Deep Learning Project - Report

Michael Hartung, Hanna Holtdirk, Niels Thiele

May 2020

1 Introduction

This document is the report for the project of the "Deep Learning" course at the University of Helsinki.
We chose the image project with the task of (building and) training a neural network to be able to assign labels to images. The different labels are: baby, bird, car, clouds, dog, female, flower, male, night, people, portrait, river, sea, tree. We have 20 000 images with annotations, i.e. labels, available for training and images can have multiple and no labels. Therefore, we are dealing with a **multi-label classification problem**.

2 Approaches and Methods

2.1 (Pre-)processing of data

To extend the basis of images we work with, we decided to augment the existing data. Afterwards, we have five versions of each image available with each one being altered in some way, e.g. being partly erased. This also allows us to deal with the data imbalance before we start training.

The given data is hugely imbalanced, meaning the labels are not distributed "equally", as can be seen in Figure 1 and 2. Especially the label "people" is really overrepresented and about half the images have no label at all.

Figure 1: Label occurrences

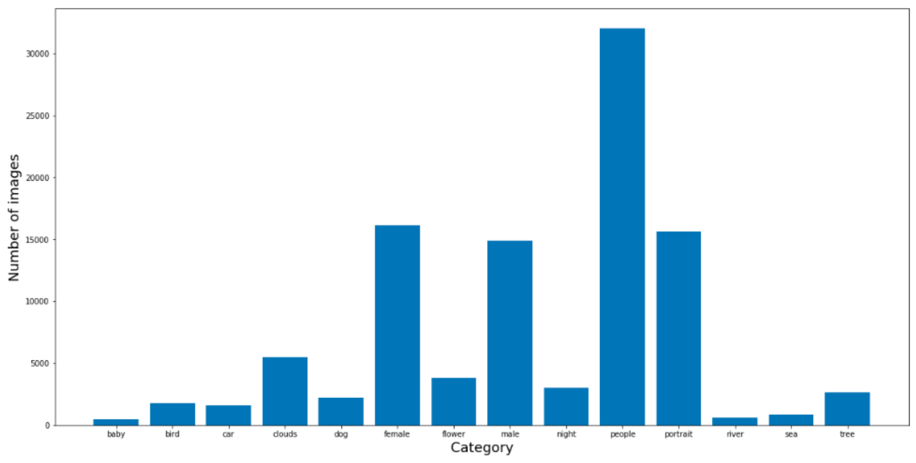
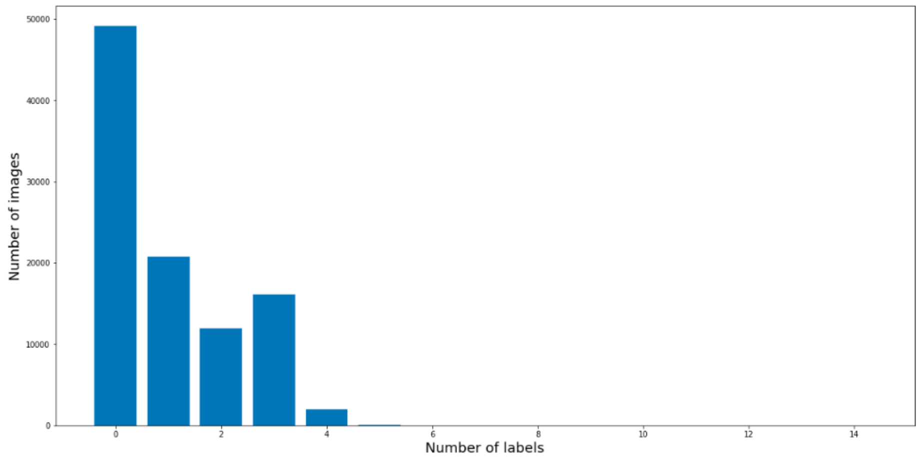


Figure 2: Distribution of labels



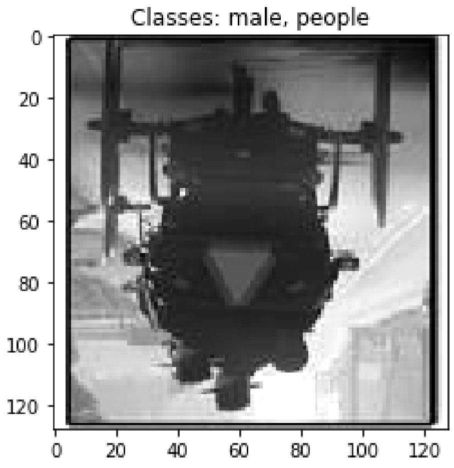
Our main technique to balance this out is to use weights in the optimization during training (see 2.3). But our program also has the possibility to sample images in the preprocessing stage in order to reduce the imbalance. We drop images (with a certain probability) if they have no category and if they have the label people (and no underrepresented label). At first, we did this without farther restrictions, but this led to a loss of information, because for some images no version of it was left. That's why we now ensure that at least one version of each image is left in the data set. Therefore, even though we undersample on the whole data set, we actually apply oversampling because of the previous data augmentation. A good side effect is that the training set is smaller and thus faster to propagate.

