# COMP 551 Mini-Project 3 Group 7

November 24, 2021

Rodolphe Baladi, Wassim Wazzi, Aidan Williams

## 1 Abstract

In this project, we explored the implementation of a variety of neural networks using different learning methods, and data processing techniques. The main challenges were determining the best architecture for our model - what to do at each layer of the model, how to normalize the data, and how to augment data.The models were tested on their ability to fit the training data, and accurately predict the number and letter values on unseen images. These metrics provided insight into how the model would perform against the test data through the Kaggle competition. We found our most effective model to make use of ResNet, data normalization and data augmentation.

## 2 Introduction

There were two primary objectives for this project. The first of these was to develop a neural network model to accurately classify image data. Additionally, we needed to optimize the hyperparameters, such as learning rate and batch size, to ensure our model learned well. As for the second objective, we aimed to process the data in order to make our model perform better. For this task, we used data normalization techniques, different schemes to denoise the images, and data augmentation.

## 3 Datasets

The datasets provided contained 30,000 labeled images, and 30,000 unlabeled images. Each image contained 1 character and 1 digit. For the labeled images, the labels were provided in the form of a binary string of length 36, where the first 10 bits represented the digit (0-9) and the 26 last bits represented the character (a-z). The images contained noise, and pixel values ranged from -100 to 600, unlike the standard 0-255 range.

## 4 Results

### 4.1 Model Architecture

In order to find the most efficient classifier, we attempted to develop different model architectures and compared their test accuracies. Our first attempt consisted of a 11 layers architecture using convolutions, max-pooling, and dense layers. We separated training on the numbers and the letters before concatenating the output layers. This model contained 8 million trainable parameters and obtained a 75% test accuracy. We then implemented a variation of VGG16 with images upscaled to 224x224 and 100 million trainable parameters. However, the VGG16 model overfitted the data, as validation loss significantly increased at each epoch.

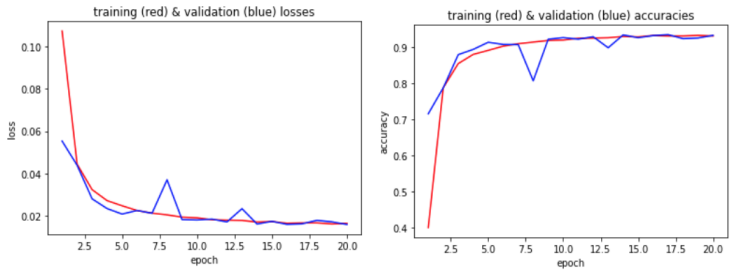


*Figure 1: Training and Validation Losses and Accuracies curves for ResNet*

Finally, we developed a ResNet model that achieved the best results. This architecture is deep without being too wide, which allows for more complex non-linear functions to be learned. Our ResNet18 relies on residual blocks

which create skip layers, and solves the problem of vanishing gradient in deep CNNs by allowing alternate shortcut paths for the gradient to flow through [1]. We used stochastic gradient descent with momentum and weight decay. This model achieved over 90% validation accuracy, as shown in figure 1.

## 4.2 Data processing

### 4.2.1 Normalization

Normalizing the inputs can make training faster and reduce the chances of getting stuck in local optima [2]. Since we used relu activations in our layers, we normalized all pixel values between 0 and 1, using the formula below. We also scaled pixels to the range 0-256, in order to denoise the images better. This achieved 92.9% test accuracy.

$$img = (img + |img.min|)/(img.max + |img.min|)$$

### 4.2.2 Image denoising

To determine if a model would perform better with or without noise in the data, we trained two separate models with and without and compared the validation accuracies. We tried two different approaches for this. For the first approach, we added random noise into the unlabeled data, and then used this and the original unlabeled data to train an autoencoder to denoise the images. After training the autoencoder, we used it to denoise the train, validation, and test data that we fed to our ResNet. As for the second approach, we attempted to address the noisy white pixels scattered in the image. We wrote an algorithm that iterates over every pixel and turns white pixels into black pixels if its neighbors are not white, as shown in figure 2. However, both techniques did not improve the model performance.

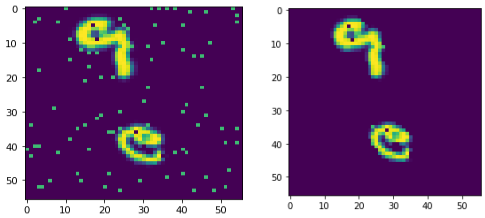

*Figure 2: Image before (left) and after (right) denoising*

### 4.2.3 Data augmentation

Our approach was to create new images by modifying our data on the fly. We performed slight modifications to the dataset during each epoch by applying a set of transformations consisting of rotations, height and width shifts, and image shearing. This allowed us to train our model on new data at every epoch, which improved the validation accuracy by 3%. This prevents the model from overfitting on the training set, thus making it generalize better [3].

# 5 Discussion and Conclusion

Throughout the project, it was evident that neural networks are complicated and opaque. After copious amounts of trial and error, different model architectures, and several submissions, we concluded that the combination of ResNet with data normalization and data augmentation gave the best performance.

## 5.1 Future Work

Further improvements on the image classification task could be achieved by applying semi-supervised learning techniques to make more use of the unlabeled data. Another option could be to use clustering algorithms, which are unsupervised machine learning techniques that group data together based on their similarities.

## 5.2 Statement of Contributions

Work was divided by section as shown in part 4 of the report. All members contributed to the code and write-up.

## References

[1]: Abdul, S. (2021, June 8). What is ResNet | Build ResNet from Scratch With Python. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/build-resnet-from-scratch-with-python/

[2]: B. (2020, October 19). Normalizing Inputs of Neural Networks. Baeldung on Computer Science. https://www.baeldung.com/cs/normalizing-inputs-artificial-neural-network

[3]: F. (2017, September 27). Data augmentation in PyTorch. PyTorch Forums. https://discuss.pytorch.org/t/data-augmentation-in-pytorch/7925