T-DeLearn PW10

- Romain Capocasale
- Jean Demeusy

Exercice 1

A

Summary of sequential architecture :

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 10)                20490
=================================================================
Total params: 39,882
Trainable params: 39,882
Non-trainable params: 0
_________________________________________________________________
```

Score of sequential :

```
Test loss: 0.8561450242996216
Test accuracy: 0.714900016784668
```

Summary of functional architecture :

```
visible = Input(shape=X_train.shape[1:])
conv1 = Conv2D(32, kernel_size=(3,3), padding="same", activation="relu")(visible)
conv2 = Conv2D(32, kernel_size=(3,3), padding="same", activation="relu")(conv1)
pool1 = MaxPooling2D(2)(conv2)
conv3 = Conv2D(32, kernel_size=(3,3), padding="same", activation="relu")(pool1)
pool2 = MaxPooling2D(2)(conv3)
flat = Flatten()(pool2)
output = Dense(10, activation="softmax")(flat)

model = Model(inputs=visible, outputs=output)
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 32)        9248
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 8, 8, 32)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                20490
=================================================================
Total params: 39,882
Trainable params: 39,882
Non-trainable params: 0
_________________________________________________________________
```

Score of functional :

```
Test loss: 0.9130173603057862
Test accuracy: 0.7125999927520752
```

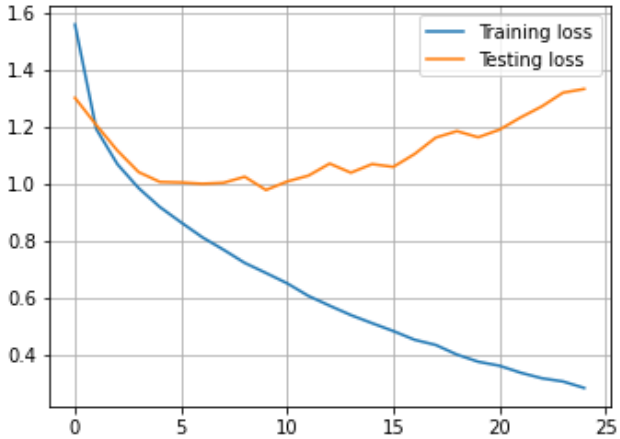
We can can that, the two model have the same number of parameters and the scores is slightly almost similar.

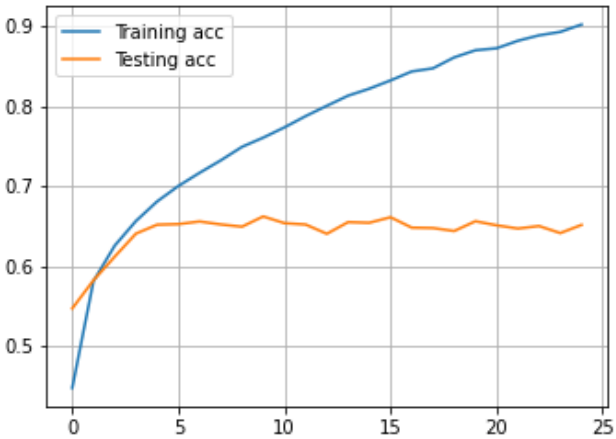
B

Model	Architecture description	Acc. train	Acc. test
1 - multiple path	BRANCH1:CONV(D=32, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; BRANCH2:CONV(D=32, w=h=6, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; CONCAT(BRANCH1, BRANCH2); DENSE(100, act=relu); DENSE(10, softmax)	0.9020	0.6517

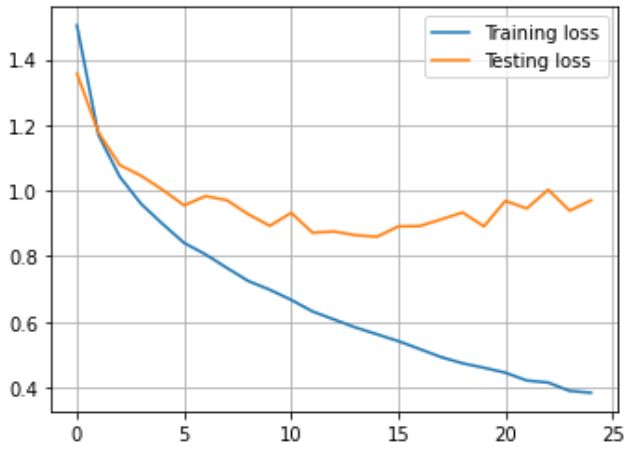
Model	Architecture description	Acc. train	Acc. test