After this, we build the training and validation data sets and their loaders. Originally, we only had a training set and used it for training and evaluation (see 3). But because this doesn't give us much information regarding overfitting and generalization, we now split the data into a training and validation set. At first, this happened fully randomly without further restrictions, but now, we only use unaugmented and unique images in the validation set in order to simulate a "real" application environment.
To further prevent overfitting, the training data can be randomly transformed in different ways including a change in brightness and contrast.

The labels are represented as a binary list, where a one at the ith element means that the corresponding image is appointed the ith category (categories are ordered alphabetically). To begin with, we represented the case that an image belongs to none of the given categories as an extra class. However, this made the interpretation of results and evaluations through metrics more difficult or misleading, because a correct 1 for the extra class counted as a true positive for example and an image could be predicted to have the extra class, i.e. no label, and another class simultaneously. Now, the label vector for an image with no annotations is the zero vector.

2.1.1 Examples

Here are some example images that are used for training:



2.2 Model

We chose to build a new model and not use a pre-trained one to be able to have full control of the training process and to experiment with the structure of the model as well as the training parameters.
For the layers, we roughly followed the common architecture for CNNs as explained in the lecture. There are four layers of convolution with a ReLU and (max) pooling.
Then, there is a fully connected layer with ReLU activation followed by the output layer that has 14 neurons, one for each class. Because we are dealing with multi-label classification, the output activation is Sigmoid instead of Softmax, which is the usual for ”normal” multi-class classification. Therefore, the output at one neuron correlates with the probability that the input has the corresponding label. The threshold for a positive prediction is 0.5 in each class because we use weighted optimization (see 2.3).

2.3 Training Algorithm

During the course of the project, we tried different approaches and parameters for training our model.
As the output activation is Sigmoid, the loss function should be binary cross entropy. There are two versions of this available in pytorch, BCE and BCEWithLogitsLoss. Because the second one already includes the Sigmoid and is more numerically stable, we ended up using that one. However, in training we discovered that the model tended to predict zero for every category for every image, so we added weights to the criterion that correlates with the ratio of negative to positive examples in every class. The weights also allow us to trade of precision against recall to get a good balance (see 3)

For the choice of optimizer, we tried SGD (with momentum) and Adam with different learning rates. For SGD, we found that the learning rate that works best is 0.01, because 0.1 didn’t give us good results and 0.001 was very slow. For Adam a good learning rate turned out to be 0.001 and in general in this approach we get good results faster and the training progress seems to be more stable.

3 Learning Progress

We use different metrics to evaluate our learning progress:

1. Loss: average loss
2. Set Accuracy: fraction of correct predictions, where a prediction is only counted as correct if all labels have been correctly predicted

3. ”Hamming” Accuracy: fraction of correctly predicted labels

4. Precision: measures ability to not assign a class when image doesn’t have it as a label (tp / (tp + fp))

5. Recall: measures ability to find the positive labels (tp / (tp + fn))

As mentioned before, we at first used the training set to evaluate our progress. Although this shows us that the model is, in fact, learning, it doesn’t give us information about how good the model is at generalizing. Extreme overfitting leads to really good measuring results, which is of no use when the model should be applied on another data set.
Because we now validate after every training epoch on a set that is distinct (even considering augmentations of the same original image) from the training set, our evaluation measures give a pretty good picture about the quality of our progress and model.

3.1 Example

In the figures 3, 4, and 5, you can see the learning progress when applying augmentation and sampling, transforming the training data, choosing the Adam optimizer with learning rate 0.001 and a batch size of 64 (see hyperparameters in code).

