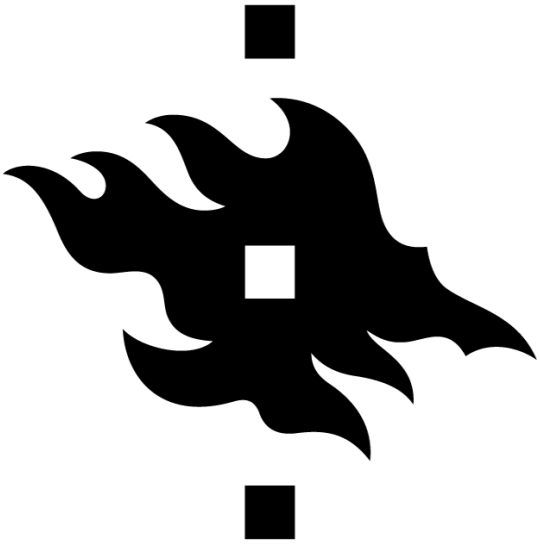
University of Helsinki



Report of the project of Deep Learning

Image project

Students:
Riccardo Menoli
Stefano Lia
Tafseer Ahmed (014973194)

2018/2019

Contents

1 About the project

To do the project we used **Google colab**, the Google's free cloud service for AI developers. The reasons that convince us to use this tool are the possibility to deploy both python and jupyter notebook file and the availability of the GPU. The whole project is conained in the file `image_project.ipynb`. To see the fully results you need only to run the file.

1.1 Precision vs Recall

We need to decide to emphasize the precision or the recall. In the project we are dealing with multi-label predictions, so we need to predict one or more classes for each sample. We think that **precision** is more important in this case. In fact, having a good recall means that the prediction could be correct for one class, but also other classes can be considered as belonging for one sample. This is not interesting. What we want is to be sure that one class appears in that image. To achieve this goal we first calculated the threshold for each class (as written in section 2.2). Then for each model we changed a bit the threshold to make the predictions more accurate (augmenting the precision).

2 Data Management

To start the project the first thing to do was finding the proper way to handle the data. In this section we explain all the problems we faced and the corresponding found solution during this process.

2.1 Creation of the Dataset

The first problem was the **creation of the Dataset**. As the guide of PyTorch and the exercise suggested we created a custom Dataset following this guide. Based on this we organized the pandas Dataframe in the following way:

- one column containing the path of the image;
- one column for each class. If the sample (the row) belongs to one on more class, it has 1 in the corresponding column which represents the belonging class;
- one column containing the representing image in the PIL format;
- one column containg one list whose values are the classes which the example belongs to.

2.2 Grayscale problem

We noticed that some images in the dataset are in grayscale. This is considered a problem because the `in_channels` parameter of the convolutional layer requires a fixed number of channels to work properly. To solve this problem we decided to **convert** the greyscale images from one channel to three channels. This can be easily done using the pytorch function `convert("RGB")`, after opening the image. This operation is done when an image is taken from the dataset (in `__getitem__` function).

2.3 Missing values problem

Some images in the dataset miss at least one label or are completely without them. If we plot these images we can easily notice that some of them can be considered uncorrectly unlabeled. In fact, there are some images, in which can be reasonable to have one (or more) of the considered classes, with some missing labels (i.e. im3 represents a river but it has no labels or im19 which represents clouds).
Our first idea was to remove these images. From our point of view they could affect the results adding undesiderable noise to the data. The algorithm could have difficulties to understand the right pattern if similar images have different labels. The problem was the precision. In fact, some images does not contain any class and have an empty prediction is

the maximum we can achieve. So, if we eliminate all the images without label the model will learn also to make always a prediction (which makes the model less precise). Since labeling all the images which suppose to have a class was to expensive, we decided to train the models on the whole training set.
The division between training and test set is respectively 90% of the data for the training and 10% for the test.

