Sequence 6
Convolutional Neural Networks

Machine Learning - 5PMBMLD0

Allan DIZET - Matteo MARENGO

November 18, 2022

1 Introduction

The convolutional neural network (CNN) is a special type of neural network model. It has a convolutional layer. This layer performs a convolution. It means that the layer will apply a specific filter to the inputs in oder to extract specific features related to the images to create a feature map. Then, we apply a classic neural network.

2 CNN for object classification

2.1 Designing a basic network based on VGG structure

The VGG structure consists of convolutional layers : the convolution consists of filters 3x3 that performs element-wise multiplication. The size of the matrix decreases as we keep on applying filters on the obtained matrix. MaxPooling is used to reduce the size of the image. MaxPooling is to select the max value from the matrix with the specified size (2x2 in our case).

```
# Two convolutionnal blocks with Relu activation functions

# First block : 2 * 32 convolution filters of size 3x3 + 2x2 MaxPooling
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Second block: 2 * 64 convolution filters of size 3x3 + 2x2 MaxPooling
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


# Neural network for classification with one hidden layer
# flatten to transform the 2D maps into a 1D vector for NN:
# => input layer of size 32x32x3 = 3072
model.add(Flatten())
#hidden layer with 128 nodes
model.add(Dense(128, activation='relu'))
# output layer with 10 nodes because of the 10 classes
model.add(Dense(10, activation='softmax'))
```

2.1.1 Performances of the baseline model

We design the CNN and we run the programme with batch size = 32, epoch = 20 and SGD as the optimizer. We obtain the evolution of the loss and accuracy functions along with the iteration in Fig 1.

SGD :

Train accuracy : 84.008%

Test accuracy : 67.600%

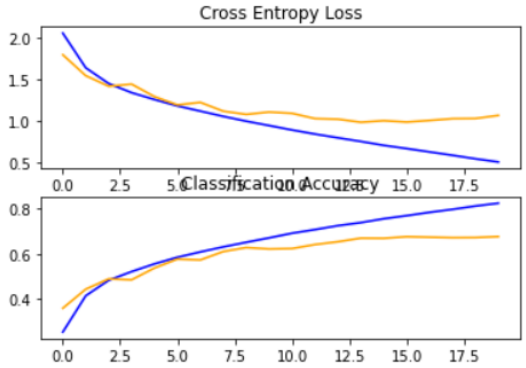

Figure 1: Loss and accuracy functions with the iterations - SGD Optimizer: train is in blue and test is in yellow

The two curves (train and test) are quite distant (deviation of approximately 20 %), therefore we can conclude that the model is not so good, and we might have to tune some parameters.

2.1.2 Changing the optimizer - Adam optimizer

We change our optimizer from SGD to Adam, and we plot the functions.

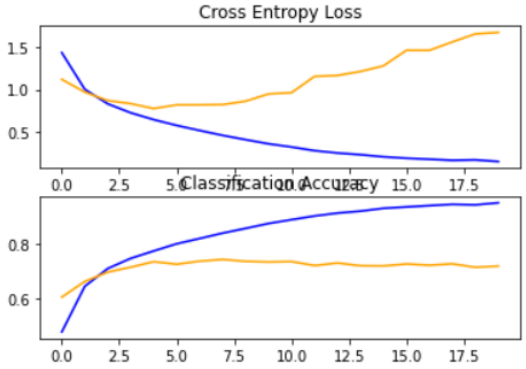

Figure 2: Loss and accuracy functions with the iterations - Adam Optimizer

Adam optimizer :
Train accuracy : 97.718%
Test accuracy : 71.980%

The accuracy percentages are higher than with the SGD optimizer and the Train accuracy is extremely high (98 %). However, there is still a huge gap with the test accuracy (20 %), the two curves diverge a lot one from the other. It means that there is overfitting during the training process. Adam is the abbreviation of adaptive moment estimation. Adam optimizer updates the learning rate for each network weight individually (whereas with SGD the learning rate is constant).

2.1.3 Introducing a batch normalization (BN)

Now, we introduce a batch normalization step between the two first CONV layers and the two second CONV layers and just before the first fully connected layer.

```
model.add(BatchNormalization())
```

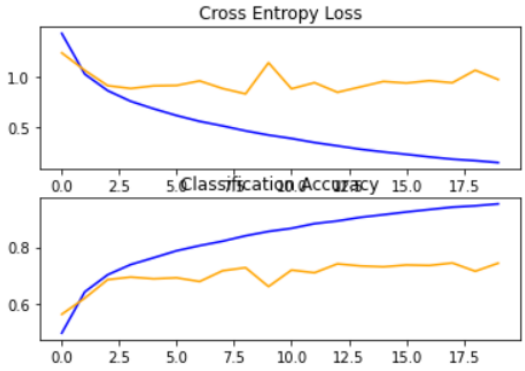

Figure 3: Loss and accuracy functions with the iterations - Normalization - SGD

SGD + BatchNormalization :
Train accuracy : 97.638%
Test accuracy : 74.380%

Once again, the performances of the training are good but not the testing part. It means that this model is still overfitting. We have to include regularization technics to our model.

