Project 1

Data Science applied to Deep Learning

Raphaël CANIN, Melisa KOCKAN
E4 AIC

Supervised by Mr Badre BOUSALEM

Table of content

Supervised by Mr Badre BOUSALEM

Preambule

This lab will be divided into 3 parts, each focused on a different dataset.
After analyzing and building a model to make predictions on our first dataset, we chose another similar dataset on Kaggle to test the efficiency of our modelfs. Finally, we looked for a whole other different dataset on Kaggle to challenge the performance of our models.

The first dataset we will be working on is the **MNIST digit** data set, containing images/labels of handwritten digits. The second dataset we decided to use is the **Kuzushiji-MNIST** dataset, containing images/labels of the Japanese alphabet. The last dataset we focused on was a **Vegetable Image dataset**.

The **Kuzushiji-MNIST** and **Vegetable Image** data sets are available here :
https://www.kaggle.com/datasets/anokas/kuzushiji?select=kmnist_classmap.csv
https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset?fbclid=IwAR1JigA6y8UIKbWunei9Yi3mrFP2PO7JReWIoXp9v5ZkRBGxGUMNnpJ4xuE

1| MNIST-digit dataset

Presentation of the dataset

To apply data science on deep learning, the dataset suggested to be used was the MNIST digit Dataset, containing pictures of handwritten digits. The purpose of this part will be to predict the digits written on each picture.

Below, is a preview of the training images and their labels :

Digits: 5
Digits: 0
Digits: 4
Digits: 1
Digits: 9
Digits: 2
Digits: 1
Digits: 3
Digits: 1
Digits: 4

The shapes of our dataset is depicted below :

```
shape of X_train is :  (60000, 28, 28)
shape of y_train is :  (60000,)
shape of X_test is :  (10000, 28, 28)
shape of y_test is :  (10000,)
```

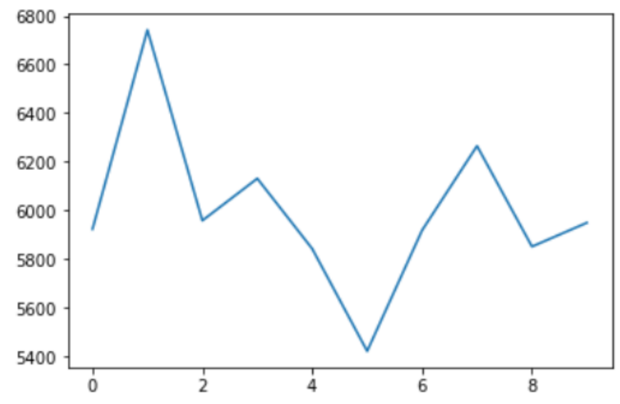
X_train corresponds to the array of images - which are themselves, represented by arrays of size 28x28 -, while y_train is an array of digits corresponding to the labels of each picture. The range values are indicated below, the value of a pixel can go from 0 to 255, while the value of the digit label goes from 0 to 9.

```
X_train is between  0  and  255  // type :  uint8
y_train is between  0  and  9  // type :  uint8
```

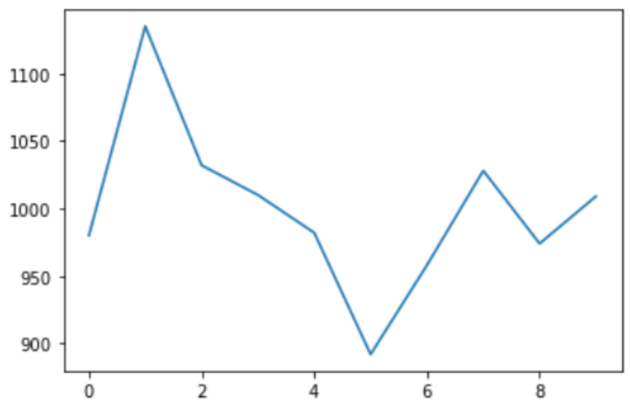
For more practical use, we decided to encode the labels of digits into array of 0 and 1 :

```
shape of X_train is :  (60000, 784)
shape of y_train is :  (60000, 10)     X_train is between  0  and  255  // type :  uint8
shape of X_test is :  (10000, 784)     y_train is between  0.0  and  1.0  // type :  float32
shape of y_test is :  (10000, 10)
```

Let’s analyze the repartition of our data set :



repartition of the train set



repartition of the test set

There is a discrepancy between the digits repartition : both training and testing set contain a lot of 1 and 7, while the digit 5 is sparsely represented.

Conception of models

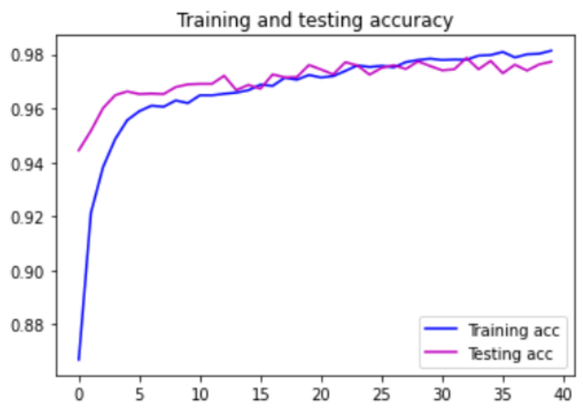
To make predictions on the pictures, we decided to use a typical neural network that we designed to our dataset by changing its parameters -namely the number of layers, neurons, the activation function and the dropout rate.

