Deep Learning - Unit 3 Assignments summary

Daniel Mínguez Camacho
Javier de la Rúa Martinez

May 14th, 2019

Contents

1 Introduction

This report is provided as the description of the work developed during the different deadlines of the Deep Learning course assignments, Unit 3, at Universidad Politecnica de Madrid. These assignments are related with computer vision. We have developed through them different models based on convolutional neural networks to solve the problems proposed.

In the introduction we will present the problem we are trying to solve, then we will go through the different deadlines we had, explaining the problems proposed and the steps followed to achieve their solution. After this, we will explain how we structured the final assignment and our results. Finally we have included dome personal comments and a short conclusion.

1.1 Problem proposed

In this practical exercise we had to solve a series of computer vision problems using deep neural networks models. The problem proposed was sign recognition, more precisely, single-image detection of traffic signs in natural (outdoor) images for real-world driving.

These problem arise because despite of human accuracy recognizing traffic signs, drivers attention is regularly drawn to different tasks and situations. Image classification can help in future applications in driving.

The goals of these assignments were:

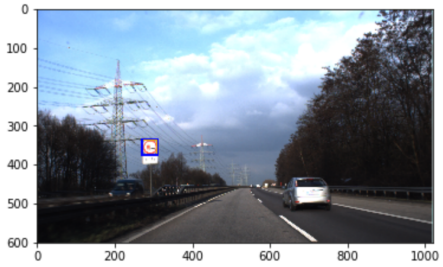
- Develop a feed-forward neural network to classify images that represents traffic signals.
- Improve the results, by developing a convolutional neural network from scratch using keras.
- Improve the results, including transfer learning in the model.
- Apply the models developed to the boxes generated by an object recognizer for signals detection in images.

Finally, we had some more time to improve the results obtained in the last part by using the feedback provided in class.

We also provide in the references some papers that address the problem we were facing.

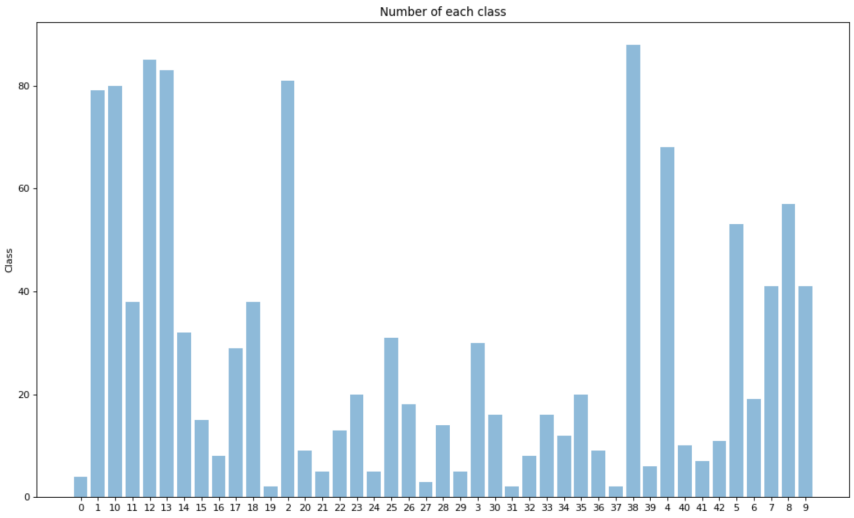
1.2 Data used

The data used for this set of assignments was the German Traffic Sign Recognition Benchmark (GTSRB) dataset, this dataset includes 900 labeled images with 43 classes of different signs. It was used first at IJCNN 2011 (International joint conference on Neural Networks). The images are annotated using ROI (Region of interest) boxes. Here it is possible to see an image and its ROI marked in blue:



The dataset involves prohibitory signs like "speed limit" signs (12), danger like "uneven road" (15), mandatory like "go right" (8) and other like "stop" (8). It is possible to see the different classes below:

The barplot of the different signs present in the dataset also shows that there are some signs more frequent than others:



The images were downloaded from the official webpage detailed in the references.

2 ffNN - Assignment 1

For this first approach we were requested to develop a feed forward neural network for solving the sign recognition problem. We had to select the ROI that represented the ground-truth of the different images, divide these ROI in train, development and test sets and use a feed forward deep neural network (without convolutional layers) to solve the problem. We had to decide over some of the parameters like:

- The number of layers and number of units in each layer.
- The optimization parameters to train the net.
- When to stop training according to the evolution of training during the optimization.

As a first approach, we decided to use the Infrastructure-as-a-Service offering of Google Cloud to run our scripts remotely and take an advantage of the benefits of the cloud.

For that purpose, we followed the steps in the material provided in class and performed additional steps which include:

- Create a new project.
- Create an instance of Compute Engine.
- Create a static IP and attach it to the instance.
- Create a firewall rule to allow TCP traffic through the 8888 port (jupyter notebook) and check rules for TCP traffic through the 22 port (ssh).
- Add an RSA certificate to be able to connect to the instance remotely from our laptops.

The initial configuration of the mentioned instance was 1vCPU (Intel Xeon 2.2GHz), 3.75GB RAM, GPU Tesla K80 and 10GB of disk.

Later, we decided to try to Google Colab, which had similar features, in order to avoid tuning steps to find the best balance between the hardware of the instance, cost and time availability. In this platform we were provided with 1-2vCPU (Intel Xeon 2.3GHz), 10GB RAM, TPU and 60GB of disk.

