DEEP NEURAL NETWORK BASED VENICE BOAT CLASSIFICATION ON THE ARGOS DATASET

Michela Ricciardi Celsi
1580884

Machine Learning

1. INTRODUCTION

This report describes a deep neural network based approach to the classification of boats travelling in the canals of Venice into 5 different classes. Transfer learning was adopted to reduce the training time for the classification model.

The dataset I used is available online on the website of the ARGOS (Automatic Remote Grand Canal Observation System [1], 2006-2007) project, which was aimed at developing a video surveillance system for boat traffic monitoring, measurement and management along the Gran Canal of Venice. The training set contains images from 24 different categories of boats navigating in the city of Venice.

The related images are to be grouped into the following 5 boat classification categories:

1. people transport;
2. general transport;
3. pleasure craft;
4. rowing transport,
5. public utility.

2. TRANSFER LEARNING WITH MOBILENET

At first, the deep neural network based approach followed is Google's MobileNet 1.0 224 [2]. Such a neural network was already trained on the ImageNet dataset. Its architecture is described in Table 1 below. Although the size of the images is 800×200 pixels, the corresponding TensorFlow pipeline implementing MobileNet compresses the images down to 224×224 pixels, as required by the neural network specifications.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

The project work presented in this report relies on TensorFlow, an open-source library for numerical computation in machine learning applications. I chose to adopt on transfer learning in order to reduce training time since the training procedure is carried out on a laptop computer.

In general, neural networks and therefore also convolutional neural network architectures exploit the fact that a model trained on a big dataset such as ImageNet captures general information about how an image is composed, e.g., the positions, structures, and the combinations of edges and shapes it contains. In this respect, lower convolutional layers capture low-level image features (i.e., edges) while higher convolutional layers capture more and more complex details, such as body parts, faces, cat whiskers, and other compositional features.

Based on this, the approach followed consists in resorting to a trained model with the aim of understanding a scene and extracting the most relevant features from the input images. Only the network parameters are tuned so as to ensure that the model do not unlearn the previously acquired knowledge. Eventually, the script trains the final layer that performs the classification onto a new dataset.

Therefore, the adoption of transfer learning implies that (i) I start from a model that has been trained on another problem before and then (ii) retrain it on a problem that is recognized to be quite similar to the previous one. In the considered case, this approach proves to be effective and less time-consuming than deep learning from scratch.

More in detail, the model I have used, i.e., Google's MobileNet 1.0 224, was trained on the ImageNet Large Visual Recognition Challenge dataset [3]. Such a model allows to distinguish among 1000 different classes, e.g., whether the current image represents a dalmatian dog or a dishwasher. Then, I retrained the same model to tell a small number of classes one from the other, with respect to the ARGOS dataset introduced above.

MobileNet was developed by Google for mobile and embedded computer vision applications, increasing efficiency with respect to size and speed, in comparison with other neural network based approaches to classification. In particular, in the MobileNet model, standard convolution filters are replaced by a first layer of depthwise convolution and then a second layer of pointwise convolution. The advantage of relying on such a structure consists in a sensible reduction of the training time, which proves useful especially if the training task is carried out on a laptop computer. MobileNet is a small efficient convolutional neural network and it is configurable according to two parameters:

1. Input image resolution: 128, 160, 192, or 224 pixels. Obviously, feeding in a higher-resolution image takes more processing time, but results in higher classification accuracy.
2. The relative size of the model as a fraction of the largest MobileNet: 1.0, 0.75, 0.50, or 0.25.

As anticipated, I used a MobileNet model with 224 pixels as input image resolution and 0.5 as relative size of the model. For further information in this respect, see [4].

For the purpose of visual analytics, I also used TensorBoard, in order to visualize the results of the implemented learning procedure. In this respect, TensorBoard can be opened by issuing the following command on the command prompt:

```
tensorboard --logdir TensorFlow/TrainingSummaries &
```

Then, the very retraining of the MobileNet onto the ARGOS dataset is launched. In other words, on top of the considered MobileNet model, that has already been trained on ImageNet images,

a new layer is generated, thus allowing to recognize other classes of images. Such a top layer receives as its input a 1001-dimensional vector for each image. On top of this representation, a softmax layer is then trained. Assuming that the softmax layer contains N labels, this corresponds to learning $N + 1991 \times N$ model parameters corresponding to the learned biases and weights.

