb.gr**
Giannakos Ilias Dimitrios: **p3352007 - idgiannakos@aueb.gr**

Contents

Abstract…… 3
1. Import and pre-process the Fashion-MNIST dataset..……..…..... 4
2. Multi-Layer Perceptron (MLP) architecture…….…….... 5
3. Convolutional Neural Network (CNN) architecture……..……. 8

Abstract

The goal of this assignment is to implement a simple workflow, building a deep learning model so that it “automatically” learns the patterns required to recognize what kind of fashion item is presented in an input image. In the assignment, we create 2 different architectures, one with multi-layer perceptron’s (MLPs) and one with Convolutional Neural Networks (CNNs), using the Fashion-MNIST dataset. The two deep learning models were created by using the Tensorflow-Keras API. In the report, we explain the architectures that were used, how they were trained and tuned, and also we describe the challenges and problems and how they were addressed.

Fashion-MNIST is a dataset of Zalando’s article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 categories. The available categories and their labels are presented below:

Labels	Description
0	T-shirt / Top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

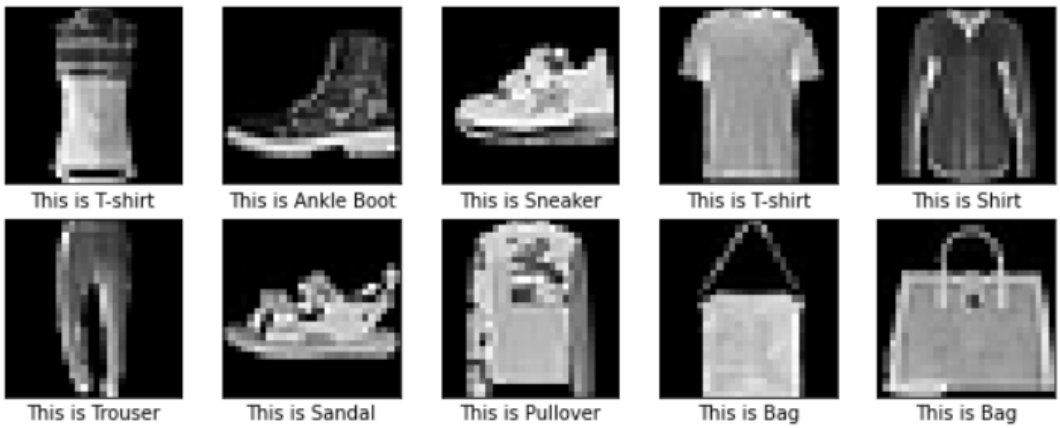
1. Import and pre-process the dataset with customers

To begin with, we import and pre-process the data in order to do computations for the items we have at hand. We import the necessary libraries, and we initialize the random number generator (seed) to ensure results of dataset split & modelling metrics (accuracy / precision) are reproducible across different sessions / computers.

Checking the distribution of each label, we observe that train and test datasets are balanced as there are the same number of occurrences between different clothes (6,000 items for each label on training set and 1,000 items for each label on test set). As a result, since we have a balanced dataset, we will utilize accuracy as our main KPI to judge the model’s effectiveness. Moreover, we split the original training set into training set and validation set, to tune our model on a subset of training data which were not used for training.

We one hot encode the clothing integers (from 0 to 9) to a vector of shape (10,1) in order to be able to use it as y during training on Keras API. Furthermore, we also normalize the pixels, by making a division of 255 to each pixel value. Finally, we flatten the train and test data by reshaping their length to 784 and map the generated one-

hot-encoded y values back to human-readable descriptions. Below are visualized the 10 categories of the items.



Finally, for the modelling phase, we will be using as "default" values the maximum number of epochs to be 70, the batch size of 64 and a keras early stopping callback of "validation accuracy", which stops training once validation accuracy starts to decrease. Regarding the batch size, we are aware that we could also experiment with different values, but we wanted to keep it on a descent level so that training could be possible on a laptop's memory.