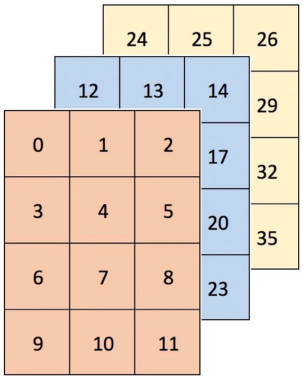
However, since Google Colab is a Software-as-a-Service product used through a web browser, specially in the free version, there were some issues in long-term computations (>40min) that end up eventually in disconnections and process losing.

Finally, we opted to prepare a simple virtual environment in our laptop by using Anaconda and run the scripts locally. Regarding the hardware, our laptop was provided with 6-12vCPU (Intel Core i7-8750H), 16GB RAM and NVIDIA GTX 1050 Ti. By using our laptop, we saw that we could speed up execution times and avoid additional management processes.

2.1 Process followed

First of all, we set up the environment (we decided to work in a local environment as we stated before) by installing Anaconda and creating a new separate environment with tensorflow-gpu and keras-gpu 2.1.6 libraries. We also installed other relevant libraries like opencv or numpy.

After that, we downloaded the dataset and created the train, development and test set as in the notebook provided. The input of our network would be the 3 dimensional rgb matrix divided by 255 (the maximum of the rgb) in order to normalize the values:



The output would be a vector of size 43 (one per class label) indicating the relevance of each class.

We decided to start with a relative simple architectures in order to get familiar with the code structure and the process of creating the NN:

- We experimented with different **batch sizes** (8,16,32), we noted that the training was relatively faster the bigger the batch size was (less iterations per epoch).
- We tried different **epochs** in all the models, here we checked that almost all of them needed around 50 to achieve their best results. We also discovered that we could stop a model manually by stopping the notebook and we could continue the training by executing the same code again. This helped us in the following assignments.
- The **architectures** we tried were relatively small and simple:
  - Dense (8) – Dropout() – Dense(8) – Dropout() – Flatten() – Dropout()
  - Dense(8-16) – Dropout() – Dense(8-16) – Dense(8-16) – Flatten() - Dropout()
  - Dense(8) – Dense(8) – Dense(8) – Dropout() – Flatten
  - ...
- **Activation**: we used Relu for input and hidden layers and Softmax for output layer. Here we did not choose different for the different examples because in the previous DL assignment they were the ones that worked better in our model.

- **Optimization**: Stochastic gradient descent optimizer. The parameters we used were $Learning\,rate = 0.001$, $decay = 1e{-}6$ and $nesterov = True$.

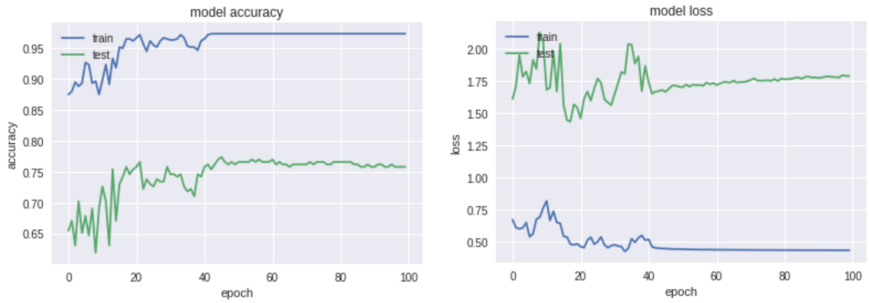
In the end we submitted the following architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 224, 224, 8)       32
_________________________________________________________________
dense_6 (Dense)              (None, 224, 224, 8)       72
_________________________________________________________________
dense_7 (Dense)              (None, 224, 224, 8)       72
_________________________________________________________________
flatten_2 (Flatten)          (None, 401408)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 401408)            0
_________________________________________________________________
dense_8 (Dense)              (None, 43)                17260587
=================================================================
Total params: 17,260,763
Trainable params: 17,260,763
Non-trainable params: 0
_________________________________________________________________
```

2.2 Results

The final results we obtained with the previous architecture were the following:

```
MLP took 3.5889580249786377 seconds
Test loss: 1.3541055078981985 - Accuracy: 0.8310249309130323
```



As we can see the model stabilize around epoch 45 with an accuracy of 0.83. After the submission we noted in class the we used models that were too simple and because of that we were not able to achieve better results. We didn´t try to generate more images or change the shape of the images neither (in order to improve performance). We tried to change this for the next submissions.

3 CNN - Assignment 2

In the previous assignment we saw that a feed forward neural network can achieve relative good results (we obtained 0.83 accuracy and other colleagues more), but in order to improve these results we needed to use CNN networks.

This is why in this assignment we were requested to create a CNN from scratch using keras, in this case we were requested to decide on different parameters like:

- Architecture of the network.
- Performance improvement (regularization, data augmentation, etc.)
- Weights optimization.

It was noted that we were not allowed to use transfer learning for this assignment. The teacher also proposed us to find a fully convolutional network in order to do not use any flatten layer, we tried to find. We were close but according to the teacher our approach was not totally correct.

3.1 Process followed

Having the environment already set up for the previous assignment, we first decided to diminish the shape of the signs to 128x128 in order to improve performance (like some of our classmates did). We also tried to reduce the height to 500 and some other configurations (like 32x32). The final configuration was height 500 and shape 128x128.

In order to have the fully convolutional network we decided to add a sequence of the following layers:

- Zero_padding
- Convolution 2d
- Batch normalization
- Max pooling 2d

So we started with an input with shape 128x128x3 and after each block we reduce the shape by a factor of 2, while increasing the depth, so after 6 blocks we arrive to a layer with shape 4x4x43, one 4x4 box for each class, here we decided to calculate the global average pooling in order to have our final layer with a softmax activation.

The final shape of the network was the following:

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 128, 128, 3)	0
zero_padding2d_1 (ZeroPaddin	(None, 130, 130, 3)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	896
batch_normalization_1 (Batch	(None, 128, 128, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 64, 64, 32)	0
zero_padding2d_2 (ZeroPaddin	(None, 66, 66, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18496
batch_normalization_2 (Batch	(None, 64, 64, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 32, 32, 64)	0
zero_padding2d_3 (ZeroPaddin	(None, 34, 34, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	73856
batch_normalization_3 (Batch	(None, 32, 32, 128)	512
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 128)	0
zero_padding2d_4 (ZeroPaddin	(None, 18, 18, 128)	0
conv2d_4 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_4 (Batch	(None, 16, 16, 128)	512
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 128)	0
zero_padding2d_5 (ZeroPaddin	(None, 10, 10, 128)	0
conv2d_5 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_5 (Batch	(None, 8, 8, 256)	1024
max_pooling2d_5 (MaxPooling2	(None, 4, 4, 256)	0
zero_padding2d_6 (ZeroPaddin	(None, 6, 6, 256)	0
conv2d_6 (Conv2D)	(None, 4, 4, 43)	99115
global_average_pooling2d_1 (	(None, 43)	0
activation_1 (Activation)	(None, 43)	0

Total params: 637,547
Trainable params: 636,331
Non-trainable params: 1,216

Since it was recommended we also tried data augmentation, we used the keras function *ImageDataGenerator* with the following parameters:

- width_shift_range=0.05
- height_shift_range=0.15
- zoom_range=0.15
- fill_mode='nearest'

We also experimented with other parameters like zca_withening, featurewise_center, featurewise_std_normalization or shear_range, but the results were not as good as expected or the images were not recognisable. We include some example of the results of data augmentation below:

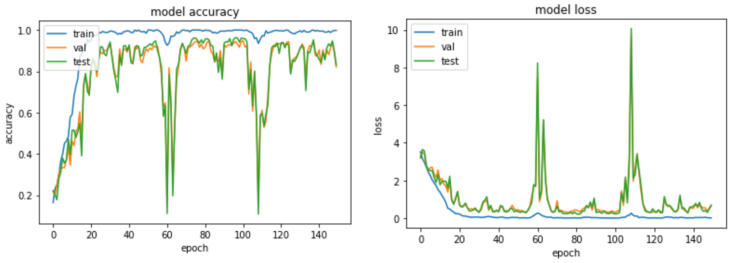
Finally, we also defined a Callback function in order to be able to evaluate testing data after each epoch (also in order to be able to plot it as the description stablished).

3.2 Results

The results we obtained in the end were:

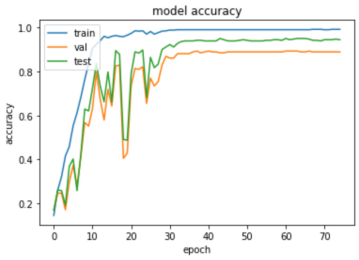
- Results with data augmentation: We obtained an accuracy of 0.95 and the evaluation time was 0.61. We noted that the accuracy had peaks of very high variance, we think this can be due to differences between the original images and the generated images and maybe also because in differences generated by changes in the learning rate.

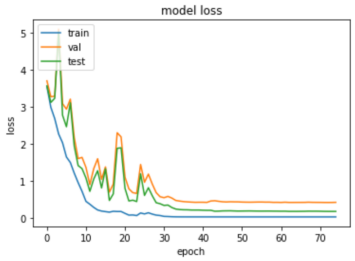
```
CNN took 0.6109945774078369 seconds
Test loss: 0.27436745561558723 - Accuracy: 0.9529085874227275
```



- Results without data augmentation: We obtained an accuracy of 0.94 with an evaluation of 0.38 seconds. We can see here that the variance present in the previous case is not shown here.

```
CNN took 0.387998104095459 seconds
Test loss: 0.1860291718084809 - Accuracy: 0.9445983382803581
```





After class we saw that the ones that had better results were those that had replicated one of the models we were going to use for transfer learning (but training it from scratch).

4 Transfer learning - Assignment 3

During this week we were requested to improve the results we obtained with our model trained from scratch by using a pretrained model provided in keras and adding some layers to it (to adequate it to our specific problem).

In this case the parameters we had to decide on were:

- Backbone architecture to use.
- Transfer learning approach: what layers to reuse, what layers to add, how to train the net,...
- Performance improvement (regularization, data augmentation, etc.).

After searching in keras documentation we saw that the models available for our use in the applications module were:

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
ResNeXt50	96 MB	0.777	0.938	25,097,128	-
ResNeXt101	170 MB	0.787	0.943	44,315,560	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

4.1 Process followed

During this stage we decided to give a try to google cloud and google colab, although after the configuration we decided to come back to local since the execution time was worse.

After studying the different models we decided to give it a try to Xception, since it was the one with better Top-5 accuracy among those models with less than 100MB. We had several problems in order to make it work in our local environment since we obtained several execution errors mainly related with lack of memory issues, finally we manage to run the model after:

- Reduce the signs size to 112x112
- Reduce the batch size to 6

Since we were having these problems we only included one dense layer and its softmax activation to the original model. We include the code example below:

```
from keras.applications.xception import Xception
from keras.layers import Dense, GlobalMaxPooling2D
from keras.models import Model
from keras.optimizers import Adam

baseline = Xception(include_top=False, weights='imagenet', input_shape=(SIGN_SIZE[0], SIGN_SIZE[1], 3))

x = GlobalMaxPooling2D()(baseline.layers[-2].output)
predictions = Dense(num_classes, activation='softmax')(x)

xception = Model(inputs=baseline.input, outputs=predictions)

opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

xception.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
xception.summary()
```

We used also the same data augmentation techniques we used in the previous assignment.

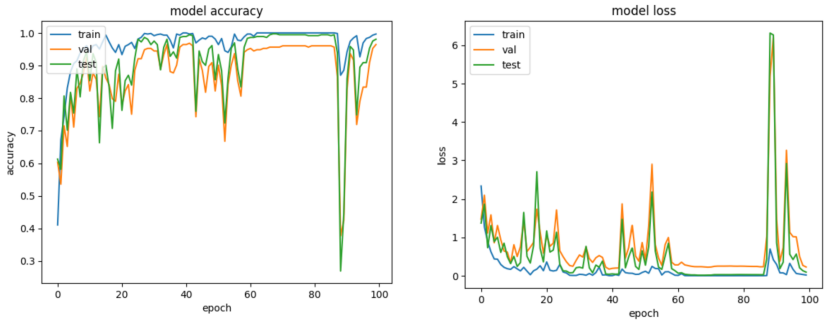
After being successful with Xception we tried with other models like ResNet152 or InceptionResNetV2, but we had memory errors while loading it, so we could not try the ones we choose. For some of them we obtained that they did not exist, we thought that this was because we were using an earlier version of keras (2.1.6) and maybe they were not included.

4.2 Results

So in the end, the results we obtained were:

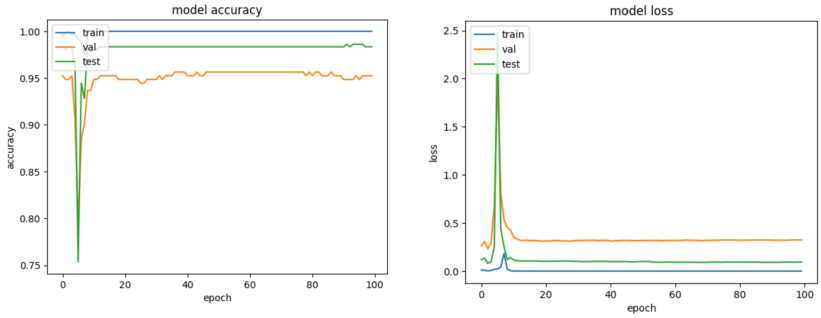
- Results with data augmentation: Accuracy of 0.9806 and execution time of 4.3220 seconds, we can note that the execution time is much bigger than our previous small model.

```
Xception (with Data Augmentation) took 4.322021484375 seconds
Test loss: 0.09281375004490386 - Accuracy: 0.9806094182825484
```



- Results without data augmentation: Accuracy of 0.9833 and execution time 4.3851, we can see that we obtained slightly better results without data augmentation.

```
Xception (without Data Augmentation) took 4.385195732116699 seconds
Test loss: 0.09433398998143537 - Accuracy: 0.9833795013850416
```



After the class we discovered that some of the models (like NASNetMobile) were smaller in order to be handled by smaller devices, we decided to give them a try in the next stages.

5 Object detection - Assignment 4

We had to built a traffic sign detection system using the RPN (Region proposal network) provided in class. We tried two different approaches:

- Build a unique model for classifying the regions obtained by the proposal
- Build, as the assignment suggested, a detector that differentiate between a sign and a not-sign and afterwards a model for set up the correct class.

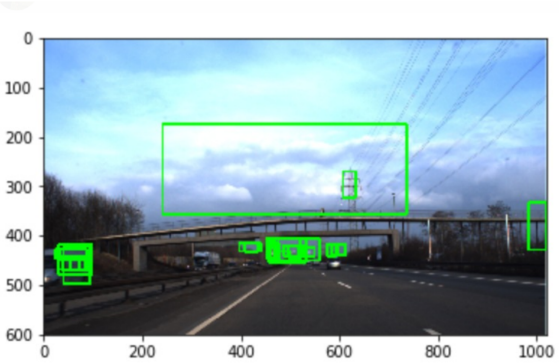
As the description of the assignments says we used mAP (mean average precision), we got the mAP calculator from the git repository provided (`https://github.com/Cartucho/mAP`). We had to modify the structure of the folders to make it work (add the detection-results folder inside the input folder).

5.1 Process followed

Here we will discuss the different approaches we had to the problem:

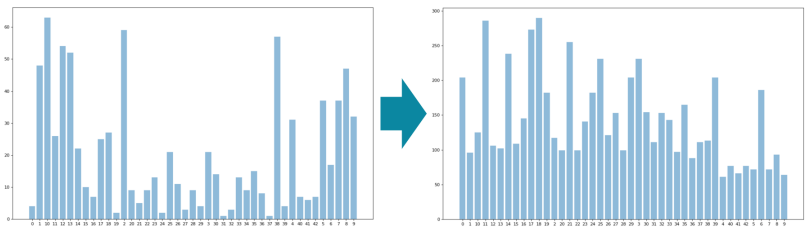
1. First of all, we applied our classifier trained in assignment 3 to the boundary boxes obtained in the RPN with default parameters. We filtered out those boxes with less than 0.9 confidence for the predicted class.

   It turned out that this was not a very good approach since we barely reached 10% mAP. Here we attach an example with 0.99 confidence filter:



2. After this first approach, we decided to go for the two classifiers method (one model for sign detection, another one for signal classifying).

   (a) We trained a new model for signal classifying, we used the same structure we used in Assignment 3 (Xception with one layer added). Additionally, we tried to leverage the labels by selecting random variations over the boundary boxes given (translations and zooming):



(b) After this we trained a Yes/No signal classifier. We generated a group of RPN boxes and we also calculated for the train set which boxes had a (normal) intersection with one of the ground truth boxes.

We picked those that intersect as 1 as those that did not intersect as 0. We thought that since the boxes will be generated by the RPN it would be better to have the training set coming from the RPN also instead that using the ground truth boxes as 1.

Then we selected a random sample for each class (around 2000 elements per class).

5.2 Results

We obtained a mAP of 20.63% using this approach:

```
3.23% = 1 AP
29.41% = 10 AP
8.33% = 11 AP
20.97% = 12 AP
16.43% = 13 AP
10.00% = 14 AP
77.14% = 15 AP
100.00% = 16 AP
37.50% = 17 AP
34.55% = 18 AP
9.09% = 2 AP
25.00% = 22 AP
14.29% = 23 AP
66.67% = 24 AP
5.00% = 25 AP
14.29% = 26 AP
1.67% = 28 AP
0.00% = 29 AP
22.22% = 3 AP
50.00% = 30 AP
0.00% = 31 AP
40.00% = 32 AP
33.33% = 33 AP
33.33% = 34 AP
40.00% = 35 AP
0.00% = 36 AP
0.00% = 37 AP
15.05% = 38 AP
0.00% = 39 AP
26.29% = 4 AP
0.00% = 40 AP
0.00% = 41 AP
0.00% = 42 AP
0.00% = 5 AP
0.00% = 6 AP
0.00% = 7 AP
16.67% = 8 AP
33.33% = 9 AP
mAP = 20.63%
```

After the class we discovered many things that maybe could improve our network, like:

- Filter the RPN boxes to obtain squares.
- Use the ground truth images as 1 label in the yes/no classifier (instead of RPN boxes that had intersection with the ground truth images).
- It would be interesting to retrain the yes/no network with previously wrong classified examples in the previous attempts.
- The network could be improved (adding more layers or trying other structures different than Xception for example).
- Tuning the parameters of the RPN could lead to better results.
- Use IOU instead of intersection for getting good images proposals for training.

- The use of not maximum suppression or intersection of classified images could also improve the results.

We tried to implement those ideas in the final process.

6 Final process

In this final revisit to the Assignment 4, we tried to implement all the improvements proposed during class in order to improve our results.

We have spent a lot of time in the beginning to develop the different scripts we used and we were having not quite good results, but in the end all the process lead us to a better performance.

In the first approach we used Xception from keras applications, we wanted to use other models to compare so we include also InceptionV3, we tried also other models but we have had problems with some of them.

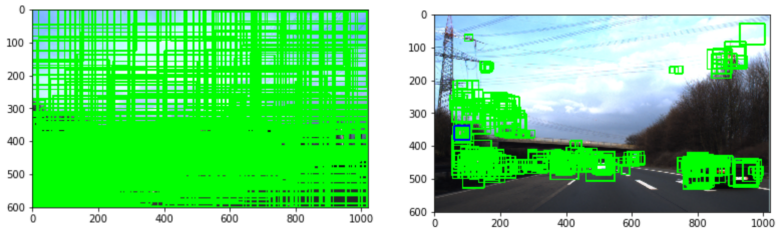
Again we had problems since we were not able to load mobileNet because of internal libraries problems, also we could not load the bigger models (like Resnet) because we had memory errors, other models were not available, as we said before, we think it was because we were using an older version of keras.

6.1 Process followed

Basically, in the end we followed the schema described below:

1. **Analysis of ground truth images** (00_Metric_extraction.ipnb), first of all we read all ground truth boxes and we extracted metrics like their biggest and smaller size, the biggest top, bottom, left and right borders and their biggest/smallest ratio (horizontal divided by vertical position for get squares).

   We filtered the boxes obtained by the RPN with these metrics. We include an example of RPN boxes before and after filter (in blue the ground truth):

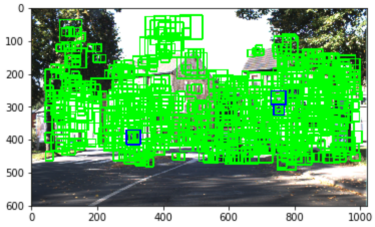


2. **Generation of region proposals** (01_Image_Generator.ipynb), we generated a bunch of boxes with the RPN, after that we applied the filters and labeled them, saving all into a folder.

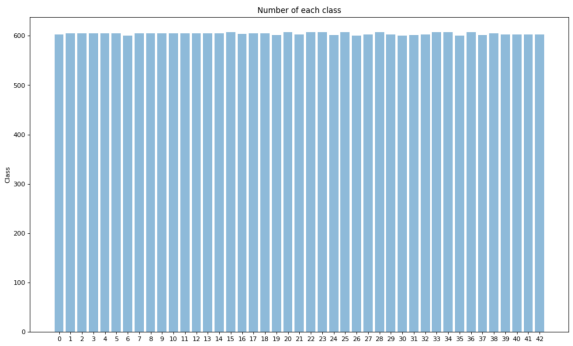
   We labeled the images with 1/0 and their sign class if the boxes had an IOU (not only intersection like before) bigger than 0.7. During the development we try different parameters for the RPN generation.

   Finally we used sign_sized equal to 224x224, img_height equal to 600, TEST_PRE_NMS_TOPK equal to 32000, TEST_POST_NMS_TOPK equal to 4000 and PROPOSAL_NMS_THRESH at 0.5.

   Here an example of the boxes saved:



3. **Training of classifier model** (02_Classification_Model_02_Xception.ipynb and 02_Classification_Model_03_InceptionV3.ipynb), afterwards we read the ground truth boxes, as we saw before the labels were not leveraged, so we increased the number of boxes and leveraged them by adding random rotations/zooming over the base ground truth boxes (we used 600 images per label for a total of around 25.800 images:



We used two different models. We tried InceptionV3 with the following configuration (the signs sized was 139x139 because of a requeriment of the library):