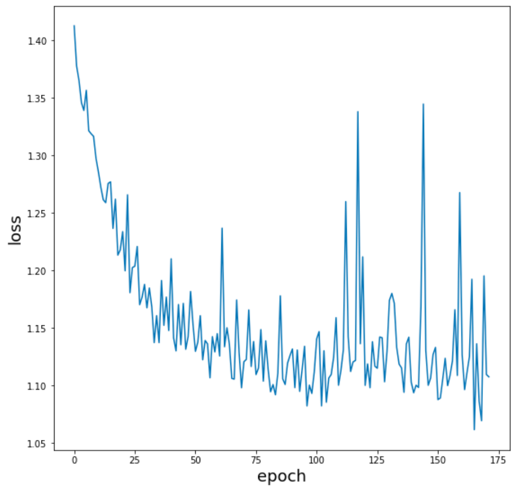


Figure 3: Loss

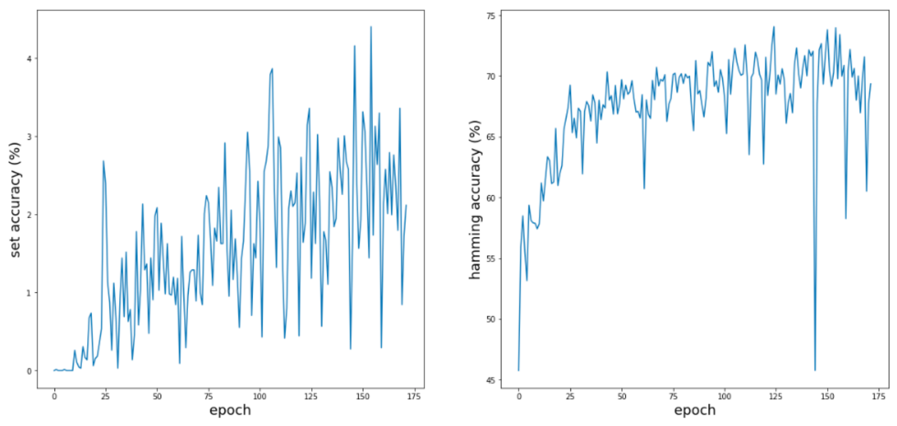


Figure 4: Accuracy

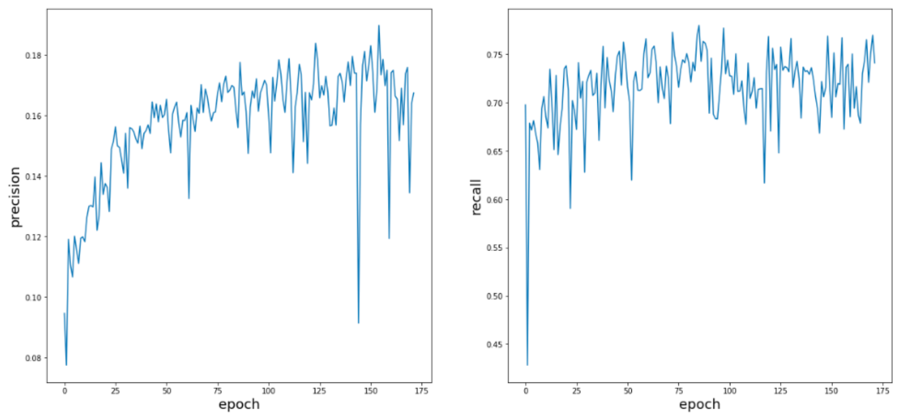


Figure 5: Precision and Recall

The development of the loss is the best way to see the overall progress of the network. You can see that it falls pretty rapidly in the beginning which is to be expected because we used a new model that wasn't trained before. Then, it evens out and from about the 100th epoch it doesn't seem to make much more progress. The hamming accuracy at that point is around 70% and the set accuracy is only at about 3%. We can see the reason for the low set accuracy when analyzing the precision and recall. Whereas the recall is pretty good, the precision is really low. This indicates that the model predicts more labels than is accurate for the images.

To get a more balanced trade off between precision and recall and therefore a better accuracy, we can adjust the weights used for training. We rescaled the weights by dividing them by the minimum and applied another round of training to the now already pre-trained model from before (using the same validation set as to not distort the evaluation results):

