Deep Image Recognition University Project

Juuso Lassila, Kasperi Reinikainen, Matti Leinonen

Helsinki July 22, 2020

UNIVERSITY OF HELSINKI
Department of Computer Science

Contents

1 Introduction

We chose the image dataset for this project. In this report we will present our approach to find a well performing model to solve the image classification task. We start by analysing the dataset by doing some explorative data analysis. We then proceed to experiments to find best performing model architecture to solve our task. After that we present the results of our findings along with the final results with provided test set. We finally analyse the classification errors of our models.

2 Dataset

The dataset for this project had a total of 20000 images with the size of 128 × 128 pixels. The images were labeled with zero, one or more of the following labels: clouds, male, bird, dog, river, portrait, baby, night, people, female, sea, tree, car and flower. The table below shows the distribution of the images and their labels. Almost half (n = 9 824) of the images had no label.

Label	# of images
clouds	1095
male	2979
bird	360
dog	448
river	120
portrait	3121
baby	95
night	598
people	6403
female	3227
sea	173
tree	525
car	319
flower	761

A major concern that raised from the explorative data analysis was the imbalance of the class labels, described in the table above. As the table shows, the least frequent class label (baby) had only 95 occurrences where the most frequent had 6403 occurrences.

Based on these findings we developed data preprocessing modules to help tackle the data imbalance issue. The two main modules, namely data augmentation and data balancing are described below, and an experimentation round to evaluate effects of these methods are described in 4.1.

2.1 Data augmentation

A dataset of 20 000 images is relatively small for a deep learning task especially when you have to split this data into training and validation sets. Also as it was previously said, the distribution of image labels was uneven. For these reason we decided to use data augmentation to increase the size of out training set.

Whenever an image is used for training, there's a probabilistic chance that data augmentation is applied to the image. When an image is chosen to be augmented, move, rotation, scale, and shear transformations are applied to the image. In addition the image is flipped and gets noise applied. Each of these transformations and other changes have a probability of being applied. The amounts of these changes is also normally distributed, with mean being 0 change.

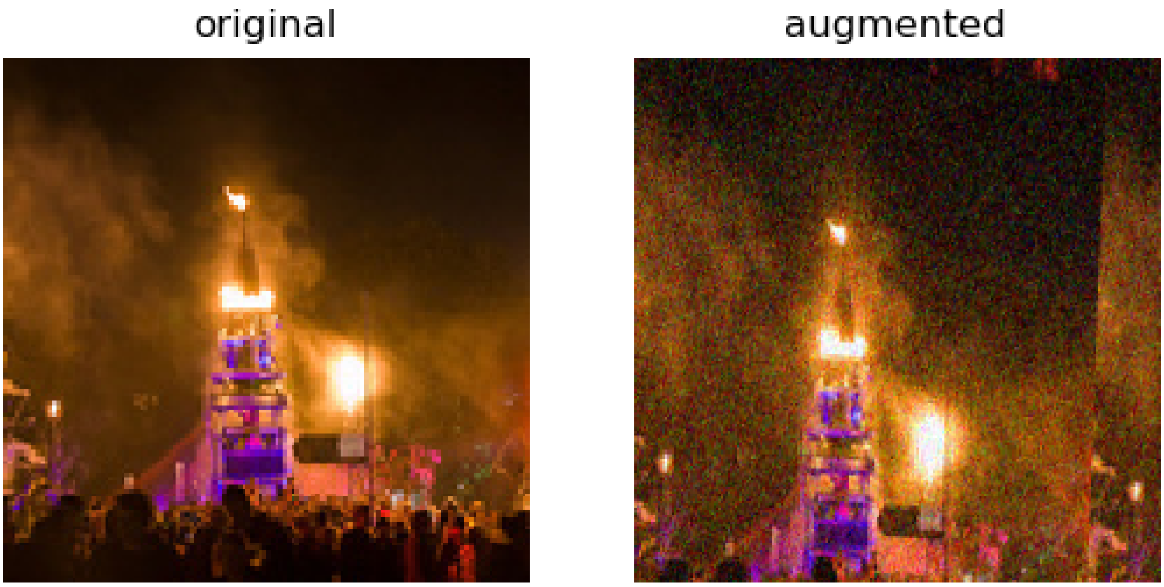


Figure 1: Example of data augmentation

2.2 Data balancing

Due to the poor balance of different classes, we created a data balancer, that over samples new images to the dataset. For each new image, we randomly selected a one form the samples that had the least common class. Before sampling new images, we had all the pictures in the dataset once.