```
from keras.applications.inception_v3 import InceptionV3
from keras.layers import Dense, GlobalMaxPooling2D,GlobalAveragePooling2D, Flatten, Dropout
from keras.models import Model
from keras.optimizers import Adam


baseline = InceptionV3(include_top=False, weights='imagenet', input_shape=(SIGN_SIZE[0], SIGN_SIZE[1], 3))

for layer in baseline.layers:
    layer.trainable = True
# add your head on top
x = baseline.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=baseline.input, outputs=predictions)

opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

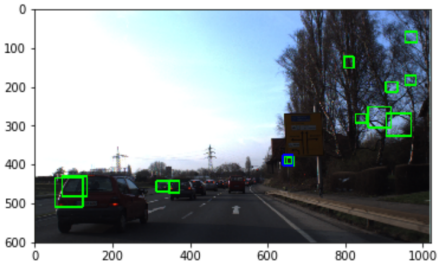
We obtained an accuracy of 0.9644 for the test set. The configuration we used for Xception (sign size of 112x112 because of memory problems) was:

```
from keras.applications.xception import Xception
from keras.layers import Dense, GlobalMaxPooling2D, Dropout
from keras.models import Model
from keras.optimizers import Adam

baseline = Xception(include_top=False, weights='imagenet', input_shape=(SIGN_SIZE[0], SIGN_SIZE[1], 3))

x = GlobalMaxPooling2D()(baseline.layers[-2].output)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=baseline.input, outputs=predictions)

opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

For an accuracy of 0.9514 for the test set. As we stated before we tried some other models from the application library of keras, but we had troubles making them work.

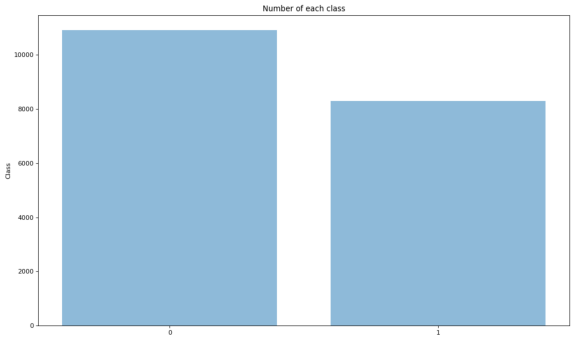
4. **Training of binary model** (Files 03_Yes_No_model.ipynb, and 03_Yes_No_model_Inception.ipynb), in order to train the binary model we used the images saved in the previous one as 1 (so having the labels leveraged) and those obtained in 2 as 0 (selecting randomly 25.800 of them). We run out of memory working with aroung 50.000 images so we used 8.000 for each class selected randomly (but ensuring we had images from each class in the case of the 1 label). We used also the same models of InceptionV3 and Xception, obtaining an around 0.95 for both of them. An example from each class:

5. **Test of binary model** (04_Final_Model_Test_YesNo.ipynb), once the models were trained, we tested first the binary model. We used the approach of saving those boxes of the RPN that were classified as a sign and they did not have an IOU bigger that 0.5 with any of the ground truth boxes in order to use them to retrain the network labeling them as 0. For example we saved the boxes that did not have an intersection with the blue one below (only for training set images, although this example is from test set):