The ImageNet based models, such as MobileNet, are made up of many layers stacked on top of each other. Such layers are pre-trained and are already very effective at finding and summarizing information that will help classify most images. In the retraining script attached to this report (`retraining.py`), I have trained only the last layer, while all the previous layers retain their already-trained state.

Here is an example, which assumes to have a folder containing class-named subfolders, each full of images for each label. The folder containing the ARGOS dataset is in fact arranged like this:

```
~/argos/dataset/train/Alilaguna/20130304_060950_07485.jpg
~/argos/dataset/train/Alilaguna/20130304_064009_12692.jpg
...
~/argos/dataset/train/Ambulanza/20130304_062055_08854.jpg
...
~/argos/dataset/train/Gondola/20130304_114650_08950.jpg
```

The subfolder names are important, since they define the ground truth annotation relative to each image. Instead, the filenames themselves do not matter. Once the images are prepared, the MobileNet retraining procedure can be run.
The label for each image is taken from the name of the subfolder it is in, within the `argos` folder.
This produces a new MobileNet model file that can be loaded and run by any TensorFlow program.

In particular, the first phase of this MobileNet retraining procedure analyzes all the images on disk and calculates the bottleneck values for each of them. A bottleneck is an informal term we often use for the layer just before the final output layer that actually does the classification. “Bottleneck” is not used to imply that the layer is slowing down the network. The term “bottleneck” is used because, near the output, the representation is much more compact than in the main body of the network.

Every image is reused multiple times during training. Calculating the layers behind the bottleneck for each image takes a significant amount of time. Since these lower layers of the network are not being

modified, their outputs can be cached and reused. Once the script finishes generating all the bottleneck files, the actual training of the final layer of the network begins.

In case some distortions occurs such as crops, scales or flips, the full model for every image must be computed again , which means that cached bottleneck values cannot be used. For this reason, random images for the requested category must be found and they are run through the distortion graph. Then, the full graph to get the bottleneck results for each.

The script attached to this report runs 2000 training steps. Each step chooses 10 images at random from the training set, finds their bottlenecks from the cache, and feeds them into the final layer to get predictions. Those predictions are then compared against the actual labels, and the results of this comparison is used to update the weights of the final layers through a backpropagation process.

As the MobileNet model retrains, the command prompt returns a series of step outputs, each one showing training accuracy, validation accuracy, and the cross entropy.

- The training accuracy shows the percentage of the images used in the current training batch that were labeled with the correct class.
- The validation accuracy is the precision (percentage of correctly-labelled images) on a randomly-selected group of images from a different set.
- Cross entropy is a loss function that gives a glimpse into how well the learning process is progressing – lower numbers are better.

3. PERFORMANCE RESULTS OF MOBILENET

The ARGOS dataset was first shuffled and then split into a training set amounting to 80% of the original dataset, a validation set amounting to 10% of the original dataset, and a test set amounting to 10% of the original dataset. The learning rate, that is, the parameter controlling the magnitude of the updates to the final layer during training[1], was set to 0.01. Random cropping, scaling, brightening and also flipping, were applied on 10% of the dataset. The entire validation set was used to test the performance of the network during its training (620 samples). Finally, the retraining procedure took 2000 epochs before reaching the end of the process and test the accuracy on the test set. The entire process required approximately 8 hours on my laptop with 2,5 GHz Intel Core i5, performing the code offline.

Fig. 1 shows accuracy (y-axis) as a function of the training progress (x-axis).

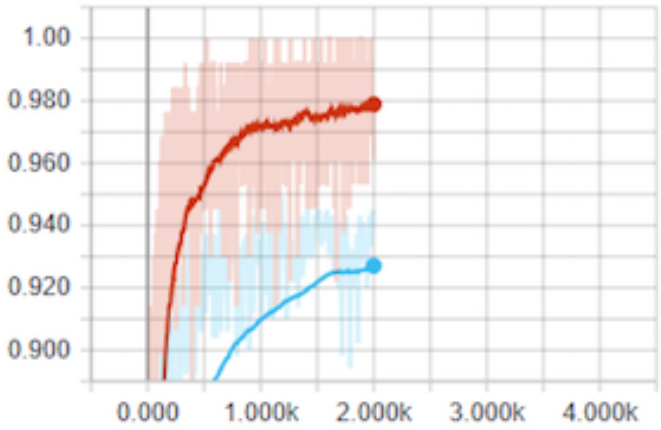

Figure 1 Accuracy obtained with 2000 epochs.

