Deep Learning 2018: Image Project

Hannu Pelttari & Noora Mäkelä

Team: Scallop

We decided to use transfer learning in this project since there exists multiple pretrained networks in the torchvision library. The pretrained networks have been trained on ImageNet. It was expected that the pretrained networks would perform relatively well out of the box and would only require some fine tuning.

We chose to start experimenting with the resnet18 pretrained network. The pretrained networks take as input images with sizes at least 224x244 so we had to rescale all of the images. The greyscale images had only one channel so we duplicated the greyscale images to three channels. We also split the training set to 16000 training examples and 4000 validation examples to test the performance of the model.

The first modification we did to the network was to replace the last linear layer with a new one. The existing linear layer had output size of 1000 and we wanted an output size of 14 so we replaced it with a linear layer of 512 input size (same as the original) and 14 output size. We wanted the output to be a one-hot encoded vector containing value 1 for the classes that the image belongs to and 0 for the classes it does not belong to. To achieve that we added a sigmoid activation function after the linear layer to get probabilities for the image to belong to each class.

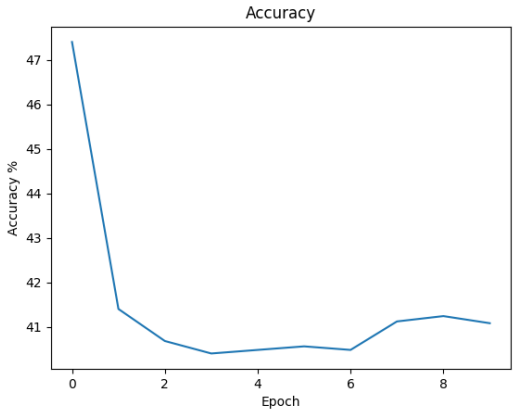
We started the experimentation by freezing all the weights of resnet except the last one and only training the last linear layer that we added. We used cutoff of 0.5 for the probability to belong to each class. We started measuring the accuracy by looking at how many images were completely correctly classified (meaning all the labels are correct, i.e. the one-hot encoded vector is completely correct). Later we also tried to use the f1-score as the accuracy measure but for some reason it did not work when we sent to code to ukko2 to use gpus. Calculating the f1-score and training worked fine on a local machine but took a long time so we decided to abandon that and use the accuracy measure of completely correctly classified images.

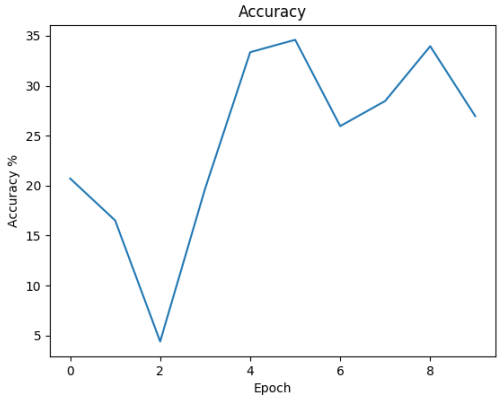
The optimizer we used was the Adam optimizer with learning rate of 0.01. The loss function was binary cross-entropy. With this setup we achieved an accuracy of around 38%. Next thing we tried was to train all the layers and the accuracy increased to above 49%. From then on we always trained all the layers. Next thing was to try different cutoffs for the probability to belong to each class. We tried values of 0.25, 0.35 and 0.65 but the accuracy was worse with all of these compared to cutoff of 0.5. The accuracy for the cutoff of 0.65 was pretty close to the accuracy for the cutoff of 0.5 so more thorough testing of the cutoff values could have provided better results.

Next thing to experiment was different networks, namely VGG16 and resnet34. Performance of VGG16 was pretty terrible: 8.8%. The training also took a long time, so we didn't experiment and tweak it any further. The performance of resnet34 was similar to resnet18 but the training took longer so we decided to stick with resnet18.

We then tried different learning rates between 0.1 and 0.001 but the initial learning of 0.01 that we chose was the best one. We also tried adding different transformations such as random crop and random flip to the data but for some reason the training did not seem to work correctly with those and the accuracy stayed exactly the same though the epochs. Due to time constraints, we didn't have enough time to debug that so we decided not to use transformations.

Some of the things that we didn't have time to try but could have probably helped are using class weights to help with the imbalanced data, train the final layer first for a couple of epochs and then the whole net or add transformations (which we tried unsuccessfully). Also using for example BCEWithLogitsLoss as a loss function could have helped since the computation of the sigmoid would have been more numerically stable. Also using the f1-score as an accuracy metric could have given a better picture of the performance of the model. We considered just how many labels were overall correct but that seemed difficult to interpret since the accuracy was always above 90% due to the sparseness of the label vectors. Cross validation would have also helped to get better sense of the level of accuracy but the training was quite time consuming so we decided not to do cross validation and spend the time tweaking the hyperparameters of the model. Other things we also didn't have time to test was other optimizers and adding more layers to the model.





Figures 1 & 2: Validation set accuracy with cutoff of 0.65 (left) and 0.25 (right).

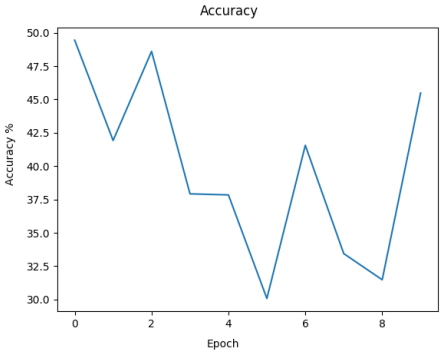


Figure 3: Accuracy with shuffled dataset.

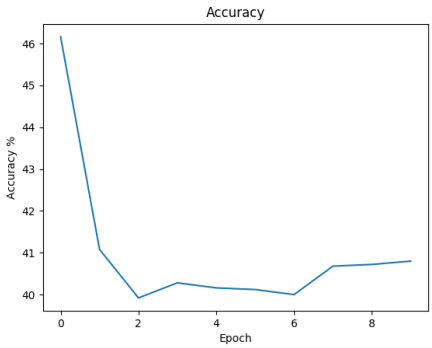


Figure 4: Accuracy when training the last layer for 5 epochs and then the whole net for 5 epochs.

In conclusion, the performance was quite poor and we noticed too late that the network was overfitting heavily since a large number of the training data didn't belong to any classes. This meant that we achieved the almost 50% accuracy by assigning no labels to any images and the final model did in fact predict only vectors full of zeros. Unfortunately we did not have time to remedy this. Assigning proper class weights or creating an extra class of "other" might have helped.