Report – Lab3

Task 2. Change the choice of instances for training, validation and testing. What accuracy did you obtain now ? How reliable was the first accuracy value ?

In task 2, the data of X_train and y_train need to be reselected. Since the original number of instances which is 60000 is increased to 70000, a higher accuracy value can be obtained than the original, which is about 0.8638.

```
X_train=X_max[10000:]
X_test=X_max[:10000]
X_train_full.shape
```

```
(60000, 28, 28)
```

```
y_train=y_max[10000:]
y_test=y_max[:10000]
y_train_full.shape
```

```
(60000,)
```

```
X_max=np.concatenate((X_train_full,X_test),axis=0)
X_max.shape
```

```
(70000, 28, 28)
```

```
y_max=np.concatenate((y_train_full,y_test),axis=0)
y_max.shape
```

```
(70000,)
```

```
history=model.fit(X_train,y_train,epochs=5,validation_data=(X_valid,y_valid))
```

```
Epoch 1/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.7049 - accuracy: 0.7671 - val_loss: 0.5370 - val_a
ccuracy: 0.8152
Epoch 2/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.4898 - accuracy: 0.8280 - val_loss: 0.4540 - val_a
ccuracy: 0.8458
Epoch 3/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.4409 - accuracy: 0.8441 - val_loss: 0.4470 - val_a
ccuracy: 0.8396
Epoch 4/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.4146 - accuracy: 0.8527 - val_loss: 0.4002 - val_a
ccuracy: 0.8624
Epoch 5/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3930 - accuracy: 0.8604 - val_loss: 0.3957 - val_a
ccuracy: 0.8584
```

```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 760us/step - loss: 0.3879 - accuracy: 0.8638

[0.3878987431526184, 0.8637999892234802]
```

Task 3. Experiment with the NN architecture: reduce number of neurons and/or number of layers, a couple of steps down to a really small NN. How is the accuracy affected ?

(1) Reduce number of neurons –-> Accuracy goes down to 0.8441

```
model = Sequential()
model.add(Flatten(input_shape=(28,28))) #input
model.add(Dense(200,activation="relu")) #Sigmoid / relu    300->200
model.add(Dense(50,activation="relu")) #layer    100->50
model.add(Dense(10,activation="softmax")) #output
```

```
history=model.fit(X_train,y_train,epochs=5,validation_data=(X_valid,y_valid))
```

```
Epoch 1/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.7476 - accuracy: 0.7520 - val_loss: 0.5182 - val_a
ccuracy: 0.8252
Epoch 2/5
1719/1719 [==============================] - 2s 954us/step - loss: 0.4996 - accuracy: 0.8253 - val_loss: 0.4427 - val
_accuracy: 0.8562
Epoch 3/5
1719/1719 [==============================] - 2s 980us/step - loss: 0.4534 - accuracy: 0.8410 - val_loss: 0.4212 - val
_accuracy: 0.8572
Epoch 4/5
1719/1719 [==============================] - 2s 967us/step - loss: 0.4256 - accuracy: 0.8499 - val_loss: 0.4204 - val
_accuracy: 0.8566
Epoch 5/5
1719/1719 [==============================] - 2s 958us/step - loss: 0.4068 - accuracy: 0.8570 - val_loss: 0.4089 - val
_accuracy: 0.8598
```

```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 634us/step - loss: 0.4398 - accuracy: 0.8441

[0.439820796251297, 0.8440999984741211]
```

(2) Reduce number of layers –-> Accuracy goes down to 0.8439

```
model = Sequential()
model.add(Flatten(input_shape=(28,28))) #input
model.add(Dense(300,activation="relu")) #Sigmoid / relu    300->200
#model.add(Dense(50,activation="relu")) #layer    100->50
model.add(Dense(10,activation="softmax")) #output
```

```
history=model.fit(X_train,y_train,epochs=5,validation_data=(X_valid,y_valid))
```

```
Epoch 1/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.7363 - accuracy: 0.7663 - val_loss: 0.5446 - val_a
ccuracy: 0.8248
Epoch 2/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.5137 - accuracy: 0.8261 - val_loss: 0.4753 - val_a
ccuracy: 0.8442
Epoch 3/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.4699 - accuracy: 0.8385 - val_loss: 0.4564 - val_a
ccuracy: 0.8458
Epoch 4/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.4437 - accuracy: 0.8477 - val_loss: 0.4444 - val_a
ccuracy: 0.8438
Epoch 5/5
1719/1719 [==============================] - 2s 1ms/step - loss: 0.4255 - accuracy: 0.8533 - val_loss: 0.4152 - val_a
ccuracy: 0.8618
```

```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 717us/step - loss: 0.4494 - accuracy: 0.8439

[0.4493940770626068, 0.8439000248908997]
```