2.4 Imbalanced data

The dataset is strongly imbalanced. What we mean here is the fact that we have just few examples for some classes. In fact, the distribution of the training set looks like this:

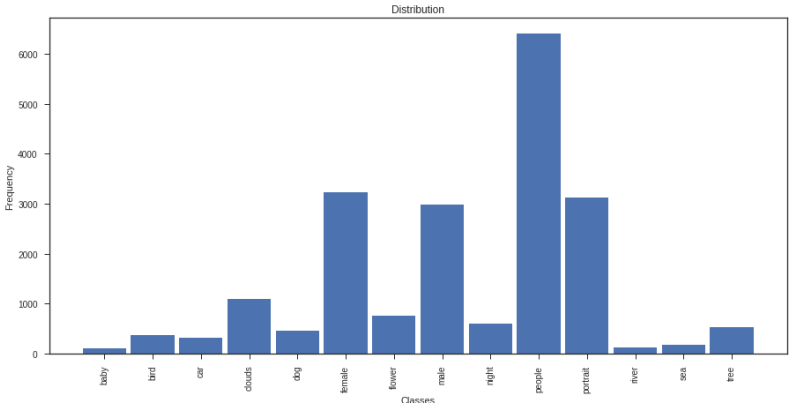


Figure 1: Distribution of the classes

The main consequence of this problem is that the network could have difficulties to predict correctly the classes with few examples in the training set. Our **solution** was to find a good threshold for the predictions. Each model that we tried has as final output a fully connected layer with 14 neurons (because we have 14 different classes). The activation function used in this layer is the sigmoid. So, finding the right threshold helps the model out to make good predictions. The idea was to request a minor activation for neurons that represent classes with few examples. Vice versa for the classes with many examples.
We started considering as a good threshold the proportion of the data in the training set. For example, if the class 'bird' is the label of the 30% of the dataset the started threshold for that class is 0.3 (so to predict 1 for this class the output of the neuron should be higher respect to this value). Notice that we have adjusted a bit this threshold in order to have better accuracy than recall. Our idea was to make the predictions when there is an high activation of the corresponding neuron.

2.5 Batching and scaling

The chosen batch size is 64 samples. Since we trained on the 90% of the dataset (180000 images), 64 samples seem reasonable because it allows both to have some updates and good training speed for all the models. Since we are using the **Adam optimizer** a good learning rate is always found. So, the model requires less optimization step.
To improve the results we applied some transformations to the images (i.e. RandomHorizontalFlip and RandomRotation).

3 Models

We tried the following different models:

1. Custom Inception Net: a custom version of the Inception_v1 network;
2. Custom Resnet: a custom version of the Resnet 101;

3. Resnet50 (pretrained).

Each model can be found in the directory `models`. For each model we will show the comparison between validation and training loss. The final models (which are saved in the directory) are chosen when the validation error is lowest in order to avoid overfitting and get the most general solution.

3.1 Custom Inception Net

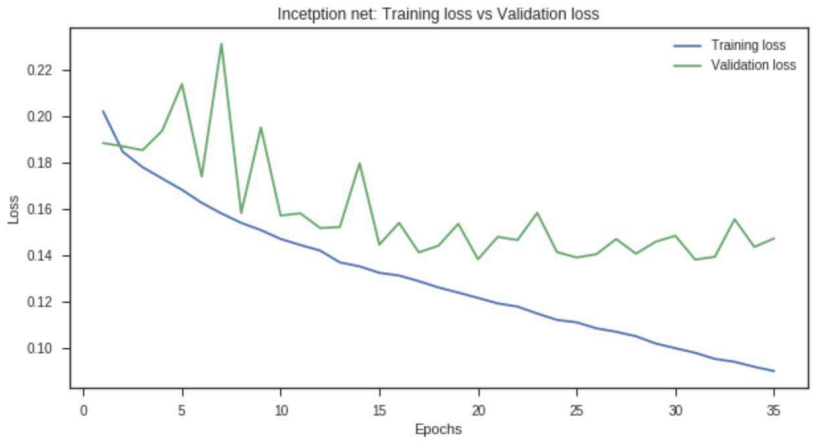


Figure 2: Custom Inception Net loss

4 Conclusion