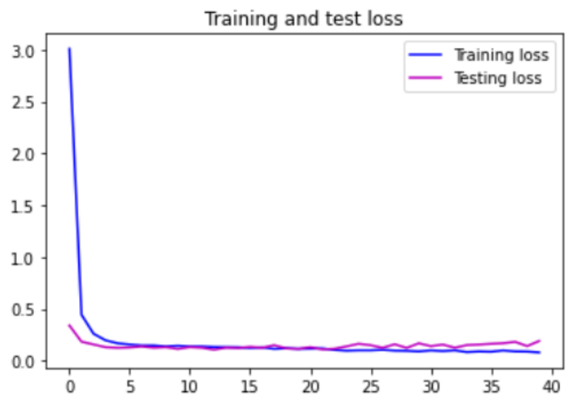
1) First Model

For our first model, we decided to use 3 layers interspersed with dropouts of rate 0.2:

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(28* 28, )))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation="softmax"))

model.compile(loss='categorical_crossentropy', metrics=['acc'], optimizer='adam')
```

After training, we obtained the following accuracy and loss curves :





```
Train accuracy : 0.9813833236694336
Test accuracy  : 0.9772999882698059
```

After using the predict command on our test set, the ratio of correct predictions was the following one :

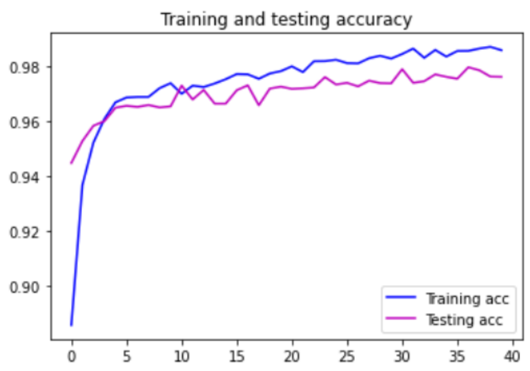
```
9618  predictions were correct
382  predictions were incorrect
96.18
```

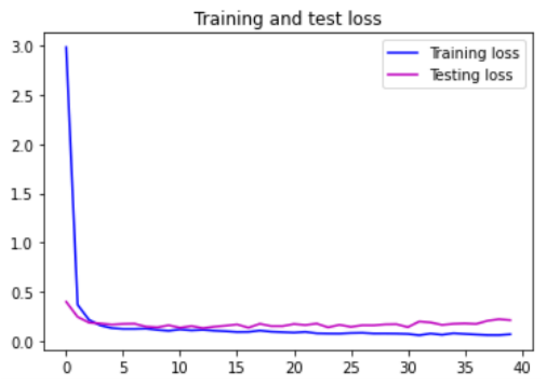
2) Second Model

For the second model, we modified the ***drop out*** parameter with different values and found out that 0.1 was a good one :

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(28* 28, )))
model.add(Dropout(0.1))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(10, activation="softmax"))

model.compile(loss='categorical_crossentropy', metrics=['acc'], optimizer='adam')
```

After training, we obtained the following accuracy and loss curves :





The accuracy we obtained is depicted below :

```
Train accuracy : 0.9872499704360962
Test accuracy  : 0.9786999821662903
```

After using the predict command on our test set, the ratio of correct predictions was the following one :

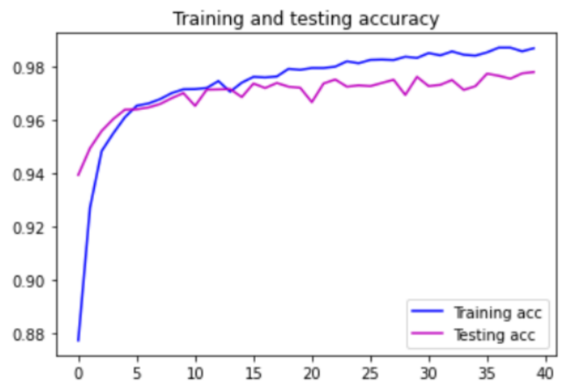
```
9659  predictions were correct
341  predictions were incorrect
96.59
```

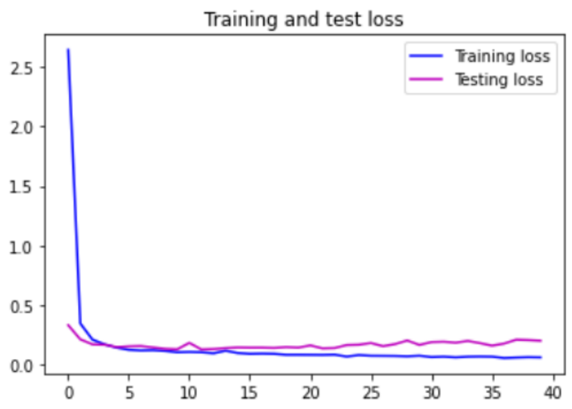
3) Third Model

For the third model, we kept the better drop out rate and looked for a more adapted ***number of neurons*** :

```
model = Sequential()
model.add(Dense(450, activation='relu', input_shape=(28* 28, )))
model.add(Dropout(0.1))
model.add(Dense(350, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(10, activation="softmax"))

model.compile(loss='categorical_crossentropy', metrics=['acc'], optimizer='adam')
```

After training, we obtained the following accuracy and loss curves :





The accuracy we obtained is depicted below :

```
Train accuracy : 0.9868166446685791
Test accuracy  : 0.9779000282287598
```

After using the predict command on our test set, the ratio of correct predictions was the following one :

```
9621  predictions were correct
379  predictions were incorrect
96.21
```

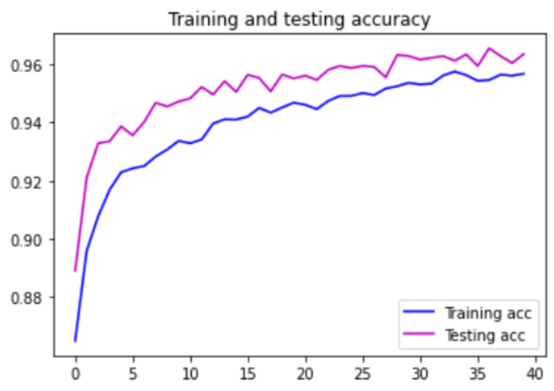
The accuracy didn’t really change, that is why we decided to add another layer in our following 5th model.