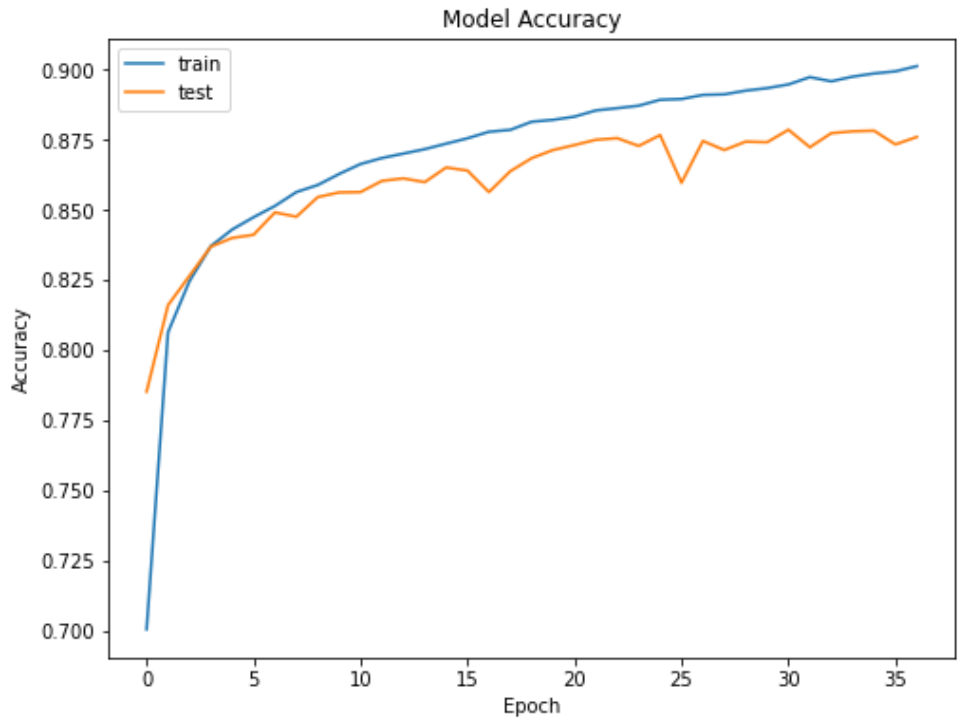
2. Multi-Layer Perceptron (MLP) architecture

We create a MLP model, defining the number of hidden layers, the number of neurons that each layer has and we set as activation function between layers, the ReLU (Rectified Linear Unit) function, as default. We start from the sequential model and we add as first layer the Input (default) with shape 784. For the first (n-1) hidden layers, we add a Dense (fully connected) layer, with the defined number of neurons and the same activation function. After each layer that we add, if there is dropout, we add one Dropout layer with the given dropout value (dropout ≠ None). Afterwards, we add one more Dense layer with the half number of neurons this time and the same activation function, and one last layer with 10 units (same as the number of different item categories), having the softmax as activation function. After setting an optimizer, we compile the model with categorical cross entropy loss since we have classification and we take as metrics the accuracy of the model. After creating the model, we fit it with the training data, while having the default values of batch size, optimizer and number of epochs mentioned earlier.

Taking the baseline model with two hidden layers, 128 neurons and the Stochastic Gradient Descent (SGD) as optimizer, we fit the model based on the training set with the development set as validation data (for hyperparameter tuning) and we take the following results:

```
Epoch 15/70
797/797 [==============================] - 1s 2ms/step - loss: 0.3584 - accuracy: 0.8735 - val_loss: 0.3779 - val_accuracy: 0.8650
Epoch 16/70
797/797 [==============================] - 1s 2ms/step - loss: 0.3521 - accuracy: 0.8754 - val_loss: 0.3772 - val_accuracy: 0.8639
Epoch 17/70
797/797 [==============================] - 1s 2ms/step - loss: 0.3468 - accuracy: 0.8777 - val_loss: 0.4051 - val_accuracy: 0.8562
Epoch 18/70
797/797 [==============================] - 1s 2ms/step - loss: 0.3427 - accuracy: 0.8784 - val_loss: 0.3763 - val_accuracy: 0.8637
Epoch 19/70
797/797 [==============================] - 1s 2ms/step - loss: 0.3375 - accuracy: 0.8813 - val_loss: 0.3680 - val_accuracy: 0.8683
Epoch 20/70
797/797 [==============================] - 2s 2ms/step - loss: 0.3330 - accuracy: 0.8820 - val_loss: 0.3620 - val_accuracy: 0.8712
Epoch 21/70
797/797 [==============================] - 2s 2ms/step - loss: 0.3288 - accuracy: 0.8831 - val_loss: 0.3575 - val_accuracy: 0.8730
Epoch 22/70
797/797 [==============================] - 2s 2ms/step - loss: 0.3251 - accuracy: 0.8853 - val_loss: 0.3571 - val_accuracy: 0.8749
Epoch 23/70
...
797/797 [==============================] - 2s 2ms/step - loss: 0.2835 - accuracy: 0.8985 - val_loss: 0.3439 - val_accuracy: 0.8781
Epoch 36/70
797/797 [==============================] - 2s 2ms/step - loss: 0.2809 - accuracy: 0.8993 - val_loss: 0.3489 - val_accuracy: 0.8732
Epoch 37/70
797/797 [==============================] - 2s 2ms/step - loss: 0.2781 - accuracy: 0.9011 - val_loss: 0.3423 - val_accuracy: 0.8759
```