One problem with this approach was that images with no labels were never added to the dataset by the balancer. Also, because people was the most common label, and it always existed with other labels such as male and female that were less common, those pictures with only the label people might have never been oversampled.

3 Technical details

Determining the final output layer and the loss function of the neural network that is suitable for the application in hand is an important part of the machine learning task.

Since our goal was to classify images with zero, one or more of the 14 labels, the final output layer of our different network architectures had 14 nodes, one for each label. Each of the neurons had a sigmoid activation function which made sure that the final output of that neuron was between 0 and 1. This corresponded to the probability that the input image had that specific label.

3.1 Loss function

Our loss function calculated the sum of cross-entropy losses for each output neuron. Sometimes our model predictions were exactly 1 or 0 due to floating-point errors. Because these numbers caused the loss to become infinite when the prediction was wrong, we preprocessed the model outputs $\hat{y}$ using function $s(\hat{y}) = 0.99\hat{y} + 0.005$. This way, the output numbers inputted to the loss are between $[0.005, 0.995]$ preventing the infinite loss. The final loss function is:

$$\mathcal{L}(y, \hat{y}) = -\sum_{c=1}^{C} y_c \ln(s(\hat{y}_c)) + (1 - y_c)\ln(1 - s(\hat{y}_c)) \tag{1}$$

Here C is the number of classes and c iterates over them.

3.2 Optimizer

We used Adam optimizer with a learning rate of 0.001. We didn't put any effort into trying to experiment with different optimizers or their hyperparameters.

3.3 Metrics

Metrics that we used to evaluate the performance of our neural networks included accuracy, precision, recall and f1-score. The best values for each of these metrics is 1 and the worst value is 0.

The accuracy metric calculates the fraction of correct predictions, precision score calculates the ratio of true positives and the sum of true positives and false positives tp/ (tp + fp) and recall is the ratio of true positives and the sum of true positives and false negatives (tp / tp + fn).

Precision measures the classifiers ability to not label negative samples as positive where as recall measures the classifiers ability to find all the positive samples. The f1-score is the weighted average of precision and recall. The f1-score is calculated as f1 = 2 * (precision * recall) / (precision + recall).

4 Experiments

We performed experiments with different convolutional neural network architectures and data preprocessing heuristics. Our objective was to find a model to accurately predict correct class labels for unknown image samples based on a set of known training samples in a multilabel classification setting.

4.1 Finding best data preprocessing settings

After initial ad-hoc experimentation we selected a relatively well performing model and started the more systematic testing rounds by first testing the effect of different data preprocessing configurations on the model performance.

The model architecture used for tests here was a transfer learning model where the basis was a pretrained ResNeXt-50 model. The final classification layer of the model was replaced and trained for our task with 14 output neurons. We only calculated the gradient for the final layer weights and trained them while leaving the other weights of the model untouched.

We tested three modules for preprocessing the dataset that we had developed for the project. The first module splits and optionally shuffles the data based on given train/test split. The second module, data augmentation (2.1) creates new augmented data samples. The third module, data balancing (2.2), balances the dataset by oversampling under-represented class samples.

In the first test only the first data processing module was used by just shuffling the dataset and splitting it into subsets of sizes 17000 for training, 1500 for validation and 1500 for testing. The second test extended the first one by applying data augmentation. The third test extended the previous two tests by also performing a data balancing, which caused the training dataset to grow from 20000 samples to 80000 as a result of generating new samples of underpresented classes.