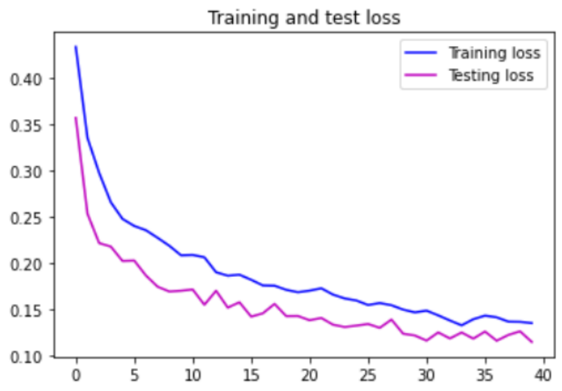
4) Fourth Model

For the fourth model, we tried to use other ***activation functions*** ; however, since relu already seemed to be the best, we decided to look for the worst and used tanh :

```
model = Sequential()
model.add(Dense(450, activation='tanh', input_shape=(28* 28, )))
model.add(Dropout(0.1))
model.add(Dense(350, activation='tanh'))
model.add(Dropout(0.1))
model.add(Dense(10, activation="softmax"))

model.compile(loss='categorical_crossentropy', metrics=['acc'], optimizer='adam')
```

The accuracy and testing curves are the following one :





The accuracy we obtain is depicted below :

```
Train accuracy : 0.9566500186920166
Test accuracy  : 0.9634000062942505
```

The results of this model are quite surprising : the test accuracy is bigger than the training accuracy. This phenomenon could be explained by the fact that the size of the testing set is way lower than the size of the training set.

The accuracy on the test set is around 96% ; however, while using the model.predict command we obtain a lower accuracy :

```
8972  predictions were correct
1028  predictions were incorrect
89.72
```

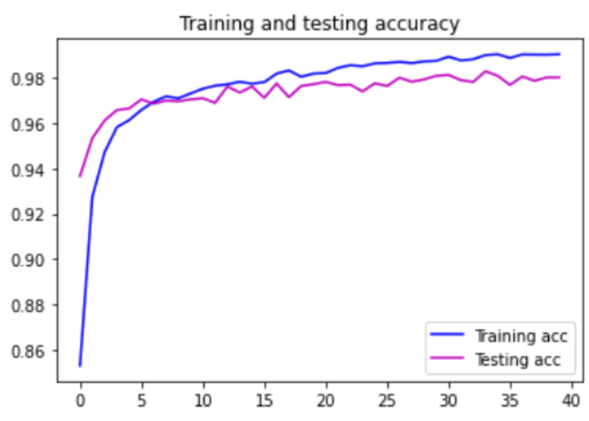
These percentages prove that using a tanh activation isn't adapted to our data set here.

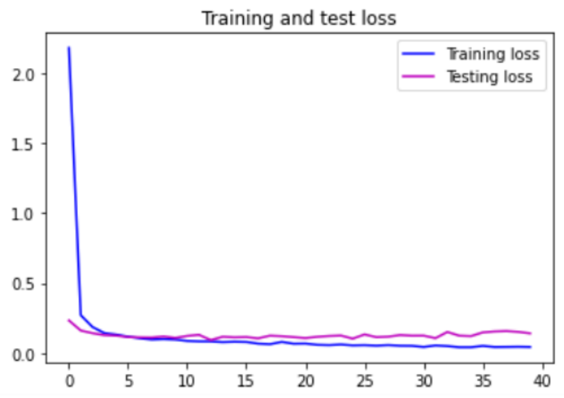
5) Fifth Model

For the fifth model, we decided to add another layer :

```
model = Sequential()
model.add(Dense(500, activation='relu', input_shape=(28* 28, )))
model.add(Dropout(0.1))
model.add(Dense(400, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(250, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(10, activation="softmax"))

model.compile(loss='categorical_crossentropy', metrics=['acc'], optimizer='adam')
```

The results of the training is represented below :





```
Train accuracy : 0.9902999997138977
Test accuracy  : 0.9800999760627747
```

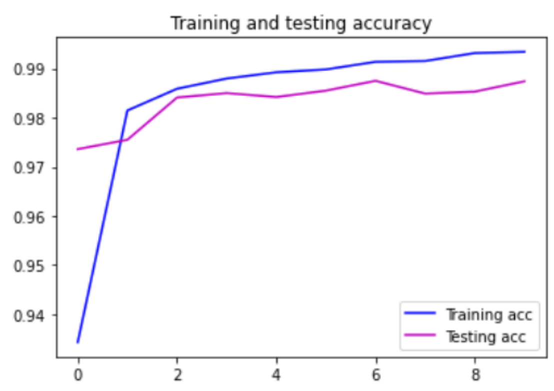
With this configuration, the accuracy obtained on the testing set is better than for the previous model, it reached 98%.

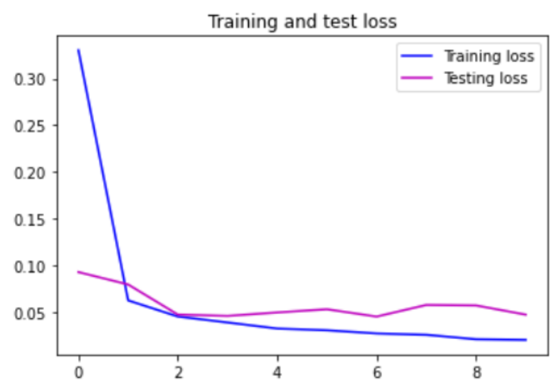
6) Sixth Model

For our last model, we decided to use a Convolutional layer -which is specifically known for performing well on images dataset.

```
model = Sequential()
model.add( layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)) )
model.add( layers.MaxPooling2D((2, 2)) )
model.add( layers.Conv2D(64, (3, 3), activation='relu') )
model.add( layers.MaxPooling2D((2, 2)) )
model.add( layers.Conv2D(64, (3, 3), activation='relu') )
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

The accuracy we obtain is way better than with a basic neural network :