2 - multiple features	BRANCH1:CONV(D=32, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; BRANCH2(BRANCH1):CONV(D=32, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; BRANCH3(BRANCH2):CONV(D=32, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; CONCAT(BRANCH1, BRANCH2, BRANCH3); DENSE(100, act=relu); DENSE(10, softmax)	0.8658	0.6945
3 - multiple features	BRANCH1:CONV(D=32, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; BRANCH2(BRANCH1):CONV(D=32, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; BRANCH3(BRANCH2):CONV(D=32, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; CONCAT(BRANCH1, BRANCH2, BRANCH3); DENSE(64, act=relu); DROPOUT(0.2); DENSE(10, softmax)	0.8216	0.7153
4 - multiple path	BRANCH1:CONV(D=32, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-CONV(D=64, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; BRANCH2(BRANCH1):CONV(D=32, w=h=3, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-CONV(D=64, w=h=6, S=1, P=same, act=relu)-DROPOUT(0.2)-MAXPOOL(S=1, size=2)-FLATTEN; CONCAT(BRANCH1, BRANCH2); DENSE(50, act=relu); DROPOUT(0.2); DENSE(10, softmax)	0.6765	0.6891

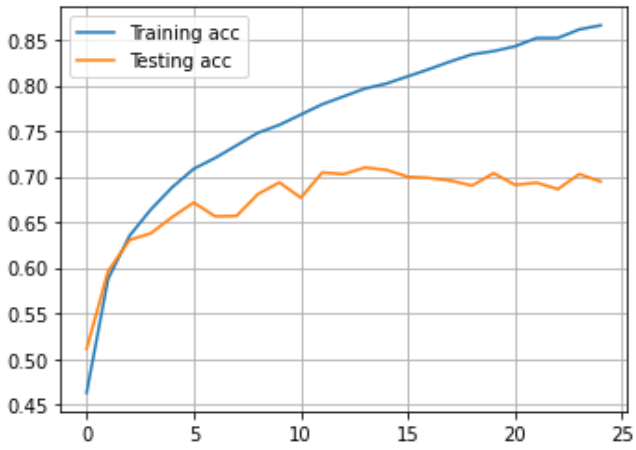
Model 1 Loss/Accs :



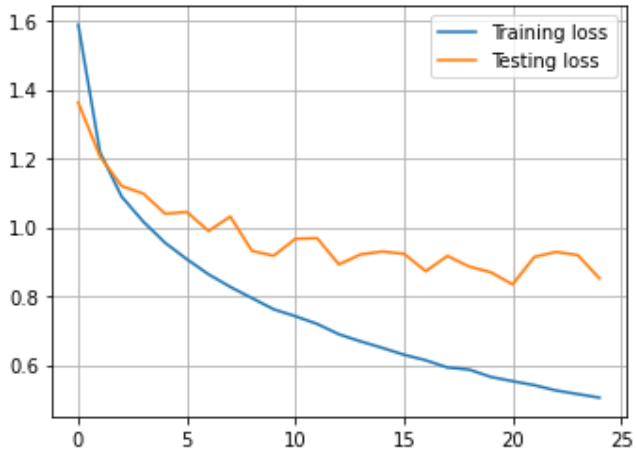


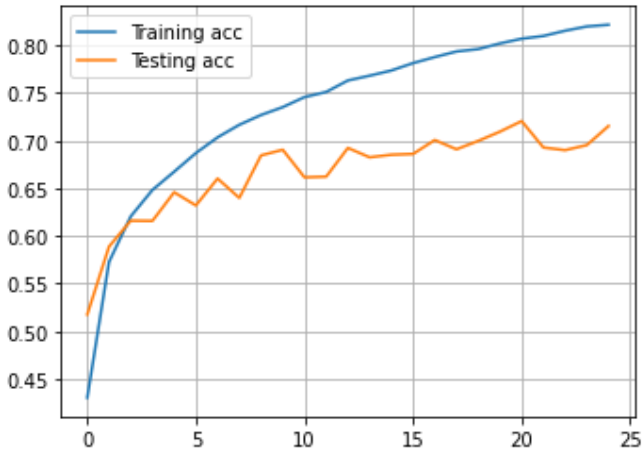