4.2 Transfer learning

As our training dataset was relatively small, we shared a strong intuition that a pretrained computer vision model should serve as a good basis for solving the task. With this approach we would only need to fine-tune the network to classify items in the domain of our dataset, and most of the representation power could be learned a priori.

We first tested three different pretrained models available in Pytorch with equal training configurations. The tested models were ResNeXt-50 (50-layers), ResNeXt-101 (101-layers) and Densenet. In all cases we applied a custom final fully connected layer with 14 output neurons and no custom hidden layers. We only calculated the gradient for the final layer weights and trained them while leaving the other weights of the model untouched. For each test we trained the models with 6 epochs.

We then selected the best performing pretrained models and tested if classification performance can be further improved by adding a custom hidden layer between the pretrained network and the output layer. We tried hidden layer implementation for ResNeXt-50 and ResNeXt-101 models as they had better performance than Densenet. The hidden layer was a fully connected layer with 256 neurons and ReLU activation function, and without regularization. We used 6-10 epochs to train hidden layer models, but no improvement was detected from training beyond 6 epochs.

4.3 Convolution neural network models

We experimented with different convolution neural network architectures. First we created a architecture framework that is based on ResNet and Inception blocks. This framework took a configuration object that defined the different inception blocks and how many of them there were as well as the ResNet inspired shortcuts. By changing the config object we tried different architectures by making small changes and seeing whether they brought any improvements.

4.4 Ensemble of transfer networks

Lastly, we created an ensemble of transfer models. Here we inputted the same image input to two different transfer models with different architectures. For each, we took the output of the 2nd last layer before fully connected and concatenated them. Then we used a fully connected layer to predict the labels based on the concatenated outputs. We only trained this one fully connected layer. For the ensemble we used the ResNeXt-101 and the DenseNet-201 networks.

5 Results

We tested the different model candidates by comparing the classification results given each neural network to a test dataset drawn from the provided training dataset. In each test we used 17000 samples for training, 1500 samples for validation during training and 1500 samples for testing models after training. All results reported here are testing set results. Our primary metric was the f1 metric.

After the intermediate model selection experiments we proceeded to use an ensamble of models we had found to work well in our domain and performed test on the ground truth test set released on Friday 29.5. This test presents our contribution to the contest of finding the best image classification model.

5.1 Data preprosessing experiments

Results from data preprocessing tests (section 4.1) indicate that data balancing (oversampling) had a strong negative impact on model performance by pushing especially precision down heavily, which caused the f1 metric to be just 0.624. On the other hand data augmentation did not seem to have a significant effect on model performance, as the test iterations for tests with or without data augmentation varied in range between f1 0.7 and 0.715. Concluding from the results, from this testing round on we decided to keep data preprocessing steps constant by having data shuffled and split to 17/1.5/1.5 sets and by applying data augmentation in the consecutive tests.

5.2 Transfer learning results

Our initial results with transfer learning models were excellent. The f1 score for these models was usually between 0.70 and 0.73. When we looked at examples of the predictions, the models seemed to predict quite well. More often, it felt like the wrong labels were more due to missing labels in the data than just wrong predictions.

ResNeXt-50 and ResNeXt-101 models with fully connected 256-neuron hidden layer produced results between f1 0.715 and 0.735 on test dataset. This was not a major improvement to the performance achieved on tests without hidden layer given that the number of parameters to train grew significantly and causing also the training time to grow.