```
Train accuracy : 0.993399977684021
Test accuracy  : 0.9873999953269958
```

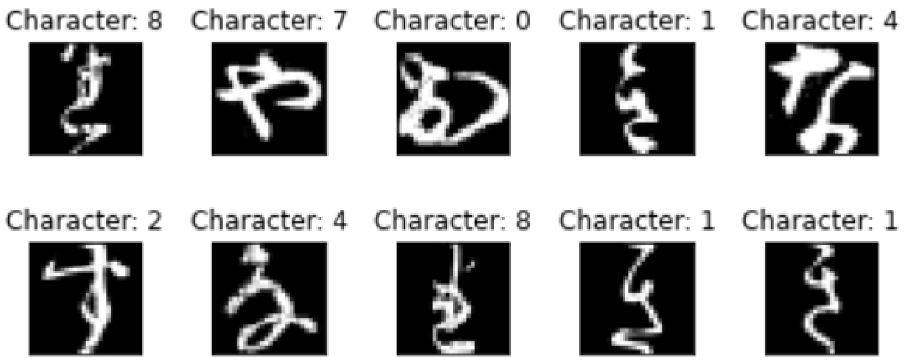
Conclusion

The results are globally satisfying, however, the models will be tested on a similar dataset in order to challenge their performance.

2| MNIST-kushiji dataset

Presentation of the dataset

The second data set used for this lab is also a MNIST, hence, the sizes of images are similar to the MNIST-digit.
To get a glimpse of what are Kushiji, here is a sample of 10 pictures :



The repartition of each character however, is very different from the first dataset. Indeed, each character has exactly the same number of images :

```
[6000. 6000. 6000. 6000. 6000. 6000. 6000. 6000. 6000. 6000.]
[1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.]
```

Overfitting and Underfitting

Using a MNIST dataset with a different repartition can enable to detect overfitting or underfitting.

Overfitting

The phenomenon of overfitting is characterized by a surprisingly high accuracy on the training set and a poor accuracy on the testing set. It appears when the model bases its learning on the pattern of the data set.

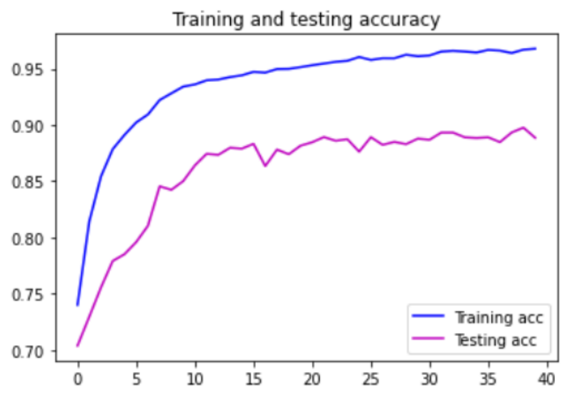
Underfitting

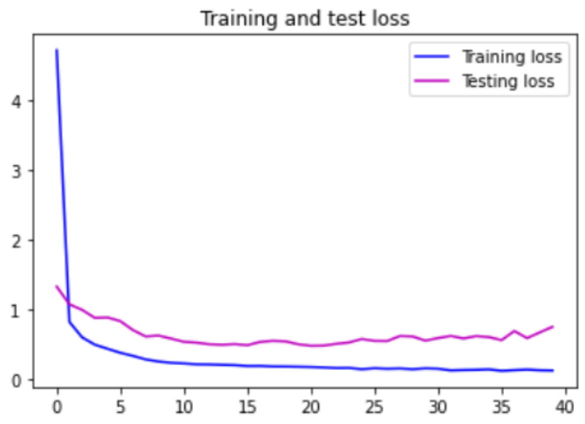
According to https://www.ibm.com/cloud/learn/underfitting, the phenomenon of underfitting appears when a model is "*unable to capture the relationship between the input and output variables accurately*".

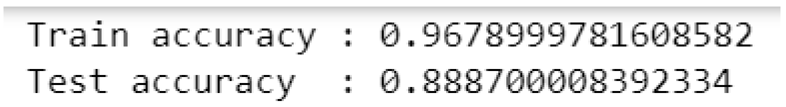
Testing models

The exact same models as for the first data set has been reused on this second data set. Hence, no explanations have been given, the following part only provides the results related to accuracy and loss ; afterward, a conclusion will be made.

1) First Model

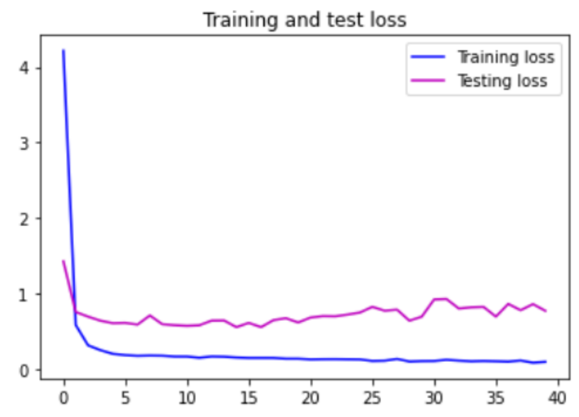


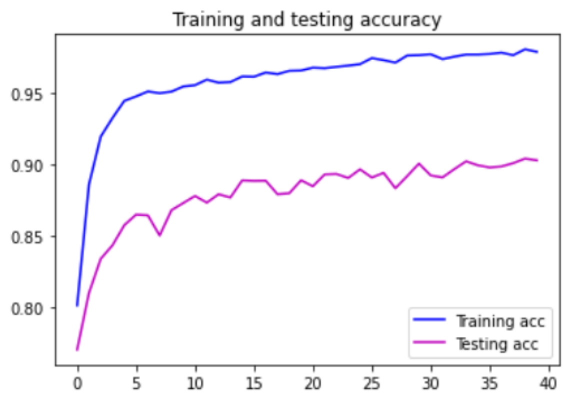