We notice that early stopping stops the procedure in epoch 37, which means that the best (higher) validation accuracy is in epoch 31, since patience is equal to 6. Therefore, for the baseline model, we have a validation accuracy equal to 0.8744. The validation accuracy of this model on the test data is 0.8887, which is even higher than validation's accuracy. Below, it is presented a plot for model accuracy, on train and test data.



Making a second try, we get the same model as above but this time with optimizer Adam. This second model gives the following results:

```
797/797 [==============================] - 2s 2ms/step - loss: 0.2158 - accuracy: 0.9199 - val_loss: 0.3568 - val_accuracy: 0.8827
Epoch 14/70
797/797 [==============================] - 2s 2ms/step - loss: 0.2093 - accuracy: 0.9221 - val_loss: 0.3169 - val_accuracy: 0.8922
Epoch 15/70
797/797 [==============================] - 1s 2ms/step - loss: 0.2043 - accuracy: 0.9236 - val_loss: 0.3216 - val_accuracy: 0.8881
Epoch 16/70
797/797 [==============================] - 2s 2ms/step - loss: 0.1965 - accuracy: 0.9265 - val_loss: 0.3423 - val_accuracy: 0.8908
Epoch 17/70
797/797 [==============================] - 2s 2ms/step - loss: 0.1919 - accuracy: 0.9279 - val_loss: 0.3387 - val_accuracy: 0.8891
Epoch 18/70
797/797 [==============================] - 2s 2ms/step - loss: 0.1851 - accuracy: 0.9298 - val_loss: 0.3216 - val_accuracy: 0.8897
Epoch 19/70
797/797 [==============================] - 2s 3ms/step - loss: 0.1792 - accuracy: 0.9333 - val_loss: 0.3553 - val_accuracy: 0.8882
Epoch 20/70
797/797 [==============================] - 2s 2ms/step - loss: 0.1756 - accuracy: 0.9340 - val_loss: 0.3378 - val_accuracy: 0.8920
```

We see that for model 2 we have the best validation accuracy in epoch 14, equal to 0.8920. We noticed that the model's training is much faster to converge with Adam optimizer compared to SGD, and it also produces better accuracy. We also used the second model we calculate test accuracy which is 0.8830 (slightly worse compared to baseline even though the drop is minimal). This implies that the model 2 is overall better than model 1 and Adam optimizer is generally better than SGD.

We now try a third MLP model, same architecture and hyperparameters as model 2 but now we change the number of hidden layers to 5.