Two lines are shown. The red line shows the accuracy of the model on the training data, while the light blue line shows the accuracy on the test set (which was not used for training). This is a much better measure of the true performance of the network. If the training accuracy continues rising while the validation accuracy decreases then the model is said to be "overfitting." Overfitting is when the model begins to memorize the training set instead of understanding general patterns in the data.

As the process continues, the reported accuracy improves. After all the training steps are complete, the script runs a final test accuracy evaluation on a set of images that are kept separate from

[1] If you specify a small learning rate, like 0.005, the training will take longer, but the overall precision might increase. Higher values of learning rate, like 1.0, could train faster, but typically reduces precision, or even makes training unstable.

the training and validation pictures. This test evaluation provides the best estimate of how the trained model will perform on the classification task.

The reported accuracy value falls in the range between 85% and 99%, even though the exact value varies from run to run since there is randomness in the training process. This number value indicates the percentage of the images in the test set that are given the correct label after the model is fully trained. More in detail, by using TensorBoard as a visual analytics tool to monitor the learning performance, the MobileNet retrained pipeline was able to achieve an accuracy of 95.4% on a test set composed of 589 elements.

Fig. 2 shows cross-entropy (y-axis) as a function of the training progress (x-axis).

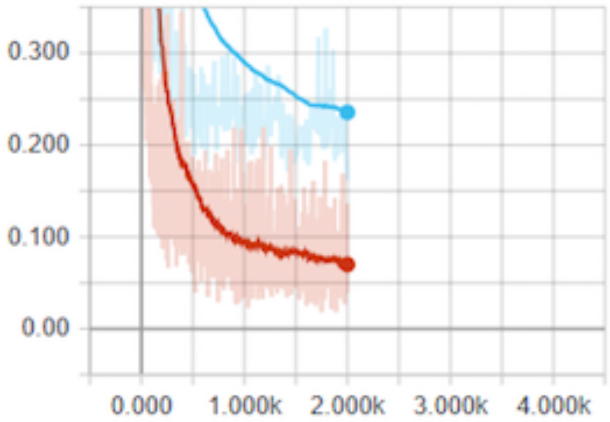

Figure 2. Cross-entropy obtained with 2000 epochs.

The objective is to minimize the cross-entropy as much as possible in order to obtain better performances. Moreover, the confusion matrix is shown below.