```
Train accuracy : 0.9678999781608582
Test accuracy  : 0.888700008392334
```

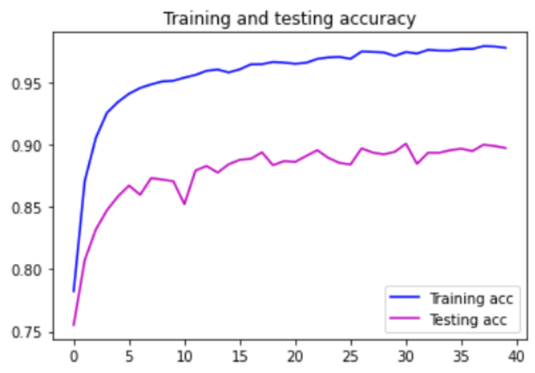
2) Second Model

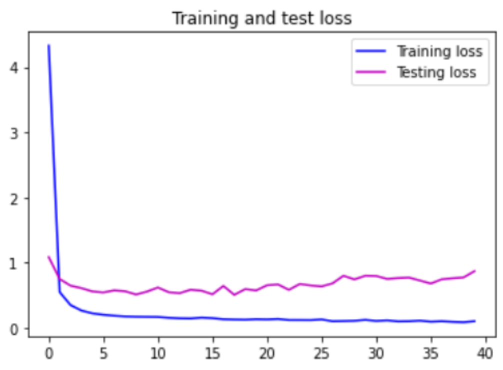




```
Train accuracy : 0.979033350944519
Test accuracy  : 0.902899980545044
```

3) Third Model

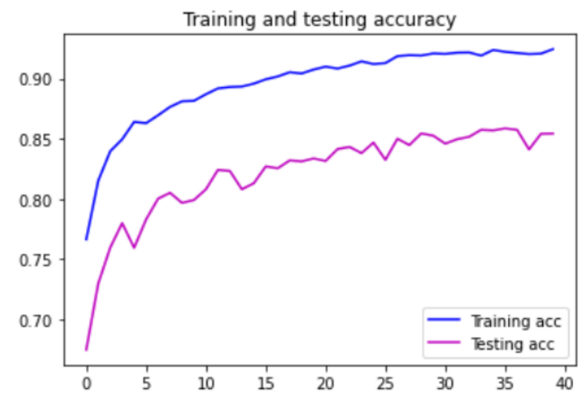


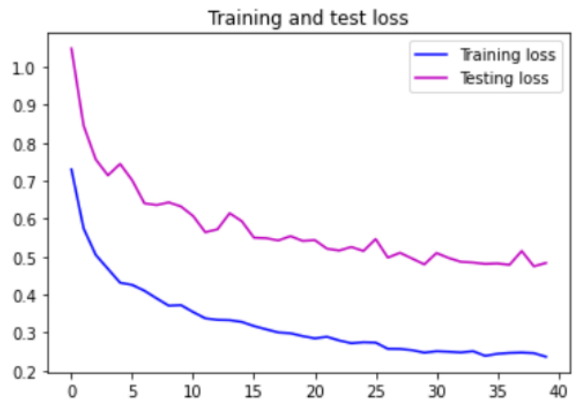


```
Train accuracy : 0.9778333306312561
Test accuracv  : 0.8973000049591064
```

Supervised by Mr Badre BOUSALEM

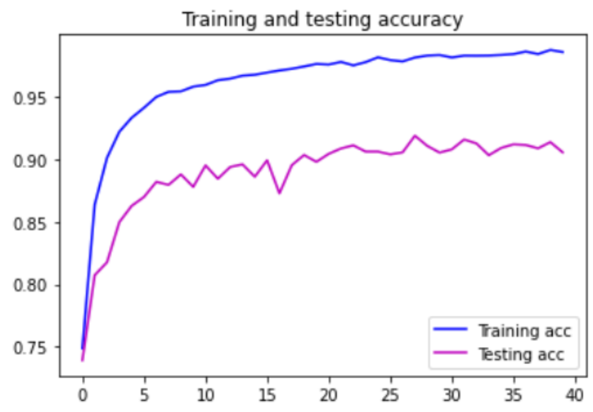
4) Fourth Model

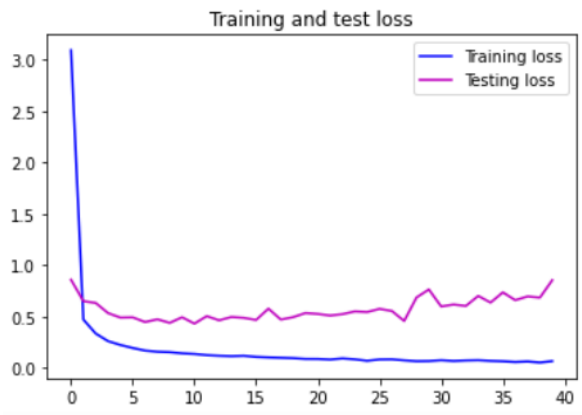