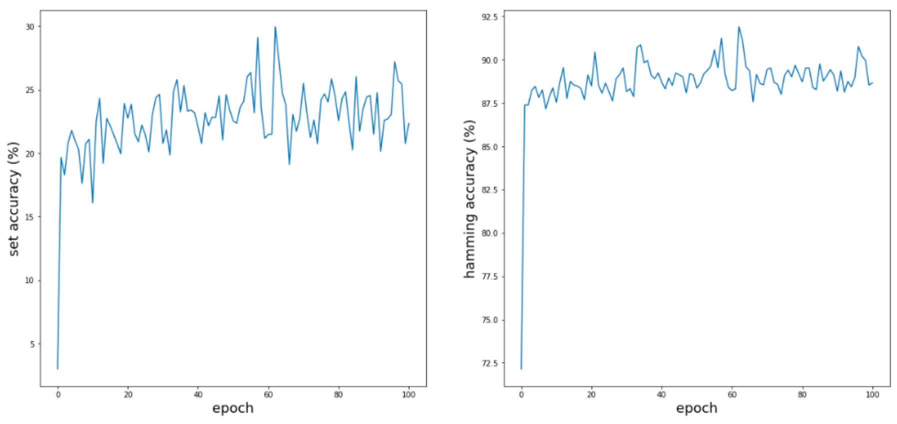


Figure 6: Accuracy

While precision and recall approach each other to about 0.3, the accuracies get to 90% (hamming) and 25% (score).

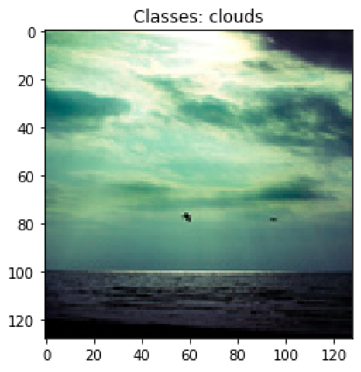
4 Error Analysis

After training is done, we analyze the predictions on the validation set farther to see where the problems lie.

4.1 Wrong Labeling

One reason for wrong predictions is just wrong labeling. Wrong labels in the training set negatively influence the learning and in the validation set, they can make the accuracy appear worse than it actually is.

```
prediction:  clouds, sea
```

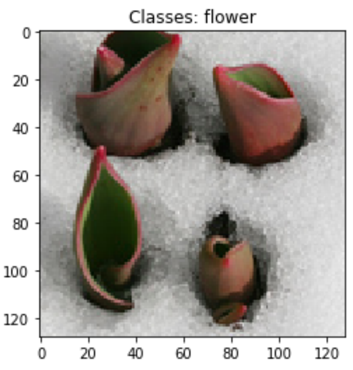


In this image for example, there should be a sea label.

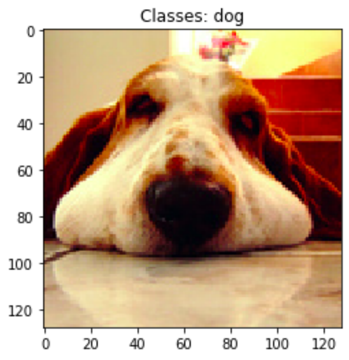
4.2 Data Sparsity

Another reason is the data sparsity in some cases. As seen in Figure 1 there are only a few examples for some classes. This leads to the model not being able to recognize a new image of that class like in the next examples.

```
prediction:  dog
```



```
prediction:  female, male, people, portrait
```

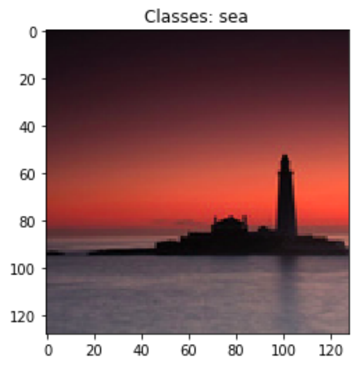


Both dog and flower appear relatively rarely and there are probably no examples of a head shot of a dog in the training set so the model confuses it with a human portrait.

4.3 Learning the Wrong Pattern

In some cases, the model learns a wrong pattern for a class like in the following example.

```
prediction:  clouds, sea
```

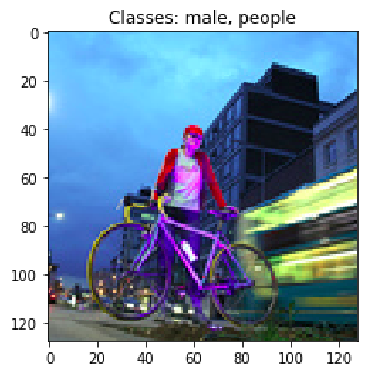


The model may have learned to recognize a prominent sky instead of clouds for the clouds category. This is also connected to data sparsity.

4.4 Unknown Reason

Then, there are some examples that don't have an obvious reason for failing.

```
prediction:  flower
```



To conclude, the most problems arise from the limited data set. If there would be more images to train with (and every image would be correctly labelled) and a more balanced data set, we could probably achieve a far better accuracy.