COMP 551 Mini-Project 3

Multi-label Classification of Image Data

Elias Tamraz - 260871813,
Saksham Mungroo - 260768072
Julian Armour - 260804046

Introduction

This project involved setting up a model to solve the image classification problem described above. It was determined that a convolutional neural network was the best fit for this task since it proved itself to have high expressivity and accuracy on image classification tasks during the last few years. Through hyper-parameter tuning and trial-and-error method, a good expressive model was found (*See table in appendix for detailed model architecture and parameters*).

The goal of training this model was to learn the best weight-matrix parameters for the different layers, which give the high expressive power of the convolutional neural network. Those weights were learnt using stochastic gradient descent as the optimizer (backpropagation of the weights). All methods used in the optimization step such as reverse-mode auto-differentiation were provided by the pytorch package. The optimizer (SGD) was put on a scheduler so that the learning would automatically decrease its value whenever a plateau was reached providing us with the best weight possible leading to very good accuracy results.

Cross-validation was considered to find the best architecture and hyper-parameter combination. However, 5-fold cross validation is very expensive, so we decided to leave out 15% of the data for validation. Since we report the validation accuracy at each epoch, this vastly improved our training time. It also proved to be sensible as, which will be shown below, the test accuracy on kaggle was very close to the reported validation accuracy. Our final architecture, a Residual Network, achieved very high training, validation and test accuracies as well fast training time.

Model

Before arriving at Residual Networks, many other networks were tested such as LeNet-5, AlexNet, VGG-16, VGG-19, our own VGG-style networks. Out of these, the ResNets still performed better or at least as well and were much faster to train. It should be noted that the ResNets performed this well *without* batch normalization being used. Once we added batch normalization, training time to get to above 95% went from around 30 to only about 5 and was accompanied by even validation accuracy. We decided then to put a lot more time into trying out several ResNet depths. Unlike the original ResNets proposed by researchers we applied a skip connection to each convolution instead of skipping two because it performed just as well, meaning the extra convolution layer does not improve the expressiveness of the model. To arrive at our final model, we started with three “blocks”, for which each contained several convolutional layers. The input was downsampled between each of these “blocks”. This means the image was downsampled twice by cutting the resolution in half on both the height and width. We iteratively added 1 layer to each of these blocks to see which improved the accuracy the most. The layer that improved the accuracy the most was kept and a new iteration was started. We continued these iterations until the accuracy stopped improving. Like other residual networks, a final global average pooling layer was added and fed into one fully connected layer that produced the label, a 55 entry vector.

The cost function used was a 5-way Cross Entropy. By this we mean that cross entropy is applied to every 11 entries in the label vector and calculated separately, then those losses are averaged together to get the final loss, this is performed automatically by reformatting the label and using PyTorch's CrossEntropyLoss class.

Data is preprocessed to be normalized to the [0, 1] range. We attempted to normalize the data using Z = (x - mean)/std.dev but this surprisingly yielded slightly worse results. We also attempted to use data augmentation with translation and rotation affine transformations but these did not improve results and slowed down training by a large margin. We also tried using the Adam optimizer but this gave worse results compared to the standard SGD no matter what learning rate was tried.

Once our best model was selected, we tried many combinations of learning rates, momentum, scheduler factors, scheduler patience, and batch sizes. The best combination we found is shown in the appendix.

Results

The model was trained on 85% of the dataset and the remaining 15% percent was used for validation. Once we were satisfied with our model, the test labels were predicted, the test accuracy was reported and the results were uploaded to Kaggle. The following results were obtained:

Best training accuracy	Best validation accuracy	Best test accuracy (Kaggle)
100%	99.738%	99.761%

Figure 1: Table presenting the training, validation and test accuracies of our Residual Network model.

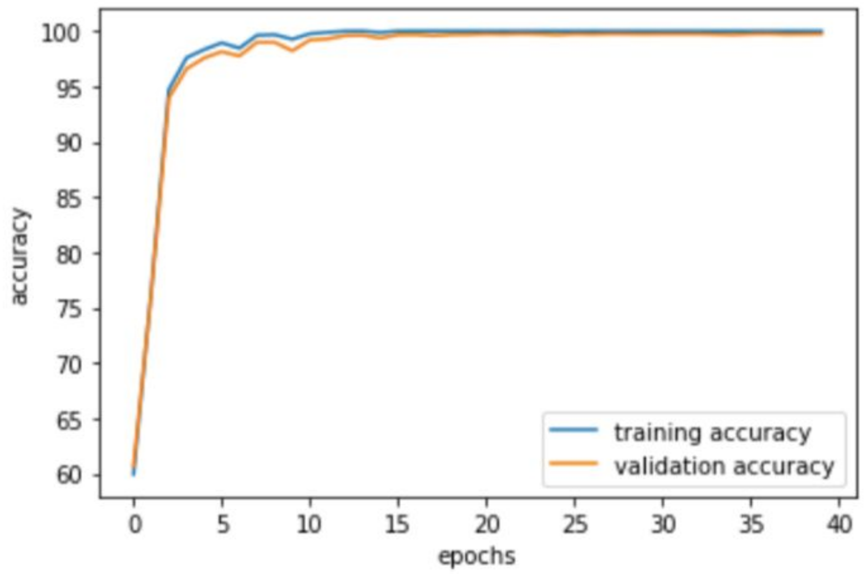

Figure 2: Graph presenting the training and validation accuracy of our model as a function of the training epochs.

Discussion and conclusion

The results obtained approved our choices of architecture and model parameters. Indeed, high validation accuracies and test accuracies alongside high training accuracies demonstrates a very expressive model which doesn't overfit and which will do a great job at predicting labels for this particular task. From the graph we can see that the training and validation accuracies evolve tightly close together, demonstrating that our model generalizes well on our dataset. Moreover, it can easily be observed that the accuracies increase rapidly to reach ≈ 99.7% within the first 5 training epochs where it starts to stabilize suggesting that the model learns the classification task really fast. (Part about why residual network + why batch normalization).

Appendix:

Model architecture Table

Hypeparameters

Percentage of the trained data	15%
Optimizer Momentum Factor	0.9
Optimizer lr (learning rate)	0.01
Scheduler factor	0.90
Scheduler Patience	3
Scheduler Threshold	0.01
Batch size	32

Architecture

Image Output size	Layer
64	[conv=3x3, channels=32] x 2
32	[conv=3x3, channels=64] x 6 Downsample with stride=2
16	[conv=3x3, channels=128] x 5 Downsample with stride=2