5.3 Convolution network results

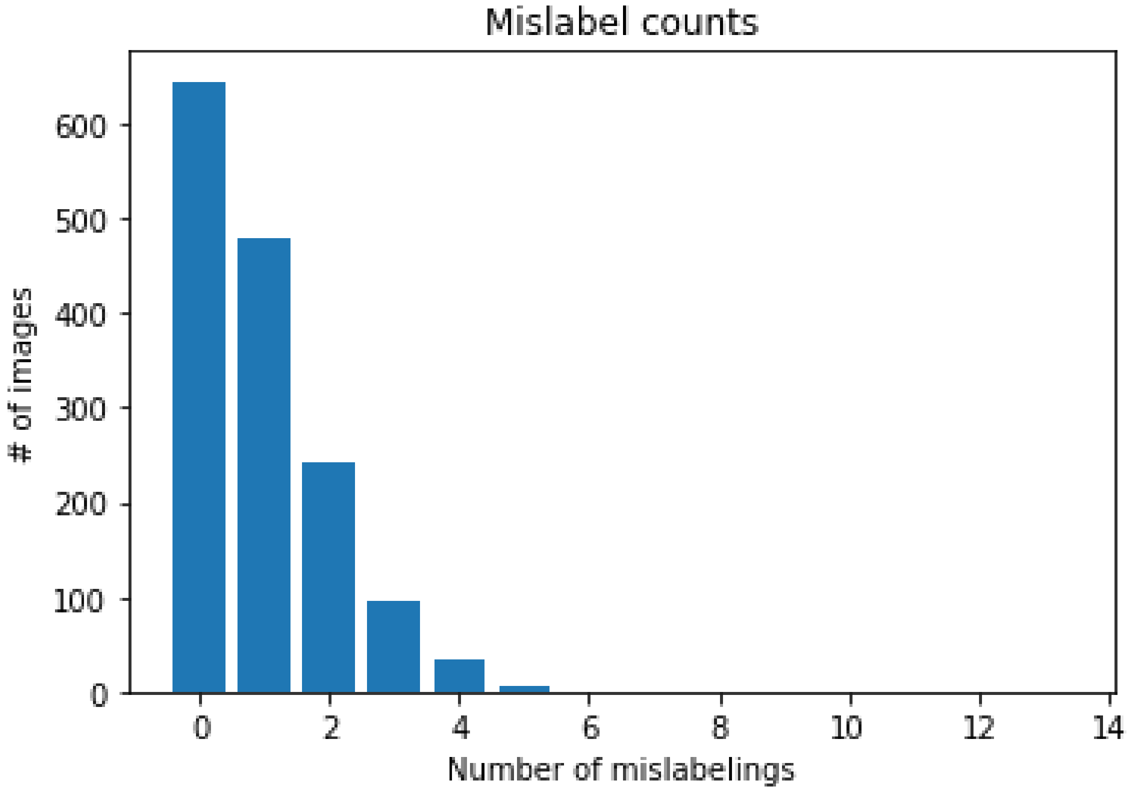
The convolution networks we created from the ground up ourselves were not as good as the transfer models. Due to using our computers for training, the duration of epochs was quite high, usually around 20 minutes. Simultaneously, training these networks all the way through would have required a lot more epochs than we were able to do to get to the same levels of accuracy. At best, our simpler models reached the f1 score of around 0.4 after multiple epochs and then plateaued around that score. With these results, we decided to focus on using the transfer models.

5.4 Ensemble network results

We didn't do very extensive experimenting with ensemble models. However, it seemed that the f1 score was often above 0.73, which is a small improvement over the original single transfer models. Due to the slight improvement, we decided to use the ensemble model for the competition's test data prediction. The final model had a f1 score of 0.7731, where precision was 0.7790 and recall was 0.7673. Accuracy was 0.9679.

6 Error analysis

We performed error analysis on a split of the dataset that we didn't use during training. We noticed that most images had a single mislabeling at most, where as images that had four of more faulty labels were quite rare.



In the model's error patterns, we noticed two different patterns. Firstly many images had missing labels, and secondly, the images with people often had some errors.

6.1 Missing labels in the data

We found many examples where the images were wrongly labeled. One case in our validation set was a river going through a town. The river was visible and covered the majority of the image area. Despite this, the images were not labeled with the river label. Our transfer model, however, predicted that the image contained a river.