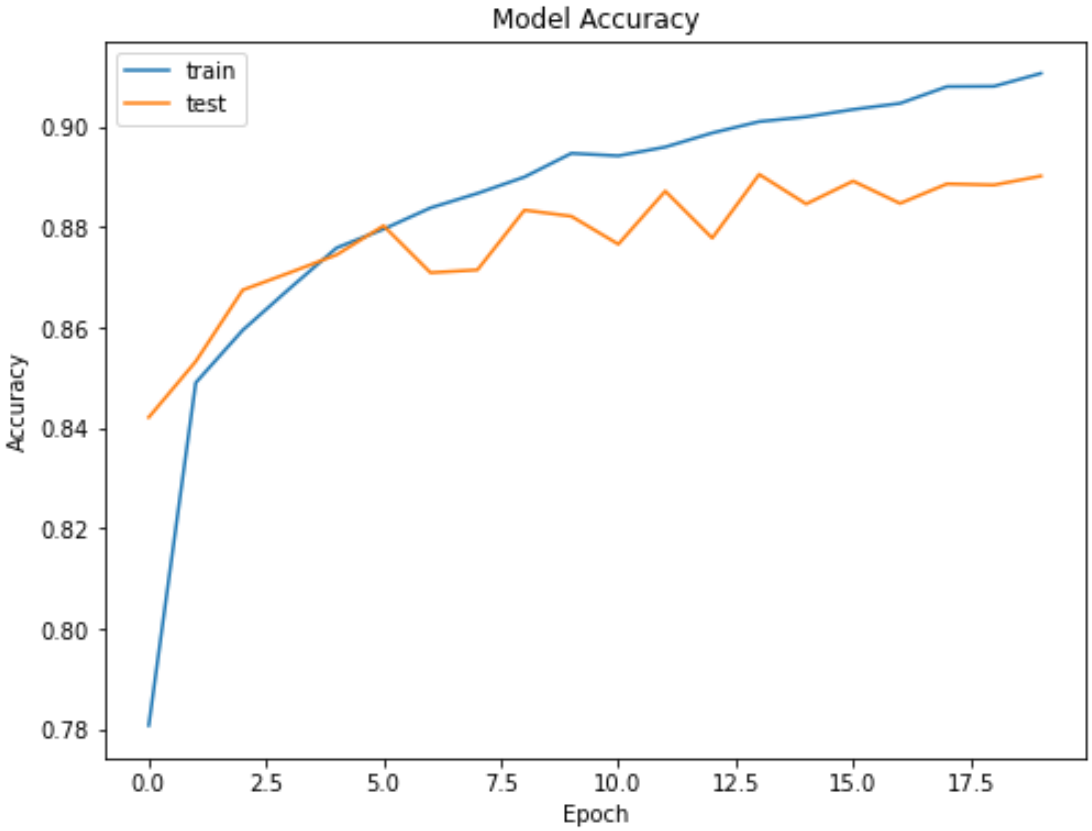
```
Epoch 19/70
797/797 [==============================] - 2s 3ms/step - loss: 0.1865 - accuracy: 0.9292 - val_loss: 0.3495 - val_accuracy: 0.8882
Epoch 20/70
797/797 [==============================] - 2s 3ms/step - loss: 0.1823 - accuracy: 0.9311 - val_loss: 0.3372 - val_accuracy: 0.8932
Epoch 21/70
797/797 [==============================] - 2s 3ms/step - loss: 0.1749 - accuracy: 0.9339 - val_loss: 0.3472 - val_accuracy: 0.8902
Epoch 22/70
797/797 [==============================] - 2s 3ms/step - loss: 0.1726 - accuracy: 0.9341 - val_loss: 0.3898 - val_accuracy: 0.8869
Epoch 23/70
...
797/797 [==============================] - 2s 3ms/step - loss: 0.1645 - accuracy: 0.9369 - val_loss: 0.4169 - val_accuracy: 0.8750
Epoch 25/70
797/797 [==============================] - 2s 3ms/step - loss: 0.1634 - accuracy: 0.9382 - val_loss: 0.3477 - val_accuracy: 0.8903
Epoch 26/70
797/797 [==============================] - 2s 3ms/step - loss: 0.1533 - accuracy: 0.9407 - val_loss: 0.3686 - val_accuracy: 0.8888
```

The process now stops in epoch 26, thus the best validation accuracy is in epoch 20 with value 0.888. Even though in theory a model with more hidden layers (thus more complex) did not actually improve the results compared to a model with 2 hidden layers. The validation accuracy of model 3 on test data is 0.8888. Therefore, we discard model3 and keep the previous model as baseline (model2).

Finally, we try a fourth model, which is same as our baseline model (model2) but now with dropout.

```
797/797 [==============================] - 2s 2ms/step - loss: 0.2263 - accuracy: 0.9156 - val_loss: 0.3007 - val_accuracy: 0.8938
Epoch 17/70
797/797 [==============================] - 2s 2ms/step - loss: 0.2227 - accuracy: 0.9150 - val_loss: 0.3211 - val_accuracy: 0.8833
Epoch 18/70
797/797 [==============================] - 2s 2ms/step - loss: 0.2198 - accuracy: 0.9154 - val_loss: 0.3093 - val_accuracy: 0.8920
Epoch 19/70
797/797 [==============================] - 2s 2ms/step - loss: 0.2134 - accuracy: 0.9192 - val_loss: 0.3451 - val_accuracy: 0.8841
Epoch 20/70
797/797 [==============================] - 2s 2ms/step - loss: 0.2123 - accuracy: 0.9187 - val_loss: 0.3101 - val_accuracy: 0.8917
Epoch 21/70
797/797 [==============================] - 2s 2ms/step - loss: 0.2090 - accuracy: 0.9209 - val_loss: 0.2999 - val_accuracy: 0.8932
Epoch 22/70
797/797 [==============================] - 2s 2ms/step - loss: 0.1993 - accuracy: 0.9250 - val_loss: 0.3138 - val_accuracy: 0.8911
313/313 [==============================] - 0s 1ms/step - loss: 0.3279 - accuracy: 0.8868
Test accuracy is  0.8867999911308289
This is a tiny improvement over model2, but as dropout regularizes the model we will keep this as our baseline model
```