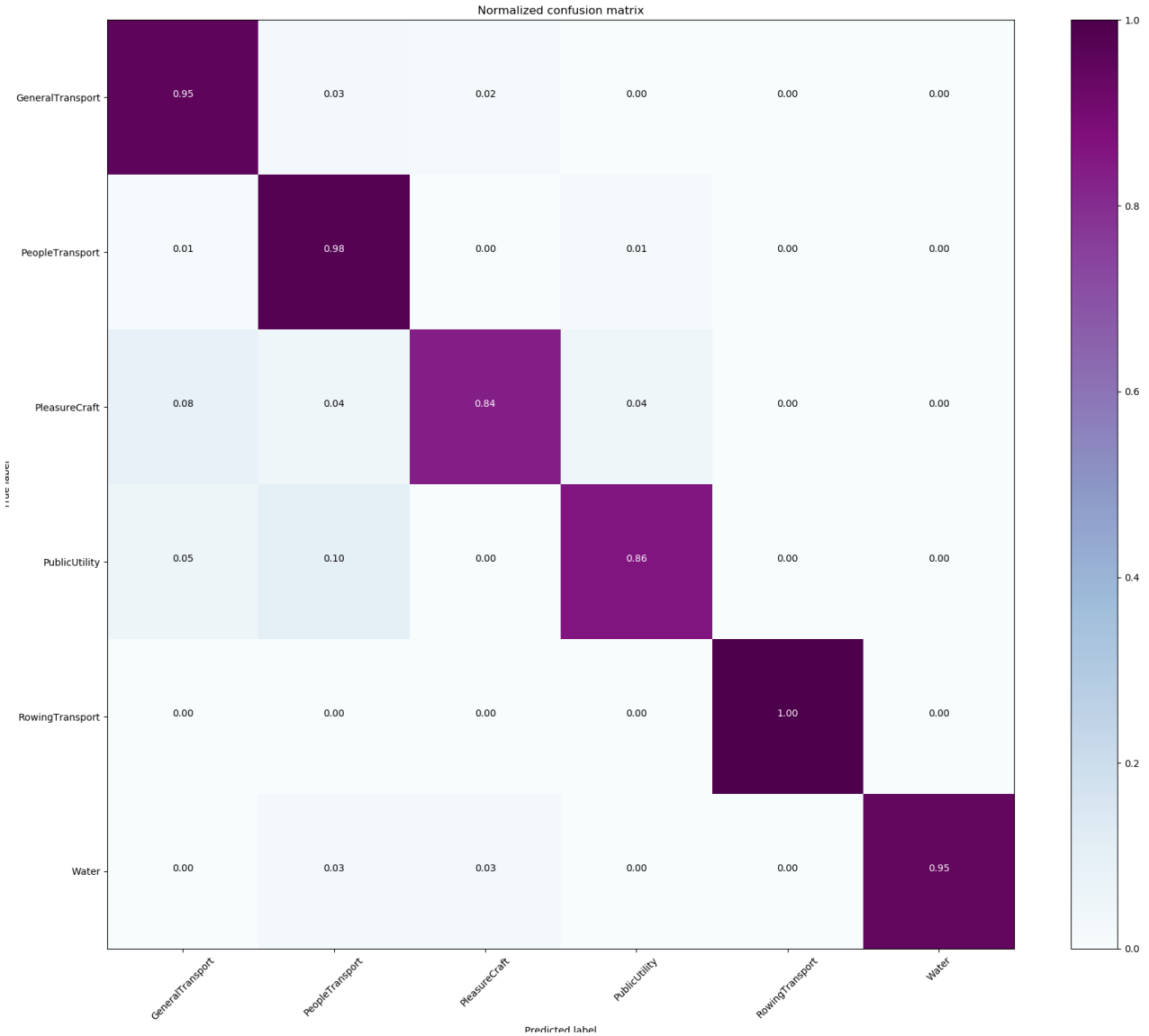

Figure 3 Confusion matrix obtained with 2000 epochs.

4. TRANSFER LEARNING WITH INCEPTION

After performing the transfer learning with a MobileNet, I also used Inception-v3 architecture in order to make a valuable performance comparison. So, this time the script uses an image feature extraction module with a pretrained instance of the Inception-v3 architecture. It consists of two parts:

- feature extraction part with a convolutional neural network;
- classification part with fully-connected and softmax layers.

It is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts and fully connected layers. It is composed of a basic unit considered as an "Inception cell" in which a series of convolutions at different scales is performed and then the results are aggregated.

In order to save computation, 1x1 convolutions are used to reduce the input channel depth. For each cell, a set of 1×1, 3×3, 5×5 filters can learn to extract features at different scales from the input. Max pooling is also used, even though with the same padding in order to preserve the dimensions and the output can be properly concatened.

In particular, the Inception-v3 architecture is represented by Fig. 4.

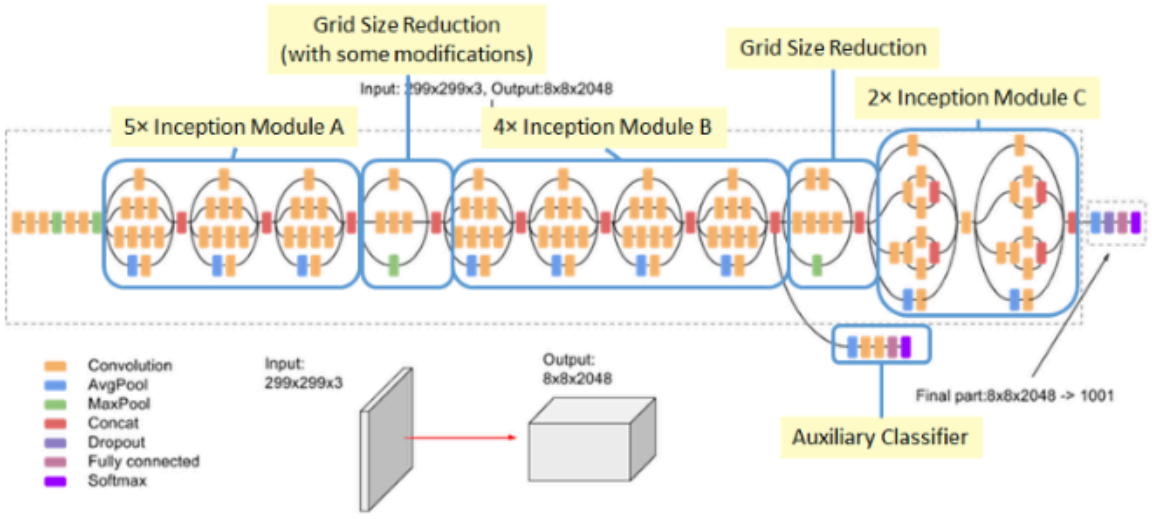

Figura 4 Inception-v3 architecture.