```
Train accuracy : 0.9244833588600159
Test accuracy  : 0.8543999791145325
```

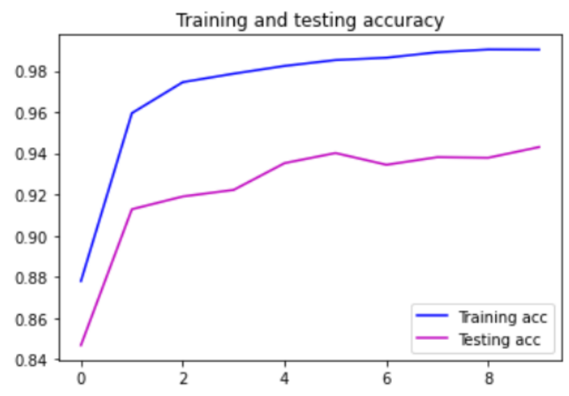
5) Fifth Model

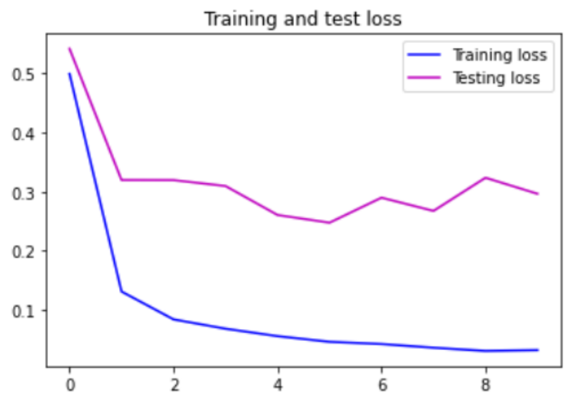




```
Train accuracy : 0.9857500195503235
Test accuracy  : 0.9054999947547913
```

6) Sixth Model





Supervised by Mr Badre BOUSALEM

```
Train accuracy : 0.9903166890144348
Test accuracy  : 0.9430000185966492
```

Conclusion

With a similar problem and different data, the 5th and 6th models are still able to perform very well although the accuracy decreases. Hence, the models we designed are satisfying though they can be enhanced.

3| Vegetable dataset

Presentation of the dataset

This dataset contains 10 classes which are described below :

```
['Bean' 'Bitter_Gourd' 'Bottle_Gourd' 'Brinjal' 'Broccoli' 'Cabbage'
 'Capsicum' 'Carrot' 'Cauliflower' 'Cucumber']
```

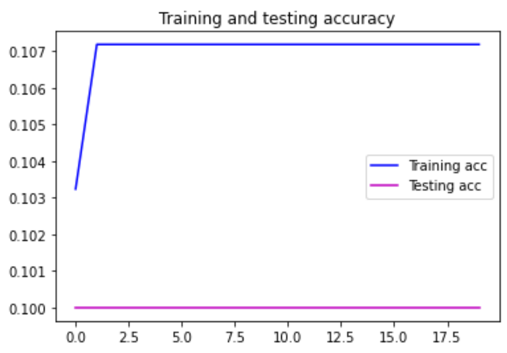
1) Model 1

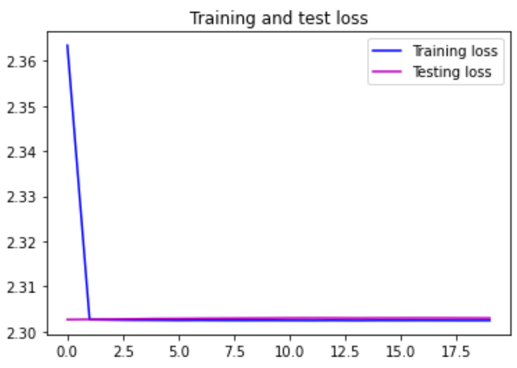
The first model is very simple. It has two layers and very few neurons.

```
model = Sequential()
model.add(layers.InputLayer(input_shape=(size[0], size[1], 3)))
model.add(layers.Rescaling(scale=1./255))

model.add(layers.Flatten())
model.add(layers.Dense(2, activation="tanh"))
model.add(layers.Dense(len(classes), activation='softmax'))

# Compilation du modèle
model.compile(optimizer='adam', metrics=['acc'], loss='categorical_crossentropy')
```

```
Train accuracy : 0.10718515515327454
Test accuracy  : 0.10000000149011612
```

As expected, the model is very bad. This is not surprising considering its constitution.





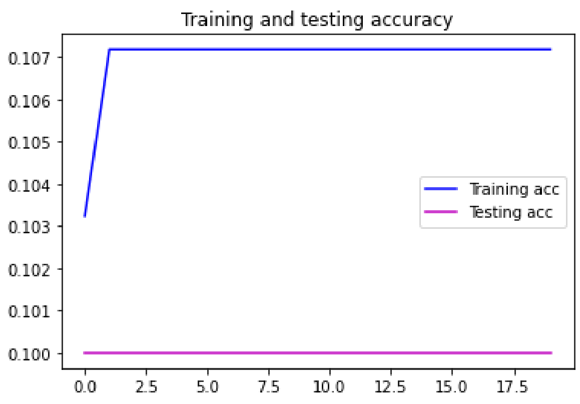
2) Model 2

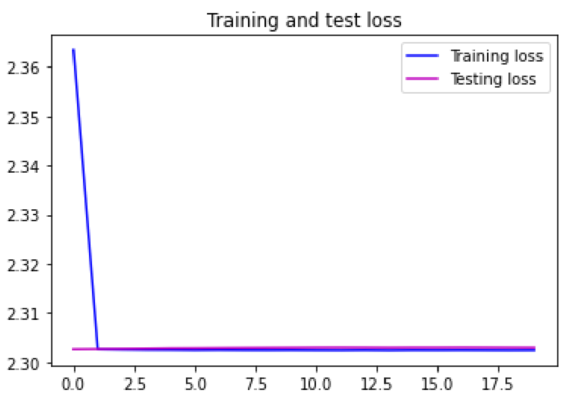
The second model that we tried is a simple 2 layers neural network. It is really basic and performs quite badly.