Model 2 Loss/Accs :



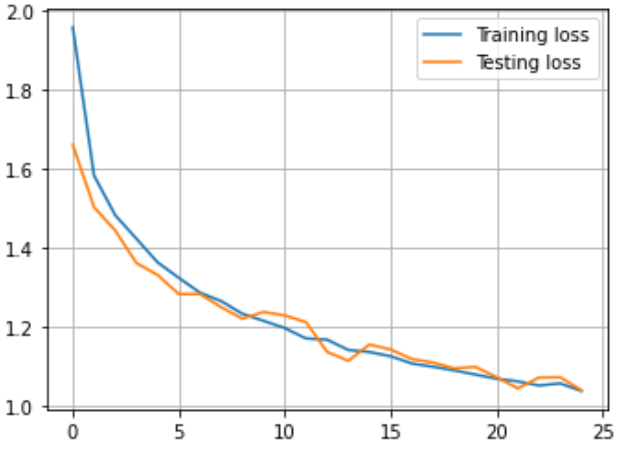


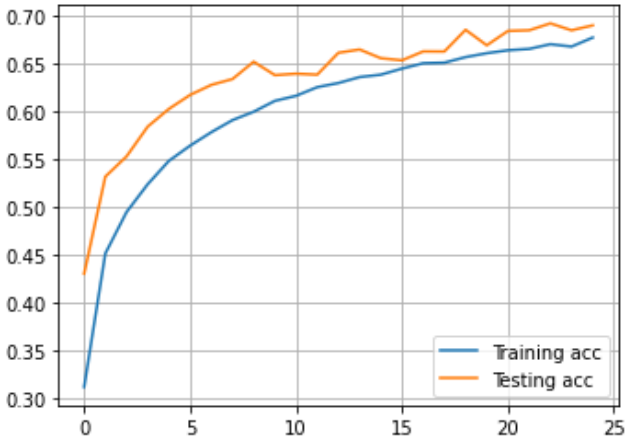
Model 3 Loss/Accs :





Model 4 Loss/Accs :





Discussion

It can be seen that the model with the best accuracy is model number 3 with multiple features. We also see that the first 3 models overfit, this problem can be avoided by using data augmentation which has not been implemented. The 4th model overfits the least and seems the most stable.

We notice with this lab and the 2 previous ones that the accuracy of the models is around 0.70% and cannot be significantly exceeded despite all attempts.

Exercice 2

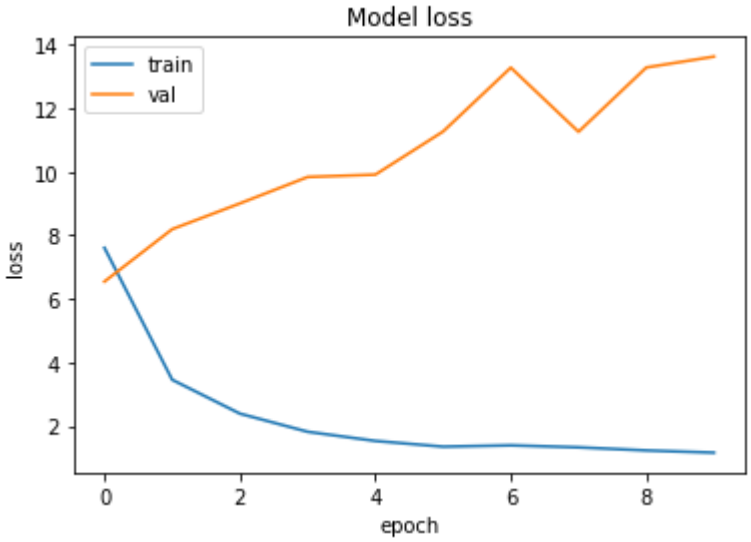
The base model chosen is MobileNet.

Architecture 1

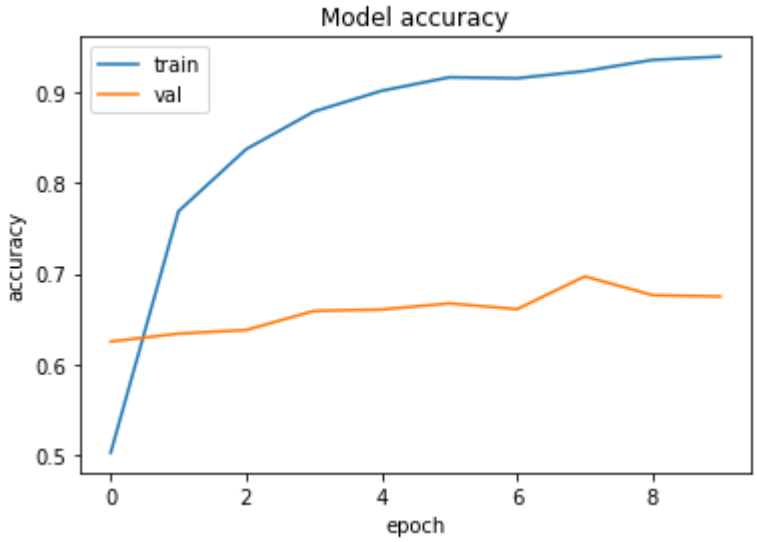
Hyperparameters:

- Learning Rate : 0.001
- Batch Size = 128
- Epoch = 20

Model	Architecture description	Acc. train	Acc. test
1	MobileNet - Flatten - Dropout(0.5) - Dense(20)	0.94	0.68



Loss architecure 1:



Accuracy architecture 1:

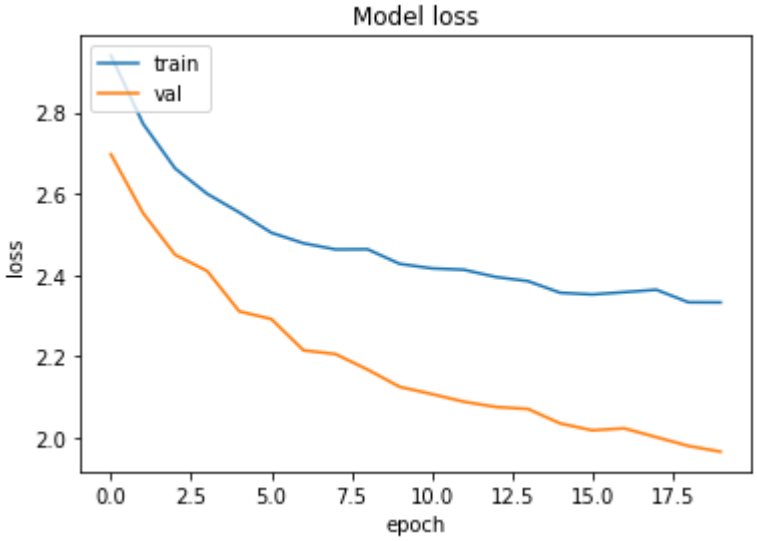
We can see that despite the performance of the model, it overfits a lot and will not be retained for the fine-tuning phase.