2.2 Improved model with regularization technics

2.2.1 Early stopping

With the first CNN model it seems that to stop around 15 epochs seems good to limit overfitting.
SGD + EarlyStopping :
Epoch : 25
Train accuracy : 88.032%
Test accuracy : 65.090%

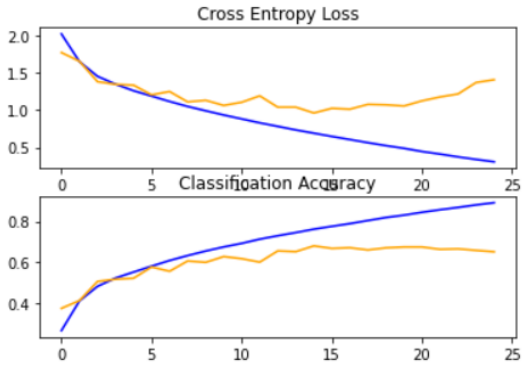

Figure 4: Confusion Matrix

Our statement was correct as it stops after 25 epochs (so during 10 consecutives epochs, the curve increased).

2.2.2 Dropout

To solve the problem of overfitting, we introduce a regularization process and especially the dropout technique. (It will drop nodes out of the network during the training step). We introduce a new hyper-parameter, the dropout probability.

```
model.add(Dropout(.2))
```

SGD + Dropout : Train accuracy : 86.690Test accuracy : 75.540

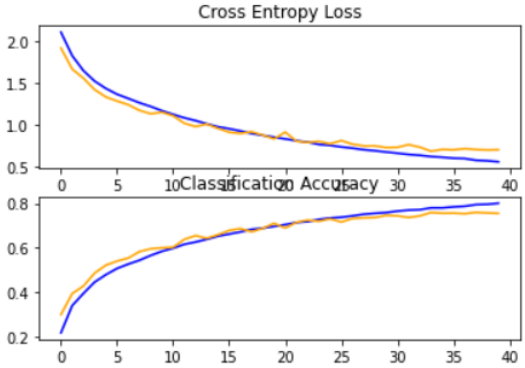

Figure 5: Confusion Matrix

There we have quite high accuracies and tests and train plots are next to each other meaning that there is not overfitting, this regularization technique is therefore interesting.

2.2.3 Data augmentation

The final technique to avoid overfitting is to do Data augmentation. We had training data by doing small random modifications on training samples. On the images, for example, we can include horizontal flips, some shifts of the image.

```
# create data generator
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1,
                          horizontal_flip=True)
# prepare iterator
it_train = datagen.flow(trainX, trainY, batch_size=32)
# fit model
callbacks = [EarlyStopping(monitor='val_loss', patience=10)]
print('Model training...')
steps = int(trainX.shape[0] / 64)
history = model.fit_generator(it_train, steps_per_epoch=steps, epochs=100,validation_data=(testX,
    testY), verbose=1,callbacks=callbacks)
```

SGD + Data Augmentation : (given in the handout)

Train accuracy : 92%

Test accuracy : 82%

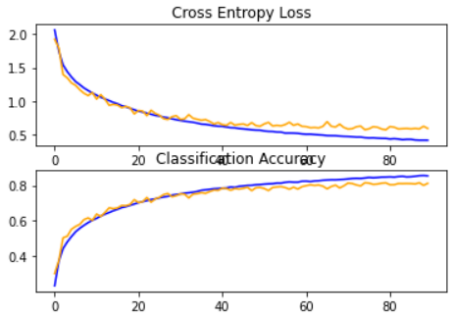

Figure 6: Loss and accuracy functions with the iterations - Adam Optimizer

SGD + Data augmentation : (found by us)
90 Epochs
Train acc= 89.064%
Test acc= 81.180% (Accuracy is a little bit different from what it is depicted in the handbook as it is a random process to do the data augmentation).
It is even better than the dropout technique as there is no overfitting and the accuracies are high. We found relatively close accuracy to what was indicated in the handbook.

2.3 Performance summary

Model	Training Accuracy (%)	Testing accuracy (%)	Overfitting (Y/N)
SGD	84.008	67.600	Y
Adam	97.718	71.980	Y
SGD + BatchNormalization	97.638	74.380	Y
SGD + EarlyStopping	88.032	65.090	Y
SGD + Dropout	86.690	75.540	N
SGD + Data Augmentation	92	82	N

2.4 Saving the model and making prediction

According to SGD + Dropout model, the sample image belong to the class 4, which correspond to deer. According to our brain, it would be legitimate to confirm this result.

3 Conclusion

We observe that CNN is a great solution to classify images. However, we have to be aware that overfitting can occur quite often and that several regularization techniques have to be applied in order to reduce it. Moreover, it will increase the overall accuracies, so it is two advantages at once.