We observed that model had a validation accuracy equal to 0.893. Even though the score did not improve, because of the dropout the regularization helps stabilize the model, therefore we will keep it as our baseline model. Below is a plot for model accuracy of model 4.



3. CNN architecture

Since CNN expects 3-dimensional data and not flattened as MLP needs, we must reshape our data by adding 1 as a third dimension of train, development and test dataset (since we already had grey-scaled images). We start by calling the sequential model and we add as first layer of our CNN model, the Conv2D (default) with shape (28,28,1). Also in this case, we take as

activation function the ReLU function (default) and we keep the Adam optimizer, since we found from MLP's that it produces better results. We continue, by adding 2 more Conv2D layers, one flatten layer and 2 Dense (full connected) layers, one with ReLU activation function and one with softmax. We compile the model with categorical cross entropy loss as before and we take as metrics the accuracy of the model.

The results of model 5 are presented below:

```
Epoch 9/70
797/797 [==============================] - 19s 24ms/step - loss: 0.1816 - accuracy: 0.9331 - val_loss: 0.2555 - val_accuracy: 0.9088
Epoch 10/70
797/797 [==============================] - 19s 24ms/step - loss: 0.1633 - accuracy: 0.9391 - val_loss: 0.2804 - val_accuracy: 0.9052
Epoch 11/70
797/797 [==============================] - 19s 24ms/step - loss: 0.1540 - accuracy: 0.9426 - val_loss: 0.2872 - val_accuracy: 0.9022
Epoch 12/70
797/797 [==============================] - 19s 24ms/step - loss: 0.1395 - accuracy: 0.9478 - val_loss: 0.2708 - val_accuracy: 0.9061
Epoch 13/70
797/797 [==============================] - 19s 24ms/step - loss: 0.1282 - accuracy: 0.9519 - val_loss: 0.3076 - val_accuracy: 0.8957
Epoch 14/70
797/797 [==============================] - 22s 27ms/step - loss: 0.1174 - accuracy: 0.9556 - val_loss: 0.2800 - val_accuracy: 0.9053
Epoch 15/70
797/797 [==============================] - 21s 26ms/step - loss: 0.1073 - accuracy: 0.9595 - val_loss: 0.3055 - val_accuracy: 0.9078
                                                                                                    + Code   + Markdown
```

