Mini Project 3: Multi-label Classification of Image Data

Aaron Nadal[a], **Diallo Oballa**[a], **and Viet Hoang**[a]

[a]Department of Computer Science, McGill University, Montreal, Quebec, Canada

In this assignment, an 18-layer Residual Neural Network was used for multi-label classification on a modified MNIST dataset. After training using stochastic gradient descent on a binary cross-entropy loss function, our model was highly successful in labeling the images, producing a 99.619% accuracy on testing data. Data augmentation was implemented in an attempt to further improve accuracy, but no such improvement occurred and no conclusion could be made about this method's effect on performance.

Image recognition has become a major field of research due to its wide range of applications from lesion detection in medical imaging to facial recognition in photography. Convolutional neural networks (CNNs) are one of the most commonly used techniques in computer vision. The main problem with CNNs that initially arose is that deeper networks started to perform worse [1].

In 2015, a Microsoft research team (He et al.) proposed a new architecture involving residual learning (Fig 1) that allowed the CNN to more easily learn the desired underlying function $H(x)$. $H(x)$ can be defined as $H(x) := F(x) + x$ where $F(x)$ is the residual. The new architecture, named ResNet, learns $F(x)$ instead of $H(x)$. This allows the model to more easily learn the indentity mapping by just setting $F(x)$ to 0, giving $H(x) = x$ [1].

This architecture served as the foundation of the model used in this project to classify digits from a modified MNIST dataset. By using a residual framework, we were able to a deeper neural network without losing accuracy. We implemented an 18-layer residual neural network (ResNet18). He et al. built deeper networks (ResNet34, ResNet50, ResNet101 and ResNet 152), but we decided against using those models due to their training times. ResNet18 has 1.8×10^8 FLOPs while ResNet34, the next shallowest, has 3.6×10^8 FLOPs. Furthermore, the deeper networks were designed for ImageNet, a dataset containing 1.2 million coloured images. Our dataset contains only 56 thousand grayscale images, and using a deep neural network like ResNet 152 would likely result in over-fitting.

We found that our model classified the data very accurately, producing 99.619% accuracy on 60% of the test data on Kaggle. This was achieved without any significant preprocessing or image segmentation using third-party software such as OpenCV.

Datasets and Data Augmentation

We used a modified MNIST data set containing one-to-five digits ranging from 0 to 9. The number 10 is used to label images with less than five written digits.

In an effort to improve our model, we implemented data augmentation. The size of the dataset was increased by applying a transformation to the training images and appending it to our training dataset. Many different transformations were considered, but in the end only a shear of randomly chosen magnitude between $-25°$ and $25°$ was applied. Unfortunately, this addition of data augmentation lead to a decrease in validation and test accuracy. This may be because the shearing had a minimal effect on the images, leaving minimal difference between the original and augmented data. It is also possible that the images sheared to a greater degree confused the model since the digits may have been harder to recognizing. However, since the difference in accuracies between the initial and data augmented models was so minimal ($<0.05\%$), there is insufficient proof to conclude that the data augmented model is worse. These discrepancies can be attributed to the training algorithm getting stuck at a local minima in stochastic gradient descent, or due to random chance.

Methods and Hyperparameter Selection

Our choice of not segmenting the images meant we trained the model to classify 55 classes, which is sufficient to encode all the digits and all their possible positions. We decided on this approach since we thought it would be easier than training multiple neural networks, to recognize first the number of digits and then each digit separately. To be able to train on 55 classes, we one hot encoded the labels as an array of length 55 with each consecutive 11 elements representing a different digit.

For the actual neural network we used a modification of the ResNet18 model presented in [1] with the differences being a stride of 1, a kernel size of 5x5, and adaptive max pooling for the first convolutional filter. In the original architecture, a kernel size of 7x7 was selected, but we opted for 5x5 because our images were much smaller (64x64 compared to 112x112). After the first filter, there are four layers. Each layer is composed of 4 convolutional filters. The first of four filters doubles the number of channels and has a stride of 2 to halve the image dimensions. The remaining three filters all have a stride and padding of 1. After filter we applied batch normalization and ReLu activation. Batch normalization was applied because it regularizes the model (Lecture 18, Slide 25), and ReLu was selected because it doesn't suffer from the vanishing gradient problem like logistic or tanh. After the four layers we applied adaptive average pooling to further reduce the image size to 1x1 and finally we have a fully connected layer to make the output a single array of size 55. We do not have a final logistic activation layer because Pytorch's binary cross entropy loss function applies it internally to take advantage of the log-sum-exp trick.