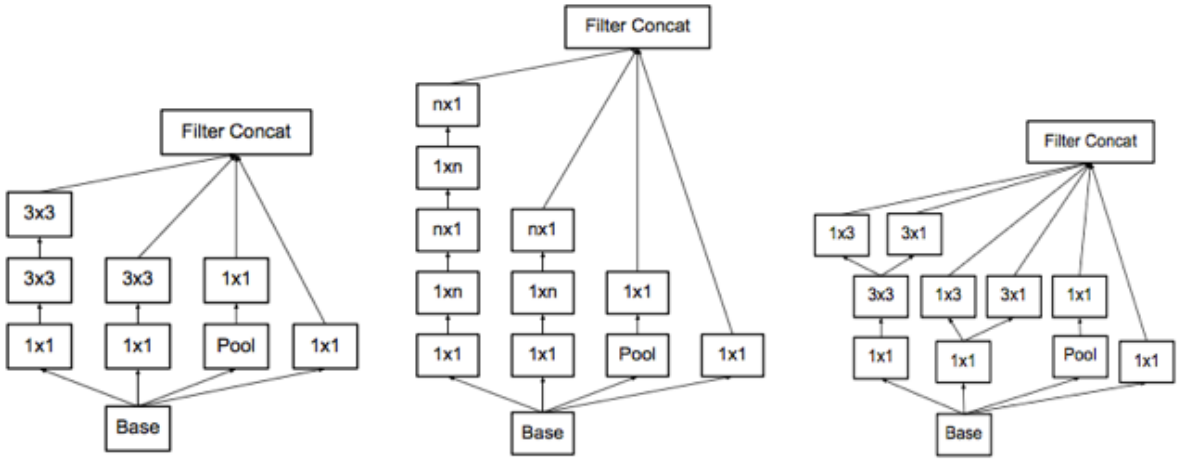

Figure 5 *a)* *b)* *c)*

type	**patch size/stride or remarks**	**input size**
conv	3x3/2	299x299x3
conv	3x3/1	149x149x32
conv padded	3x3/1	147x147x32
pool	3x3/2	147x147x64
conv	3x3/1	73x73x64
conv	3x3/2	71x71x80
conv	3x3/1	35x35x192
3xInception	As in 5-a)	35x35x288
5xInception	As in 5-b)	17x17x768
2xInception	As in 5-c)	8x8x1280
pool	8x8	8x8x2048
linear	logits	1x1x2048
softmax	classifier	1x1x1000

1. PERFORMANCE RESULTS OF INCEPTION

I performed the simulations with Inception setting 900 epochs and it took more than 8 hours to complete them on my laptop. The other parameters used are the same as the MobileNet. The original dataset is divided as before: 80% is dedicated to the training set, 10% is dedicated to the test set and the last 10% is dedicated to the validation set. The accuracy and the cross-entropy obatined are shown below.

In order to compare the performances with those obtained using the MobileNet, I can underline that after setting the number of epochs as 900 to be completed in 8 hours , only 730 of them were reached using the Inception v3, while the target number of epochs set for the MobileNet was reached in 8 hours (i.e. 2000 epochs).

