# Convolutional Neural Networks for Multilabel Classification of Image Data

Department of Computer Science
McGill University
Quebec, Canada

## Abstract

This paper investigates the performance of several convolutional neural networks on a multilabel image classification problem. We experimented with various architectures, including our own CNN design, VGGNet, AlexNet, and ResNet, observing that the ResNet50 model outperformed the other models, including its deeper counterparts, ResNet101 and ResNet152. We then tested various amounts and types of data augmentation to increase the diversity and size of our training set. Using random search hyperparameter optimization, we found the optimal numbers of epochs, batch size, and other transformation parameters for our models. We experienced the most substantial performance boost when we implemented and utilized two different bagging techniques, both of which decreased model variance greatly. Finally, we used semi-supervised learning to incorporate the unlabeled dataset and achieve our highest test accuracy.

## Introduction

The aim of this project was to train a model to predict two alphanumeric characters in noisy images. We observed that the ResNet50 model generalized best on unseen data while being relatively computationally cheap. Our preprocessing techniques included augmenting a subset of training data with randomly selected transformations for each image in the subset. We concluded that a 25% subset of the original data performed best. Furthermore, we implemented two different ensemble learning bagging methods, and utilized them to predict labels for the unlabeled dataset. Using this much larger dataset to bootstrap samples within our bagging technique, we achieved a top test accuracy of 94.66%.

## Datasets

The dataset we used in this project is the Combo MNIST, which consists of 30,000 labeled images, 30,000 unlabeled images, and 15,000 unlabeled test images. Each image contains one number and one letter, which may appear anywhere in the image, with different orientations, sizes, and styles as well as being affected by a variety of noise.

## Results and Discussion

Initially, we began our experimentation with heavy preprocessing. This included applying a Gaussian blur to remove noise, eroding then dilating to disconnect close characters, and finally using bounding boxes to individually extract each character. We then reconstructed each image by scaling and placing the extracted characters side by side. This was an attempt to rid the images of the various noise they were subject to by keeping them in a consistent format. For the model, we developed our own convolution neural network based on implementations found in research [1]. However, after several runs, we plateaued at around 71% accuracy so decided to explore different techniques.

With the help of the Kaggle discussion board, we found that our peers were utilizing preexisting network architectures such as AlexNet, ResNet, and VGGNet. Upon conducting further research, we found that VGGNet is simply an improvement of AlexNet, and thus, using AlexNet would be suboptimal; however, research was inconclusive with regards to VGGNets and ResNets [2]. Testing different implementations of each, we found VGGNets to be too deep, leading to overfitting and extremely slow training speeds. On the other hand, ResNets utilize skip connections, allowing them to avoid overfitting and be less computationally expensive [1]. We found the ResNet50 model, which consists of 50 deep layers, including convolutional, pooling, and dense layers, to have the best tradeoff between training time and performance on an unseen validation set compared to its deeper counterparts, ResNet101 and ResNet152. One potential reason why ResNet50 outperformed ResNet101 and ResNet152 with our data is that their increased depth is meant to account for more complex images [1]; however, it became evident that the images in our dataset were too simple for the deeper models, which led to overfitting. In the end, we fully implemented a ResNet50 model, employing a sigmoid function in the final layer to allow for multilabel classification [3].

The next step was to find ways to optimize the model and tune its appropriate hyperparameters. Since training each model was time consuming, we had to prioritize the most relevant hyperparameters, which we determined were number of epochs and batch size. Utilizing a random search method, we found the optimal number of epochs to be 50 and batch size to be 64. The relatively small batch size was consistent with what we learned in class about how the added noise has a regularizing effect, preventing overfitting. Utilizing these optimal hyperparameters, we trained a few models on the dataset and achieved accuracies of around 85%. Next, we tested various data augmentation techniques and found that by augmenting only a select sample, in this case a 25% subset of the training data, we improved performance to around 91%. The transformations we applied were randomly selected for each image in the subset from the following techniques: horizontal flip, vertical flip, 10-degree rotation, and increasing sharpness. We determined the optimal rotation angle of 10 degrees through experimentation and the Kaggle discussion board. The table below shows how the test accuracy changed with different subset size for augmentation techniques.

| Augmentation (%) | 100 | 50 | 25 | 10 | 1 |
|---|---|---|---|---|---|
| Test Accuracy (%) | 89.85 | 89.33 | 91.28 | 91.08 | 91.06 |

*Table 1: Effect of varying augmentation size on test accuracy*

Noticing that our models were beginning to overfit, we found the need to implement bagging techniques to decrease variance with little to no effect on bias. We used two bagging techniques to yield the best result. For our first method, single-model bagging, we created several bootstrapped samples from the training dataset, and for each sample, we would train a single uninitialized model on it and form a test-set prediction. We then used a mode-based voting technique to consolidate all test-set predictions into one. In the second method, multiple-model bagging, we provided slightly different multiple models that we had already pretrained ourselves, as input. For each model, we applied single-model bagging, this time using the aforementioned pretrained models that require less training epochs to yield accurate predictions. The reason we used pretrained models is to save resources while improving accuracy, which is a form of transfer learning [4]. In the end, we consolidated test-set predictions for each model and then consolidated them again into one, using the same mode-based voting technique. Both models had a large impact on our overall performance, resulting in a test accuracy of around 93%.

Finally, we began researching semi-supervised learning techniques for unlabeled data usage and found that we could implement the teacher-student paradigm [5]. Our multi-model bagging technique and previously trained models acted as our teachers, and we used them to form predictions for our unlabeled set. By appending the newly labeled data to our augmented training set, we created a bigger dataset that we then used to train five other models. These models, our students, each had small hyperparameter differences to introduce diversity in their predictions. Finally, we input the students into our multiple-model bagging implementation, this time forming our bootstrap samples from the larger dataset, and after consolidating these predictions, we were able to achieve our top test accuracy of 94.66%.

## Conclusion

In conclusion, we found that multiple-model bagging, using various ResNet50 models, and bootstrapping from the concatenated dataset which includes the original training data, an augmented dataset, and the unlabeled dataset, performed the best with nearly 95% accuracy. For future investigation, we would like to see the effects of other hyperparameters such as learning rate and momentum on our model's performance. Additionally, we would like to successfully implement the multi-model bagging technique, this time training each model entirely from scratch, which we were not able to do given computational and timing constraints.

## Statement of Contributions

All group members contributed equally and fairly to this project and were engaged for all sections.

## References:

[1] J. Wu, Lecture, Topic: “Introduction to Convolution Neural Networks.” National Key Lab for Novel Software Technology, Nanjing University, China, May., 1, 2017.

[2] A. K. D. Ravilla, “CNN based Image Classification Model,” International Journal of Innovative Technology and Exploring Engineering, vol. 8, no. 11S, pp. 1106–1114, 2019.

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Microsoft Research.

[4] L. Torrey and J. Shavlik, “Transfer learning,” Handbook of Research on Machine Learning Applications and Trends, pp. 242–264, 2010.

[5] Z. Yalniz, H. Jegou, K. Chen, M. Paluri, and D. Mahajan, “Billion-scale semi-supervised learning for image classification.”