Since our task was multi-label classification, our loss function was binary cross-entropy loss with logits. Before the loss is calculated, a logit followed by a logistic function is applied to every element in the prediction array.

We tested the effects of batch size on the performance of the model. We found that increasing the batch size reduced the time to train for each epoch (marginally), but validation and training accuracy decreased. With a batch size of 16, the training and validation accuracy reaches 98% by epoch 20, and with a batch size of 500, the accuracy plummets to 7%.

Results

We trained multiple models with varying number of epochs, ResNet architectures and batch sizes. Our highest performing model was achieved with the modified ResNet18 architecture over 200 epochs and with a batch size of 16 which resulted in an accuracy of 99.619% on the testing data. The accuracy of the model as a function of epochs can be seen in Fig 1 where you can see the training accuracy reaches 100% after around 5 epochs and the validation accuracy reaches a plateau after around 20 epochs. After this point the accuracy fluctuates over the remaining epochs but is maximum at 150 epochs.

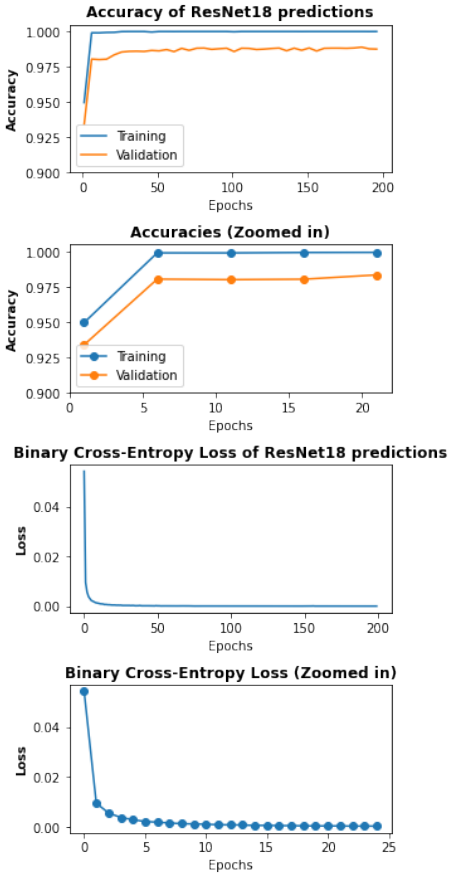

Fig. 1. The first two plots show the training and validation accuracies for each epoch. The first plot shows all 200 epochs while the second shows the first 20. The third plot shows the loss over 200 epochs and the last plot is the loss over 20 epochs

Conclusion and Discussion

Our 18-layer Residual Neural Network was highly successful in labeling images from the modified MNIST dataset, achieving 99.619% accuracy on the test set. We found that our model learned very quickly, obtaining nearly perfect accuracy in fewer than ten epochs.

Given access to greater computational power, we could perform deeper analysis on the effects of changing hyperparameters, specifically by running 5-fold cross validations on varying momenta, batch sizes, and learning rates. This would have allowed us to more effectively tune the hyperparameters to ensure our model performs optimally.

Statement of Contributions. All three members contributed equally to the project and were involved in all steps of the project.

References

[1] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

Appendix

Layer	Input Size	Output Size	Filters
1	1x64x64	64x32x32	5x5, 64, stride 1, padding 2 Adaptive Max Pool
2	64x32x32	128x16x16	3x3, stride 2, padding 1 3x3, stride 1, padding 1 3x3, stride 1, padding 1 3x3, stride 1, padding 1
3	128x16x16	256x8x8	3x3, stride 2, padding 1 3x3, stride 1, padding 1 3x3, stride 1, padding 1 3x3, stride 1, padding 1
4	256x8x8	512x4x4	3x3, stride 2, padding 1 3x3, stride 1, padding 1 3x3, stride 1, padding 1 3x3, stride 1, padding 1
5	512x4x4	1024x2x2	3x3, stride 2, padding 1 3x3, stride 1, padding 1 3x3, stride 1, padding 1 3x3, stride 1, padding 1
6	1024x2x2	1024x1x1	Adaptive Average Pool
	1024x1x1	1x55	Fully Connected

Table 1. Hyperparameters of the optimal model.