```
model = Sequential()
model.add(layers.InputLayer(input_shape=(size[0], size[1], 3)))
model.add(layers.Rescaling(scale=1./255))

model.add(layers.Flatten())
model.add(layers.Dense(16, activation="relu"))
model.add(layers.Dense(len(classes), activation='softmax'))

# Compilation du modèle
model.compile(optimizer='adam', metrics=['acc'], loss='categorical_crossentropy')
```

```
Train accuracy : 0.10718515515327454
Test accuracy  : 0.10000000149011612
```

The accuracy and loss are absolutely horrible. We can not even talk about overfitting or underfitting, it is simply bad.





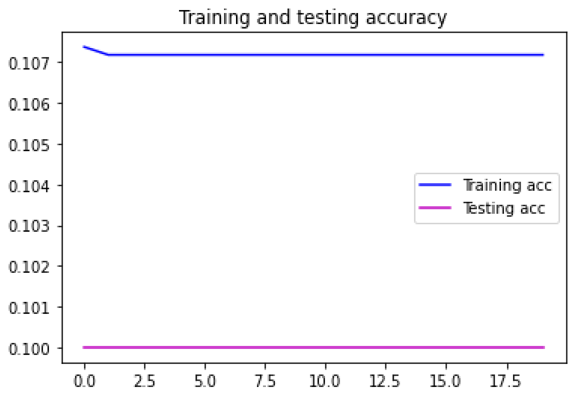
3) Model 3

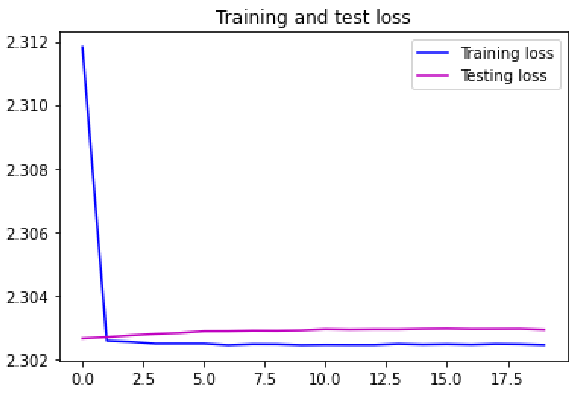
We added a new layer to see how it goes. However, it performs as horribly as the first model.

```
model = Sequential()
model.add(layers.InputLayer(input_shape=(size[0], size[1], 3)))
model.add(layers.Rescaling(scale=1./255))

model.add(layers.Flatten())
model.add(layers.Dense(16, activation="relu"))
model.add(layers.Dense(16, activation="relu"))
model.add(layers.Dense(len(classes), activation='softmax'))

# Compilation du modèle
model.compile(optimizer='adam', metrics=['acc'], loss='categorical_crossentropy')
```

```
Train accuracy : 0.10718515515327454
Test accuracy  : 0.10000000149011612
```

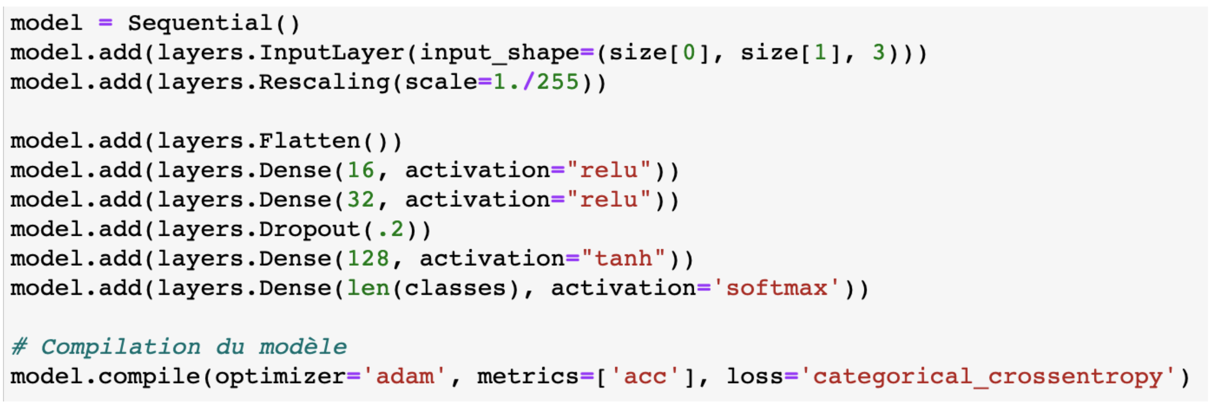
Again, no real comments can be made on the accuracy and loss. Actually, we get pretty similar results. This tells us that using dense layers here is not the right way.



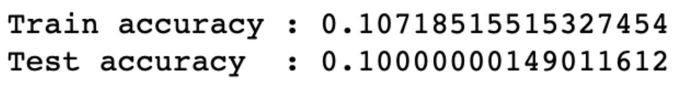


4) Model 4

The 4th model is a neural network with dense layers more elaborate than the previous ones. We added a dropout to reduce overfitting because we know that models overfit a lot on image datasets.

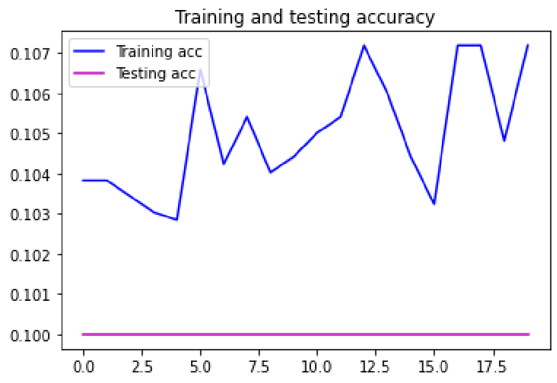


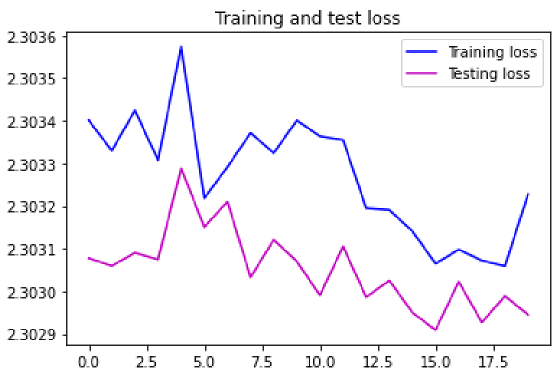
```
model = Sequential()
model.add(layers.InputLayer(input_shape=(size[0], size[1], 3)))
model.add(layers.Rescaling(scale=1./255))

model.add(layers.Flatten())
model.add(layers.Dense(16, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dropout(.2))
model.add(layers.Dense(128, activation="tanh"))
model.add(layers.Dense(len(classes), activation='softmax'))