We think that wrongly labeled images in the validation set caused a significant decrease in our accuracy. Incorrectly labeled images in our train set also probably caused our model's quality to decrease.

We decided not to add correct labels to the dataset, as it would have taken too much time and resources. Either we would have had to find these mislabeled images manually or would have had to research and implement active learning methods.

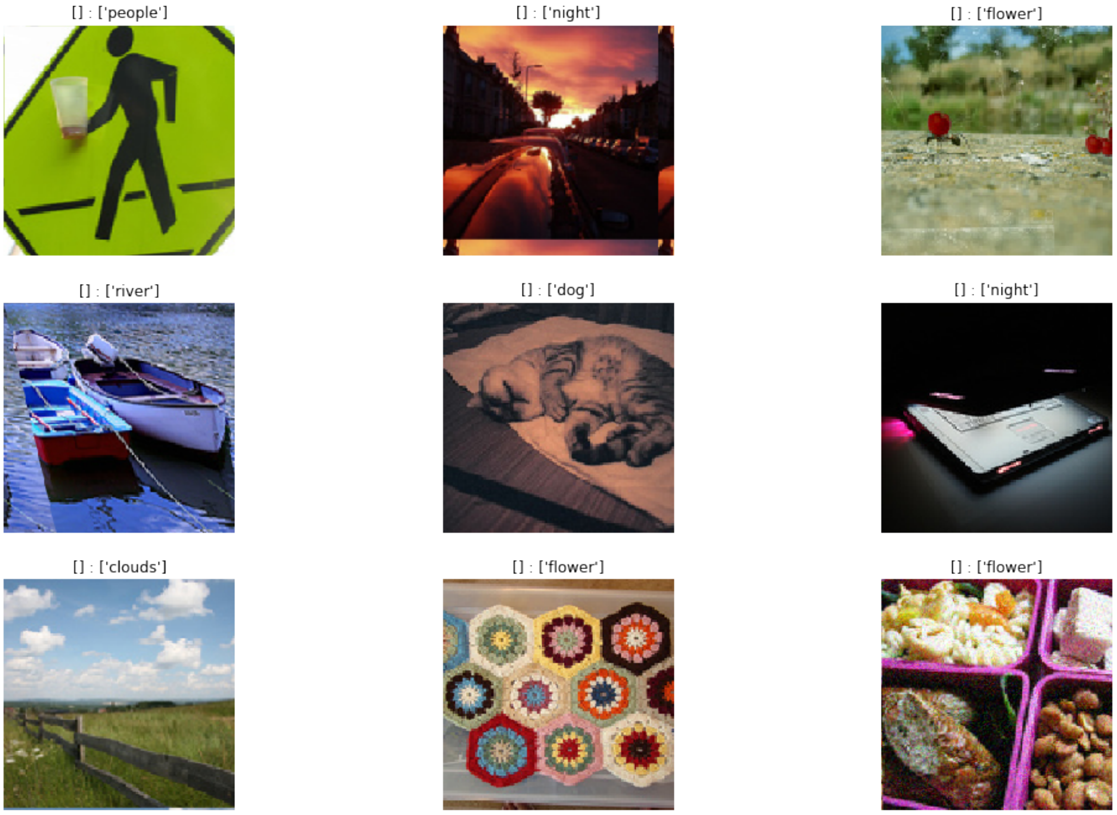


Figure 2: These images had no labels but many of them were given a reasonable label by our classifier.

6.2 Wrongly labeled predictions

Images with people turned out to be the most challenging task for our models. Below is a table of mislabelings for each class. Any time a image either has a label and it was not classified to have that label or a image didn't have that label and it was classified to have that label, it was counted. We can see that labels male, portrait, people and female had the most mislabelings.

Label	# of mislabelings
clouds	88
male	221
bird	36
dog	47
river	98
portrait	139
baby	91
night	108
people	220
female	176
sea	26
tree	26
car	29
flower	110

For example, the model had difficulty differentiating between males and females. These two subjects are often not that different looking from each other. Also, images often contained both a male and a female, which could have easily confused the model, making it learn male's features to predict female or vice versa.