(3) Reduce both number of neurons and layers –-> Accuracy goes down to 0.8195

```
model = Sequential()
model.add(Flatten(input_shape=(28,28))) #input
model.add(Dense(150,activation="relu")) #Sigmoid / relu     300->200
#model.add(Dense(50,activation="relu")) #layer     100->50
model.add(Dense(10,activation="softmax")) #output
```

```
history=model.fit(X_train,y_train,epochs=5,validation_data=(X_valid,y_valid))
```

```
Epoch 1/5
1719/1719 [==============================] - 2s 937us/step - loss: 0.7542 - accuracy: 0.7577 - val_loss: 0.5495 - val
_accuracy: 0.8212
Epoch 2/5
1719/1719 [==============================] - 2s 883us/step - loss: 0.5217 - accuracy: 0.8241 - val_loss: 0.4766 - val
_accuracy: 0.8382
Epoch 3/5
1719/1719 [==============================] - 2s 910us/step - loss: 0.4739 - accuracy: 0.8378 - val_loss: 0.4785 - val
_accuracy: 0.8346
Epoch 4/5
1719/1719 [==============================] - 2s 880us/step - loss: 0.4455 - accuracy: 0.8467 - val_loss: 0.4366 - val
_accuracy: 0.8496
Epoch 5/5
1719/1719 [==============================] - 2s 876us/step - loss: 0.4279 - accuracy: 0.8517 - val_loss: 0.4424 - val
_accuracy: 0.8424
```

```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 615us/step - loss: 0.4836 - accuracy: 0.8195

[0.48359552025794983, 0.8195000290870667]
```

Task 4. A common question when deciding to work with NN is if one has enough data ? Test this by picking only a tenth of the available data, decrease the NN architecture and play around to see what accuracy you can obtain. Was it necessary to have as much data available as the MNIST fashion dataset ?

The more data, the better the prediction, but when the training sample size is large and the network layer is too small, the feature training will be insufficient. So the premise is that the feature extraction ability of the network should not be too bad (neural network capacity problem).

```
X_train=X_max[:10000]
#X_test=X_max[:10000]
X_train_full.shape
```

```
(60000, 28, 28)
```

```
y_train=y_max[:10000]
#y_test=y_max[:10000]
y_train_full.shape
```

```
(60000,)
```

```
model = Sequential()
model.add(Flatten(input_shape=(28,28))) #input
model.add(Dense(150,activation="relu")) #Sigmoid / relu     300->200
#model.add(Dense(50,activation="relu")) #layer     100->50
model.add(Dense(10,activation="softmax")) #output
```

```
history=model.fit(X_train,y_train,epochs=5,validation_data=(X_valid,y_valid))
```

```
Epoch 1/5
1719/1719 [==============================] - 2s 967us/step - loss: 0.7415 - accuracy: 0.7623 - val_loss: 0.5413 - val
_accuracy: 0.8220
Epoch 2/5
1719/1719 [==============================] - 2s 907us/step - loss: 0.5173 - accuracy: 0.8243 - val_loss: 0.4867 - val
_accuracy: 0.8384
Epoch 3/5
1719/1719 [==============================] - 2s 905us/step - loss: 0.4729 - accuracy: 0.8372 - val_loss: 0.4484 - val
_accuracy: 0.8480
Epoch 4/5
1719/1719 [==============================] - 2s 882us/step - loss: 0.4481 - accuracy: 0.8467 - val_loss: 0.4244 - val
_accuracy: 0.8570
Epoch 5/5
1719/1719 [==============================] - 2s 888us/step - loss: 0.4302 - accuracy: 0.8512 - val_loss: 0.4298 - val
_accuracy: 0.8514
```

```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 622us/step - loss: 0.4687 - accuracy: 0.8345

[0.46867451071739197, 0.8345000147819519]
```

By using only 10% of the data set for training and reducing the neural network architecture, it can be found that the accuracy value is reduced to 0.8345.