# Compilation du modèle
model.compile(optimizer='adam', metrics=['acc'], loss='categorical_crossentropy')
```



```
Train accuracy : 0.10718515515327454
Test accuracy  : 0.10000000149011612
```

As can be seen, this model is also extremely bad. It is now time to use convolutional neural networks to solve this problem.





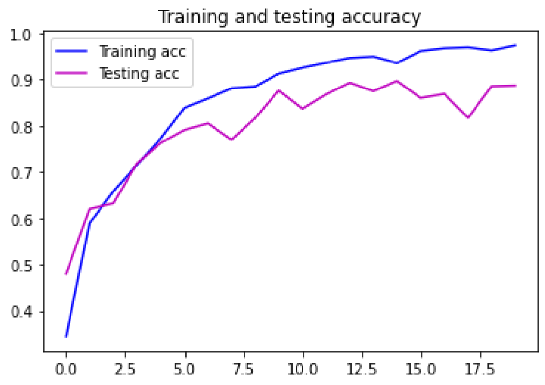
5) Model 5

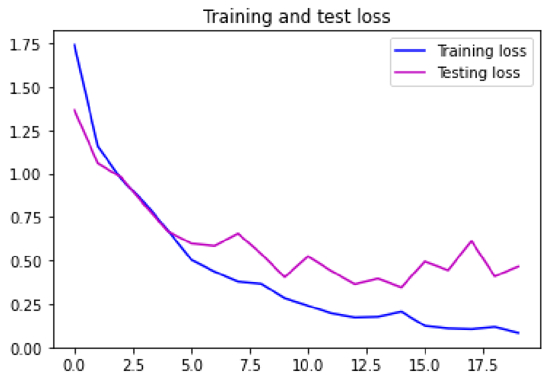
Model 5 is a convolutional neural network. Indeed, on images, dense networks tend to perform very poorly, as seen above. The number of parameters to train is huge (as the images are 64*64 pixels, that makes arrays of 4096 elements), which often leads to a very high overfitting. Fortunately, there is a solution: CNNs. They allow to reduce the size of the input arrays by processing the image through a kernel. We can then reuse dense layers to process the result.

```
model = Sequential()
model.add(layers.InputLayer(input_shape=(size[0], size[1], 3)))
model.add(layers.Rescaling(scale=1./255))

model.add(layers.Conv2D(50, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(20, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))


model.add(layers.Flatten())
model.add(layers.Dense(16, activation="relu"))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(len(classes), activation='softmax'))

# Compilation du modèle
model.compile(optimizer='adam', metrics=['acc'], loss='categorical_crossentropy')
```

```
Train accuracy : 0.9733517765998840
Test accuracy  : 0.8865000009536743
```

As we can see, the results are much better.





The accuracies are increasing while the losses are decreasing. However, we can see that the test accuracy is not as high as the train accuracy. And, the test loss is starting to stop decreasing at the 20th epoch, while the train loss continues to decrease. This is characteristic of an overfitting. We need to change our network to prevent this.

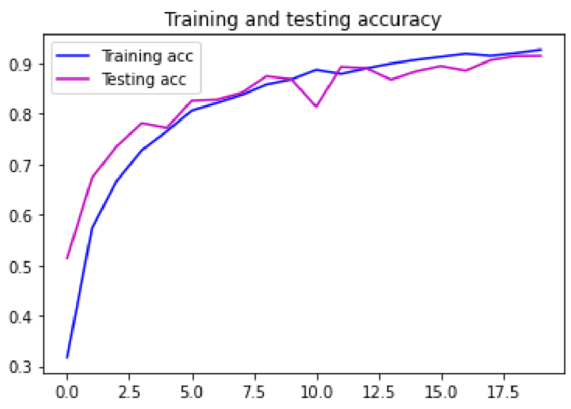
6) Model 6

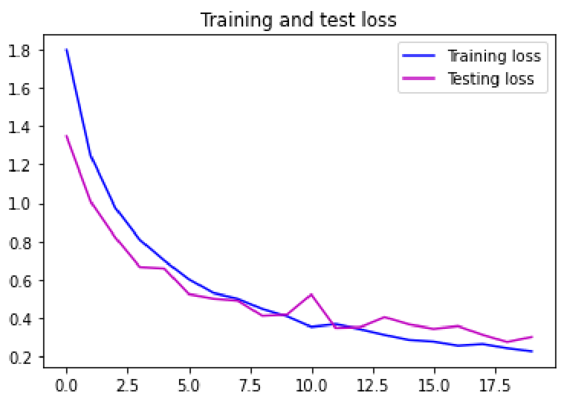
To combat overfitting, we added dropout layers (which remove a certain percentage of trained neurons). Also, we decreased the number of neurons per layer and the number of layers.

```
model = Sequential()
model.add(layers.InputLayer(input_shape=(size[0], size[1], 3)))
model.add(layers.Rescaling(scale=1./255))

model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(20, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(0.7))

model.add(layers.Flatten())
model.add(layers.Dense(16, activation="relu"))
model.add(layers.Dense(len(classes), activation='softmax'))

# Compilation du modèle
model.compile(optimizer='adam', metrics=['acc'], loss='categorical_crossentropy')
```

```
Train accuracy : 0.9269640445709229
Test accuracy  : 0.9150000214576721
```

This works quite well because we see a clear decrease in overfitting. The losses decrease together while the accuracies increase together. This model is very correct and allows satisfactory results.


Training and testing accuracy
Training acc
Testing acc


Training and test loss
Training loss
Testing loss