Another example was predicting between people and portrait. These often appeared together, which could have caused portrait to learn similar features as people, making these appear together very often. It didn't help that the people label, while being plural, was commonly used when there was only one person in the image.

Labels unrelated to people seemed to be quite easy for our model. There probably were some errors, such as confusing some animals as dogs. But mostly these performed well.

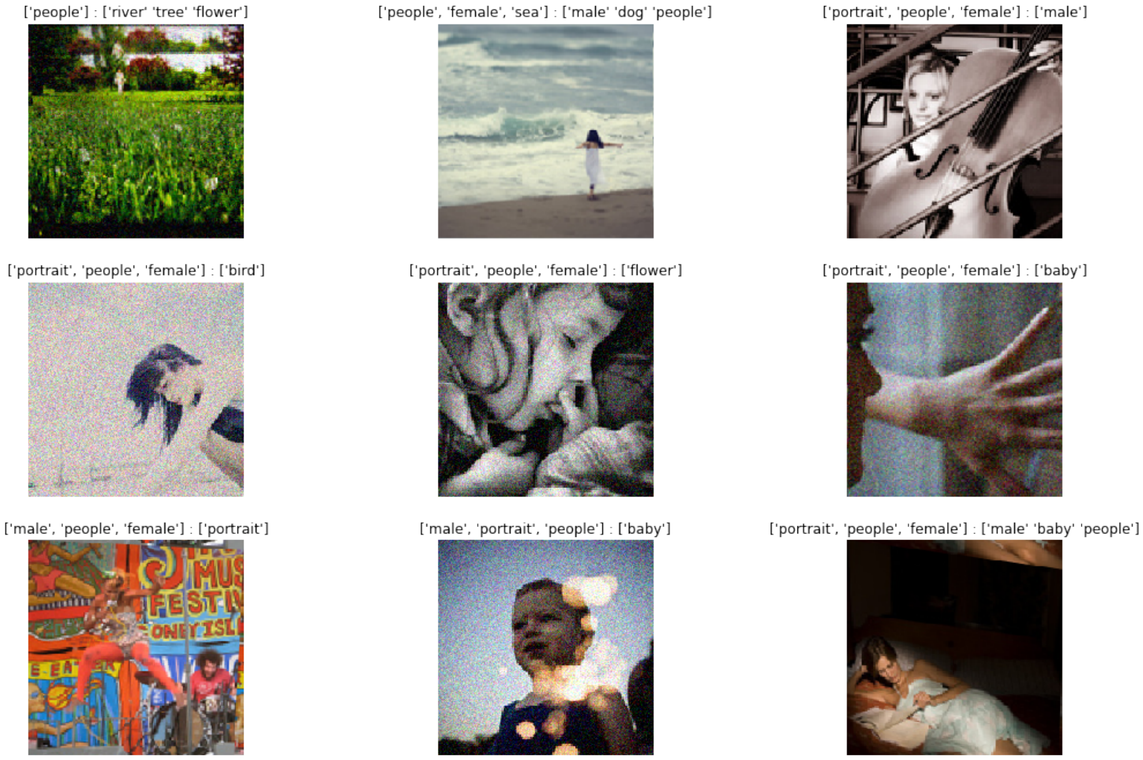

['people'] : ['river' 'tree' 'flower']
['people', 'female', 'sea'] : ['male' 'dog' 'people']
['portrait', 'people', 'female'] : ['male']
['portrait', 'people', 'female'] : ['bird']
['portrait', 'people', 'female'] : ['flower']
['portrait', 'people', 'female'] : ['baby']
['male', 'people', 'female'] : ['portrait']
['male', 'portrait', 'people'] : ['baby']
['portrait', 'people', 'female'] : ['male' 'baby' 'people']

Figure 3: Original labels are on the left and the labels given by our classifier are on the right of the colon sign. These are examples of images with most wrong labels given by our classifier.