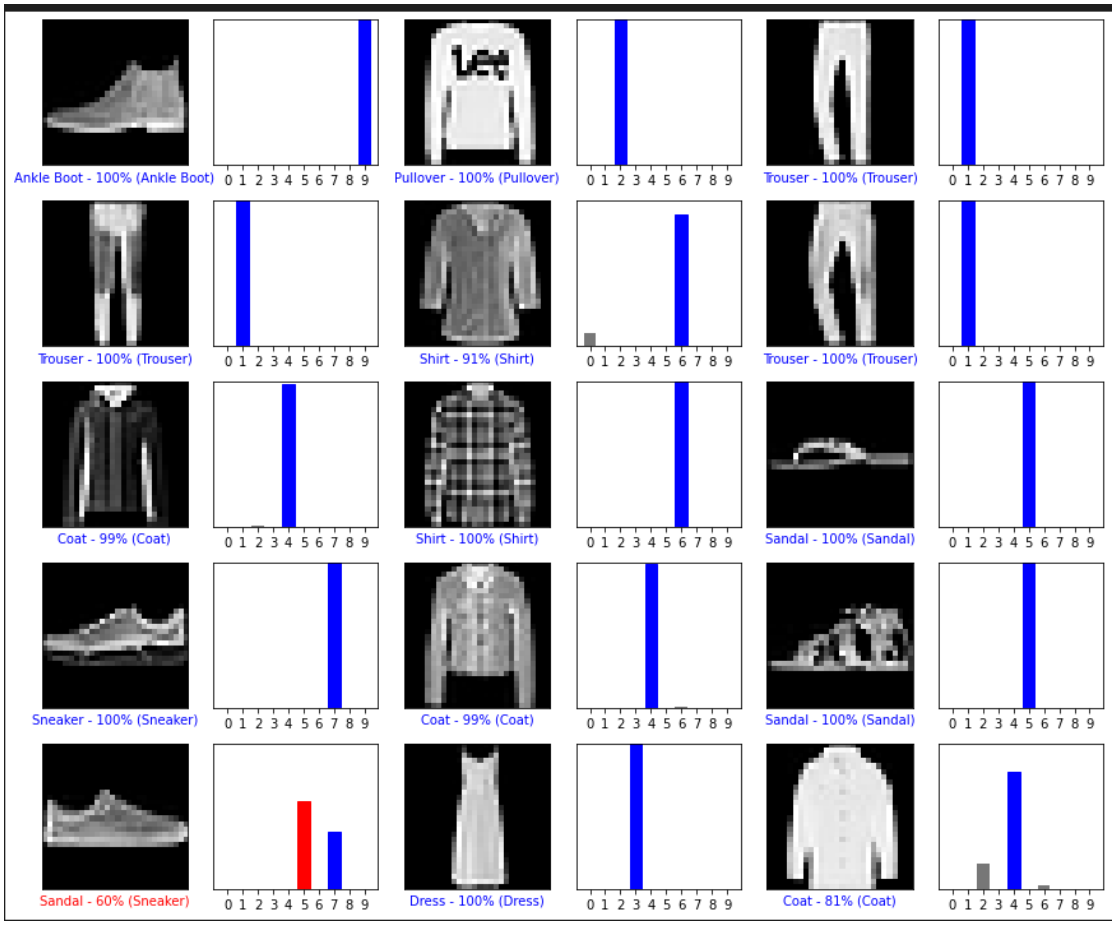
Without any regularization, CNN's have a very descent score, having a validation accuracy of 0.9088 and the validation accuracy on test data is 0.907. It is obvious that in this case we get better results than in MPL modelling.

We try one more final case; we take the model 5 but now we add two regularization techniques combines: dropout and batch normalization. We used batch normizalization as it is used to normalize the output of the previous layers. Observing the results below, we notice that model 6 has even better accuracy on test data than model 5.

```
Epoch 14/70
797/797 [==============================] - 33s 42ms/step - loss: 0.1692 - accuracy: 0.9356 - val_loss: 0.2532 - val_accuracy: 0.9130
Epoch 15/70
797/797 [==============================] - 35s 44ms/step - loss: 0.1634 - accuracy: 0.9376 - val_loss: 0.2544 - val_accuracy: 0.9148
Epoch 16/70
797/797 [==============================] - 35s 43ms/step - loss: 0.1599 - accuracy: 0.9386 - val_loss: 0.2716 - val_accuracy: 0.9132
Epoch 17/70
797/797 [==============================] - 35s 44ms/step - loss: 0.1537 - accuracy: 0.9419 - val_loss: 0.2616 - val_accuracy: 0.9128
Epoch 18/70
797/797 [==============================] - 34s 42ms/step - loss: 0.1485 - accuracy: 0.9428 - val_loss: 0.2716 - val_accuracy: 0.9140
Epoch 19/70
797/797 [==============================] - 35s 44ms/step - loss: 0.1470 - accuracy: 0.9428 - val_loss: 0.2644 - val_accuracy: 0.9138
Epoch 20/70
797/797 [==============================] - 34s 43ms/step - loss: 0.1430 - accuracy: 0.9454 - val_loss: 0.2572 - val_accuracy: 0.9139
Epoch 21/70
797/797 [==============================] - 36s 45ms/step - loss: 0.1404 - accuracy: 0.9457 - val_loss: 0.3150 - val_accuracy: 0.9071
Accuracy on test dataset= 0.9136999845504761
```

Our final model had a validation accuracy of 0.9148 and a test accuracy of 0.913, which is considered descent. Indeed, the 2 regularization techniques helped the CNN not overfit the training data.

Finally, we plot the first 15 test images with their predicted and the true labels as well as the model's confidence of the preidction. Color blue is for correct and color red for incorrect predictions.



Github link:

https://github.com/ilias-giannakos/Deep-Learning