6. **Retrain of binary model** (same as in 3.), after the testing we retrained the same models, including those boxes saved as 0 labels. We obtained an accuracy of 0.9935 for Xception and 0.9921 for InceptionV3.

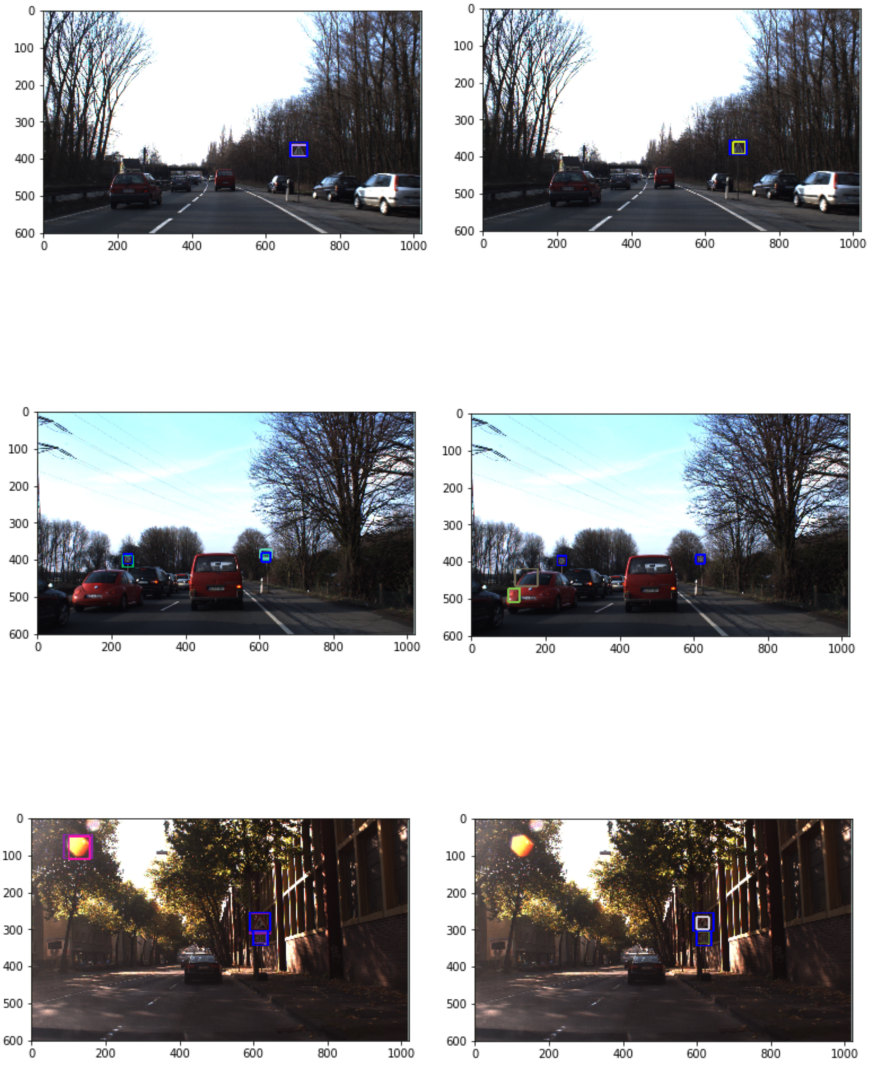
In the beginning we tried to retrain the models (load them and continue training with the new training set), but we were not able to retrain them because we obtained incoherent results, so we trained new models from scratch. We ended up with the following structure of data (we added the new boxes that is why we have more elements of class 0):



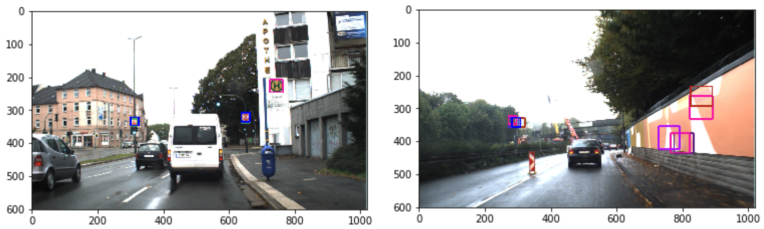
7. **Test of full model** (Files 05_Final_Model_03.ipynb and 05_Final_Model_03_Inception.ipynb), finally, we used both models to classify the RPN images obtained in the last assignment. We tried at first with the default parameters for the RPN, afterwards we tried to reduced the PROPOSAL_NMS_THRESH to 0.5 and increase it to 0.9, we obtained better results with 0.9. Then we tried to generate more samples during proposals and obtained even better results (TEST_PRE_NMS_TOPK equal to 50.000 and TEST_POST_NMS_TOPK equal to 5000).

   Regarding the configuration we filtered the boxes labeled as 1 by the first model to select those with confidence bigger than 0.7 and also they had to have the confidence of the 0 label less than 0.15. After all labels are calculated we use also not maximum suppression to prevent overlapping boxes with the same label.

We tried to compute the intersection of the boxes with the same labels that overlapped, but in the end we got better results with not maximum suppresion. Here we include some examples of images classified using Xception (left) and Inception (right):



As a remark we saw that sometimes the classifier detects signs that are not in the dataset like parking, crosswalk or hospital signs. We also had a bunch of images where the classifiers did not detect any signal (although it was) and others with wrong detected boxes like the following:



6.2 Results

Before reaching this final process we were having results around 20-30% mAP, but as we were adding features our results were improving.

We reached 54% before the retrain and 63% after changing the NMS.

Finally we got the final result after increasing the number of images and adding not maximum supression. In the end the best result we obtained was 76.02% mAP for InceptionV3:

```
93.55% = 1 AP
94.12% = 10 AP
100.00% = 11 AP
70.97% = 12 AP
90.32% = 13 AP
100.00% = 14 AP
100.00% = 15 AP
100.00% = 16 AP
75.00% = 17 AP
72.73% = 18 AP
90.91% = 2 AP
50.00% = 22 AP
100.00% = 23 AP
66.67% = 24 AP
70.29% = 25 AP
71.43% = 26 AP
80.00% = 28 AP
100.00% = 29 AP
77.78% = 3 AP
100.00% = 30 AP
0.00% = 31 AP
80.00% = 32 AP
100.00% = 33 AP
100.00% = 34 AP
100.00% = 35 AP
100.00% = 36 AP
100.00% = 37 AP
61.29% = 38 AP
0.00% = 39 AP
78.38% = 4 AP
0.00% = 40 AP
100.00% = 41 AP
0.00% = 42 AP
75.00% = 5 AP
50.00% = 6 AP
75.00% = 7 AP
95.87% = 8 AP
69.44% = 9 AP
mAP = 76.02%
```

And 82.96% for Xception:

```
85.96% = 1 AP
94.12% = 10 AP
100.00% = 11 AP
81.59% = 12 AP
71.76% = 13 AP
99.09% = 14 AP
86.67% = 15 AP
100.00% = 16 AP
50.00% = 17 AP
72.73% = 18 AP
90.69% = 2 AP
75.00% = 22 AP
87.50% = 23 AP
100.00% = 24 AP
90.00% = 25 AP
64.29% = 26 AP
100.00% = 28 AP
50.00% = 29 AP
66.67% = 3 AP
100.00% = 30 AP
100.00% = 31 AP
100.00% = 32 AP
100.00% = 33 AP
83.33% = 34 AP
96.67% = 35 AP
100.00% = 36 AP
100.00% = 37 AP
67.84% = 38 AP
36.11% = 39 AP
86.49% = 4 AP
46.67% = 40 AP
50.00% = 41 AP
72.92% = 42 AP
80.80% = 5 AP
100.00% = 6 AP
79.86% = 7 AP
96.69% = 8 AP
88.89% = 9 AP
mAP = 82.96%
```

Regarding execution time, Xception took 5511.63 seconds and Inception around 3042.39 seconds (we had a problem measuring it for Inception). The number of boxes that passed the filter could also affect these difference.

7 Personal comments and suggestions for next year

As we discussed in the last class, we think that this module of Deep Learning course is probably one of the best structured of the master, and it was possible to see that the teachers have devoted more time to the subject than the average across the different courses we have had.

Regarding specific comments about the module, we would like to have had at least a notion of how video/voice processing works (just an approach of voice to text or real time image classification for example).

Talking about the course structure, and like we said in class, we think the course can be reorganized in order to spend less weeks with the introductory materials (most of them already covered in subjects from the first semester) and expand this module and maybe the text recognition part (by including also a coding assignment).

We think that in general it could be highly recommended to have more coordination among the teachers of the master in order to deliver a program without overlapping subjects and of high quality, without revisiting subjects already covered in a basic computer science course and using technologies we will use in the future (unlike for example Weka).

Finally, we think Kaggle also offers services for private competitions (although we are not sure if it is for free) and could be an interesting point also to kept the accuracy obtained in the different assignments.

8 Conclusion

We have learnt a lot during the development of this set of assignments. In the first two we were able to solidify the knowledge we already had about feed forward and convolutional neural networks, as well as about keras.

During the third assignment we explored something we were not aware of, transfer learning, and during class we also visited those models and their drawbacks and advantages.

Finally during assignment 4 we faced a close approach to a real world problem. It was really challenging and we have devoted a lot of time, but it was worth because we have acquired the knowledge of how to approach these kind of problems.

Probably one of the worst things we faced during the development was the execution time, this has prevented us for trying more approaches. Maybe parallelization with different computers could help with this issue, but of course is not in our current capabilities.

We still see room of improvement of our solution, since we have checked some papers where they claim to achieve even 98% mAP. Maybe using not maximum suppression between labels and more specific architectures as well as more computing power to train on more images can lead to this results.

References

[1] INI Benchmark Website, http://benchmark.ini.rub.de/?section=home&subsection=news

[2] Sermanet Pierre, LeCunn Yann. Traffic Sign Recognition with Multi-Scale Convolutional Networks.

[3] Arcos-García, Alvaro et al. Evaluation of deep neural networks for traffic sign detection systems.

[4] Numpy array programming, https://realpython.com/numpy-array-programming/

[5] Image Augmentation for Deep Learning With Keras https://machinelearningmastery.com/image-augmentation-deep-learning-keras/

[6] Fully Convolutional Network (LB 0.193) https://www.kaggle.com/bluevalhalla/fully-convolutional-network-lb-0-193

[7] Object Detection Tutorial in TensorFlow: Real-Time Object Detection https://www.edureka.co/blog/tensorflow-object-detection-tutorial/

[8] A Beginner's Guide to Object Detection https://www.datacamp.com/community/tutorials/object-detection-guide

[9] Transfer Learning using Keras https://medium.com/@14prakash/transfer-learning-using-keras-d804b2e04ef8

[10] Traffic signs classification with a convolutional network https://github.com/navoshta/traffic-signs