Architecture 2

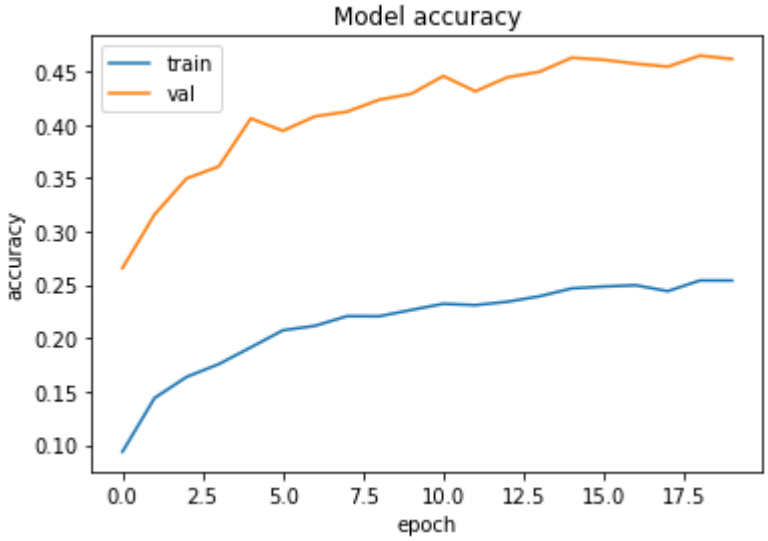
Hyperparameters:

- Learning Rate : 0.001
- Batch Size = 128
- Epoch = 20

Model	Architecture description	Acc. train	Acc. test
2	MobileNet - Flatten - Dropout(0.5) - Dense(32) - Dropout(0.5) - Dense(20)		



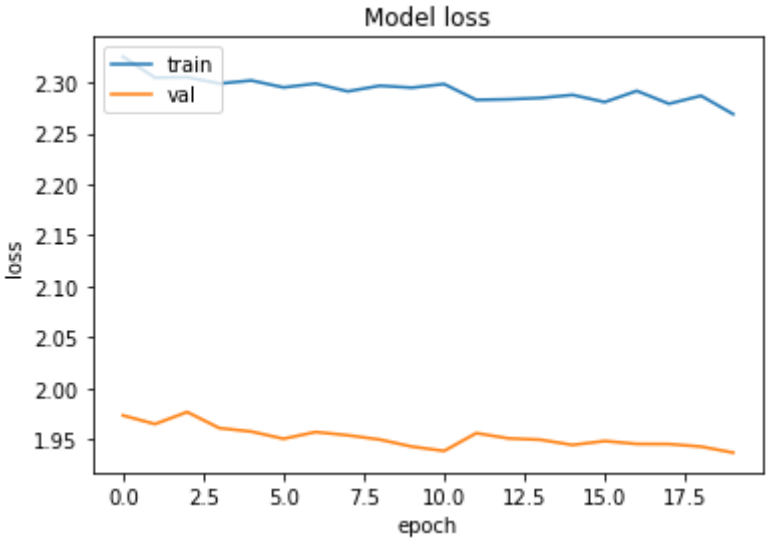
Loss architecure 2:



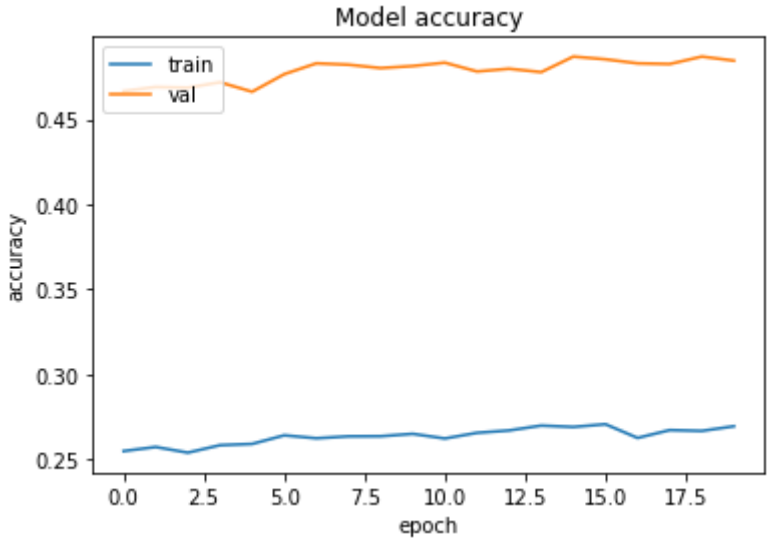
Accuracy architecture 2:

Hyperparameters fine tuning:

- Learning Rate : 0.00000001
- Batch Size = 128
- Epoch = 20



Loss fine-tuning architecure 2:



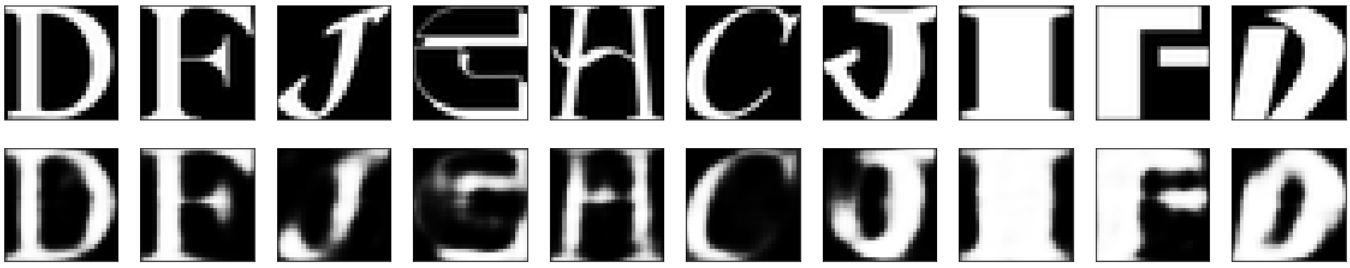
Accuracy fine-tuning architecture 2:

Discussion

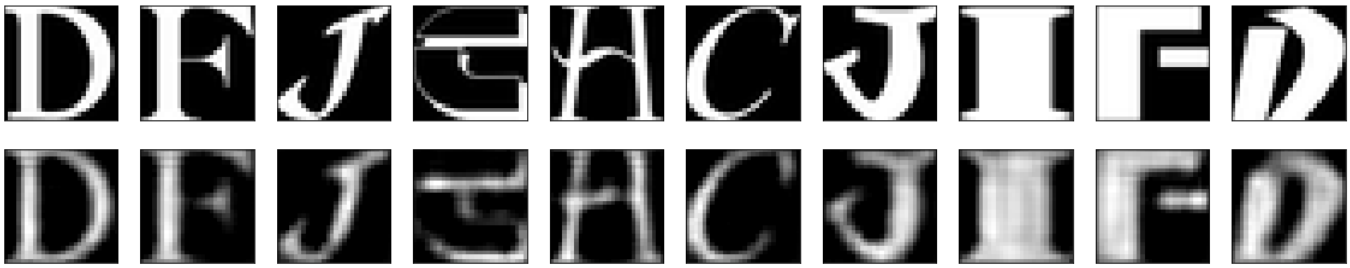
We can see that despite the fact that the first architecture gives better results it overfits a lot. The second architecture gives less good results but overfits less. This second model is therefore more stable and has been retained to perform the finetuning. We can see that the finetuning has only slightly improved the score of the model. The accuracy has gone from about **0.4614** to **0.4845**.

Exercise 3

Input and output of the autoencodeur:



Prediction with autoencodeur:



f) We could train a classifier on the input data. The model obtained is then saved, and used with the images obtained with the decoder. Then the performance could be estimated with the evaluation of recall, f1scor or precision.

g) With the SVM, we get a calssification accuracy of 90%. SVM results:

```
Classification report for classifier SVC(cache_size=7000, kernel='linear'):
              precision    recall  f1-score   support

           0       0.91      0.90      0.90      1000
           1       0.90      0.86      0.88      1000
           2       0.91      0.92      0.91      1000
           3       0.91      0.90      0.91      1000
           4       0.90      0.88      0.89      1000
           5       0.90      0.94      0.92      1000
           6       0.88      0.88      0.88      1000
           7       0.91      0.89      0.90      1000
           8       0.86      0.86      0.86      1000
           9       0.87      0.93      0.90      1000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

Denoising with autoencodeur (noise_level = 0.3):