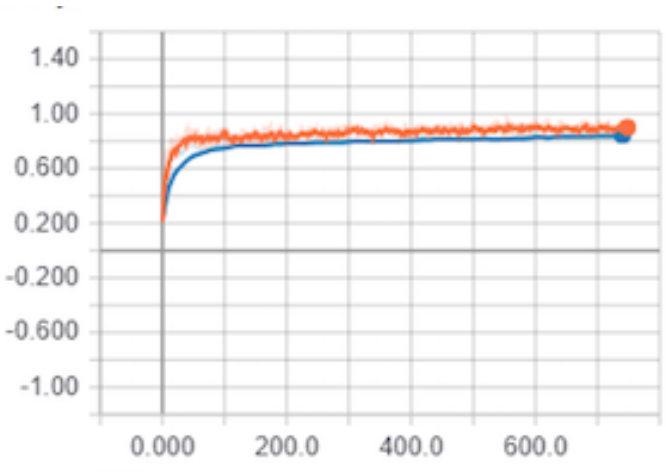

Figure 6 Accuracy obtained with 730 epochs

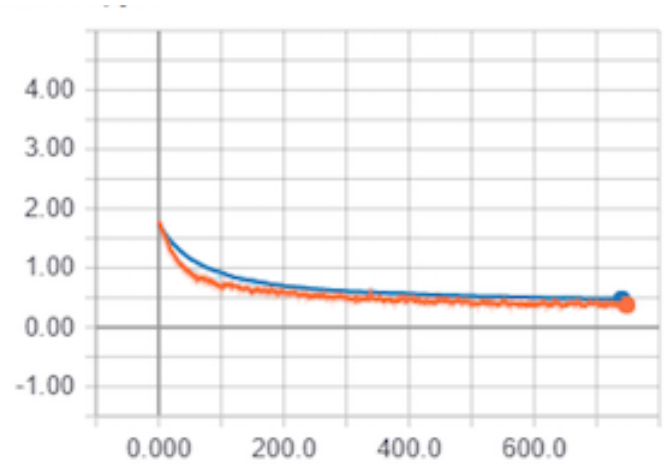

Figure 7 Cross-entropy obtained with 730 epochs.

The red line shows again the accuracy and the cross-entropy of the model on the training data, respectively. While the light blue line shows the accuracy and the cross-entropy on the test set, respectively. As it can be seen from the results obtained, this time the Inception v3 retrained pipeline was able to achieve an accuracy of 89.7% on a test set composed of 589 elements. Therefore, the accuracy level is lower with respect to the one reached with the MobileNet. Regarding cross-entropy, it seems the results are pretty good.

The confusion matrix is the one shown below:

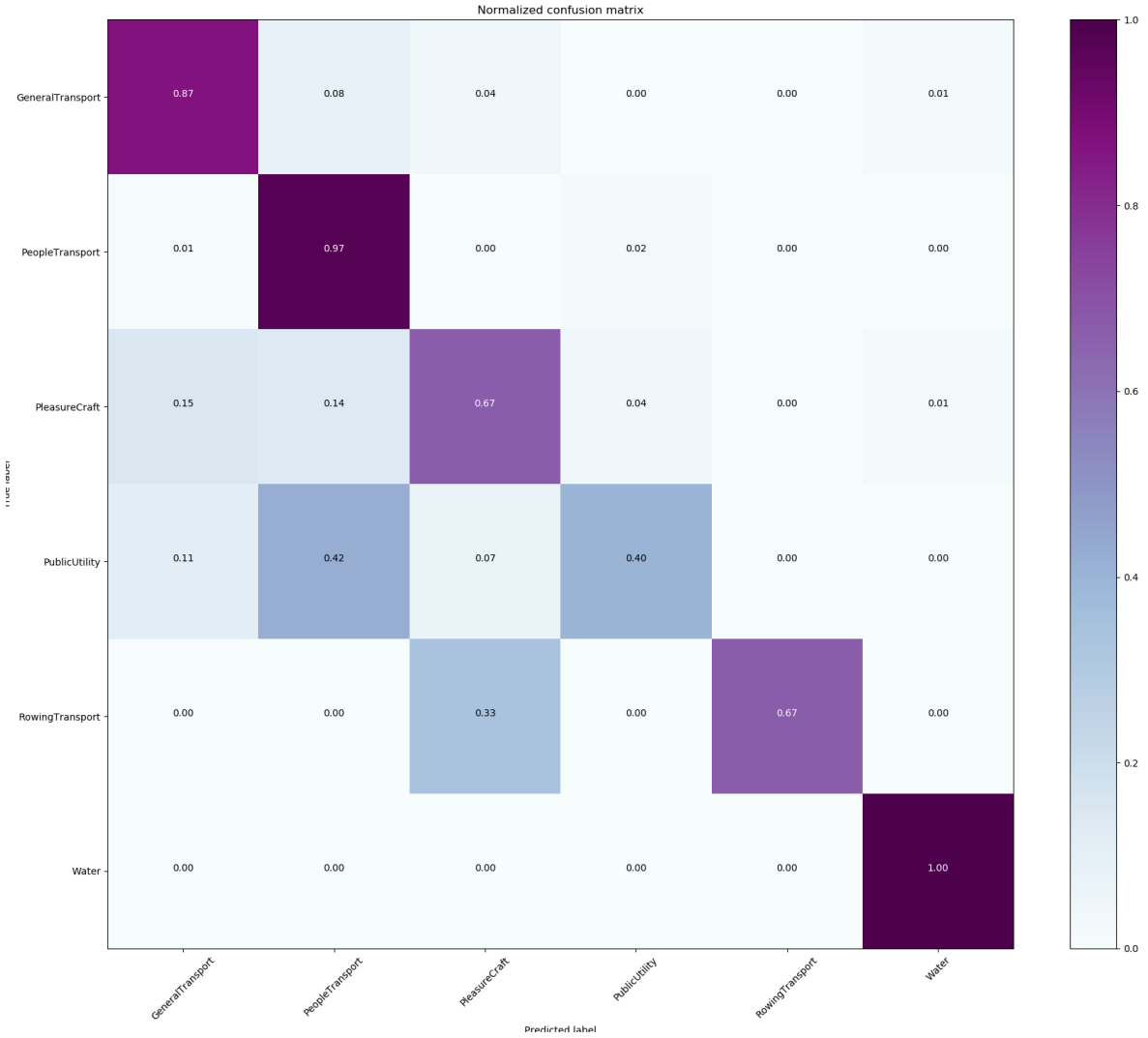

Figure 8 Confusion matrix obtained with 730 epochs.

In approximately 10 hours, the 900 epochs were completed using Inception v3 and the results of the simulations are shown below:

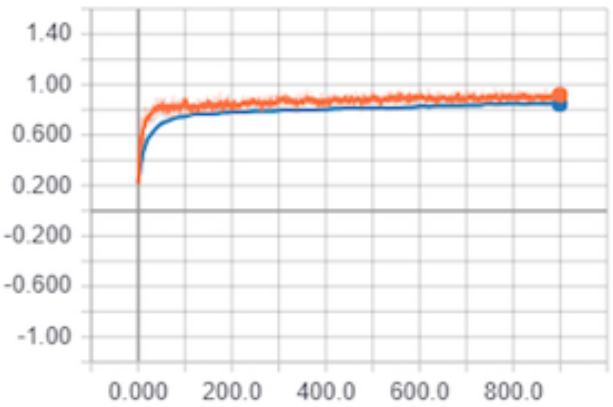

Figura 9 Accuracy with 900 epochs.

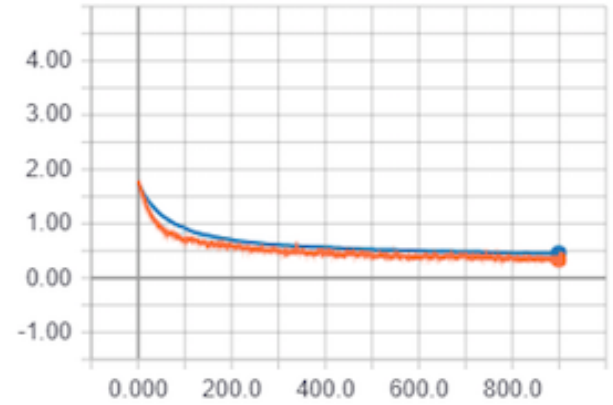

Figura 10 Cross-entropy with 900 epochs.

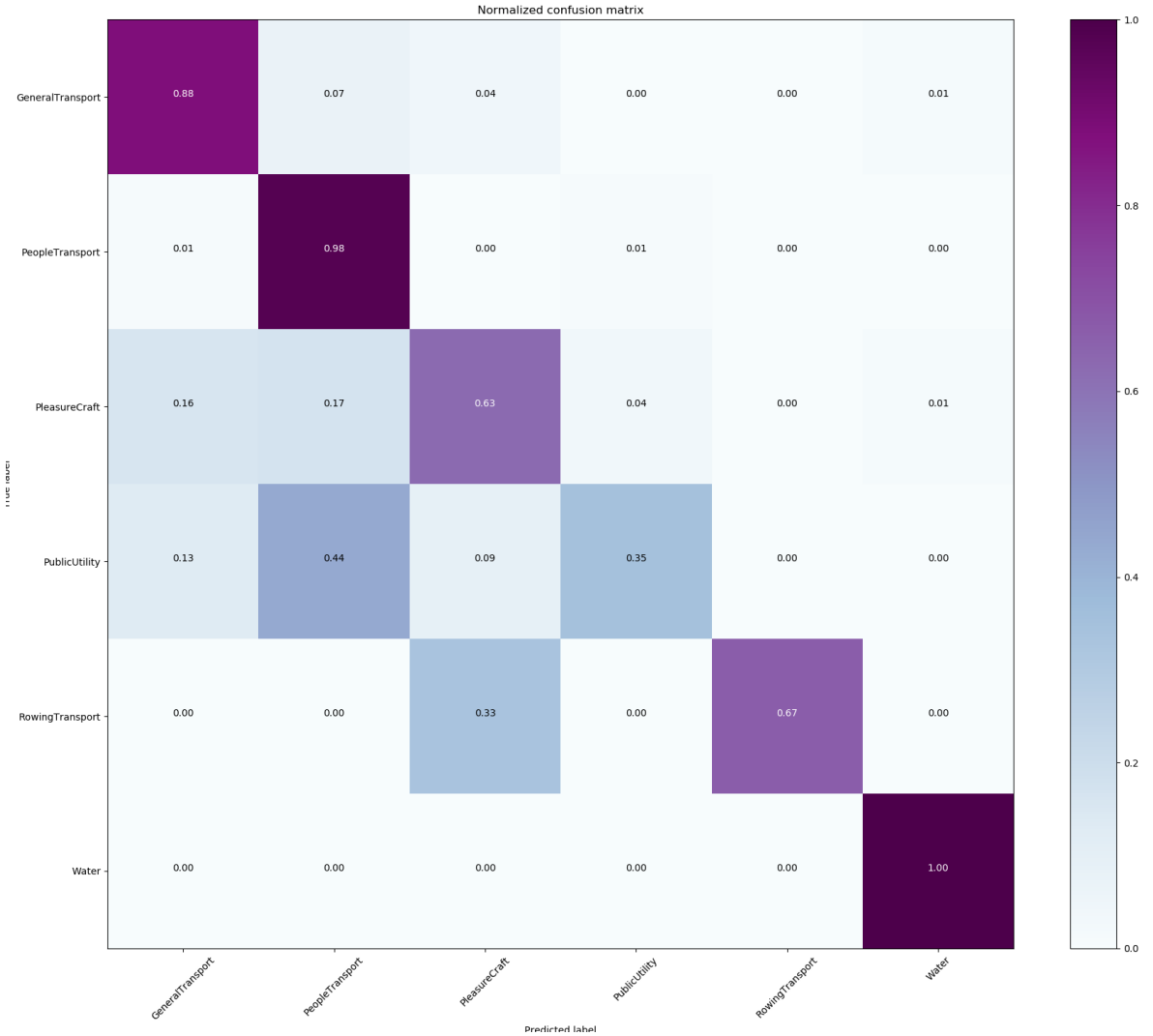

Figura 11 Confusion matrix with 900 epochs.

Also considering 900 epochs, the values of the accuracy and cross-entropy is not varying from the case in which I considered just 730 epochs. So, I can confirm that the MobileNet is performing better than Inception v3.

6. CONCLUSIONS

MobileNets are small, low-latency and low-power models and they can also be run efficiently on mobile devices with TensorFlow Mobile. They trade off between latency, size and accuracy while comparing with other popular models. However, sometimes the network misclassifies a few images, namely those belonging to the class of Public Utility boats, since they are very similar to the People Transport but are used by officers instead of civilians.

Moreover, Inception-v3 architecture has turned out to be slower than the MobileNet. Then, I can conclude that the MobileNet is a good choice because it provides high accuracy results with moderate running time for the retraining script.

Although I obtained always good accuracy and cross-entropy, for faster or smaller models a MobileNet such as 'mobilenet_1.0_224' is preferred: it picks a model that is 17MB in size and takes 224 pixel input images (as explained in the introduction). In order to make it a little faster, I could reduce the size of input images from 224 down to 192, 160 or 128 pixels squared. Another improvement could be to choose percentages 100, 075, 050 or 035 instead of 025 in order to control "feature depths" or the number of neurons per position. Moreover, the number of weights (e.g. the file size and speed) shrinks with the square of that fraction.

7. REFERENCES

[1] D. Bloisi, L. Iocchi, A. Pennisi, and L. Tombolini, "Argos-venice boat classification," *2015 12th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1-6, 08 2015, http://www.dis.uniroma1.it/~iocchi/ARGOS/index.html

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[3] http://image-net.org/

[4] https://research.googleblog.com/2017/06/mobilenets-open-source-models-for.html

[5] C.Szegedy, V.Vanhoucke, S.Ioffe, J.Shlens and Z.Wojna, "Rethinking the Inception Architecture for Computer Vision", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.