Miniproject 3 Report

Rezvan Sherkati
McGill University

Manoosh Samiei
McGill University

Aarash Feizi
McGill University

Abstract

In this project we performed a supervised prediction task on the Modified-MNIST dataset, which is a more challenging variant of the original MNIST. Our goal was to output the digit in the image with the highest numeric value. We developed convolutional neural network architectures for our model and also used famous image classification models: VGG16, MobileNet, and ResNet-50. Also in some models we used methods such as data augmentation to further improve the accuracy. We obtained the best performance with VGG16 network with a validation accuracy of 99.03% and an accuracy of 98.933% on Kaggle test set.

1 Introduction

MNIST dataset is a database of handwritten digits from 0 to 9. The Modified-MNIST is an altered version of the original MNIST, containing 3 digits with some noisy backgrounds. Our task was to predict the largest digit in each image using deep learning methods.

We applied two approaches to this task. First using convolutional neural network architectures, and second using image classification models: VGG16, MobileNet, and ResNet-50. While using these models, we tested training the architecture from scratch, using pre-trained model as feature extractor, and also fine-tuning the pre-trained model which means training the higher layers while freezing the initial layers. Among all architectures, VGG16 had the best performance, while it was trained using augmented training data with its weights pre-trianed on the ImageNet dataset.

The rest of the paper is organized as follows. Next we talk about the previous works relevant to this problem and then we elaborates on the dataset. After that we explain our proposed approaches and implemented models and then we present our results and models' accuracy. Then we talk about future works and also statement of contributions.

2 Related Work

Recognizing handwritten digits is a famous problem in image classification, and several researchers have been working on this. The authors of [9] have integrated the Convolutional Neural Network (CNN) and Support Vector Machine (SVM) models to build a hybrid model. The hybrid model takes advantage of the ability of the CNN to extract features and the SVM model to recognize and classify them. By testing their model on the MNIST dataset, they concluded that its recognition and reliability performance outperformed the previous works on this dataset.

Models with more biological background have also been presented. In [2], the authors present a biologically inspired model: the map transformation cascade (MTC). The MTC model first extracts features with respect to the given raw pixel data in a series of sequential layers. Finally the output of the final feature layer is fed to a linear classifier for classification, which in their case, the linear SVM was used. After experimenting the MTC model with the MNIST and the USPS dataset, they concluded that the MTC model is a great improvement comparing to networks using linear classifiers.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

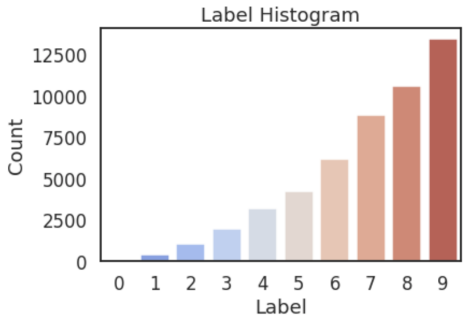
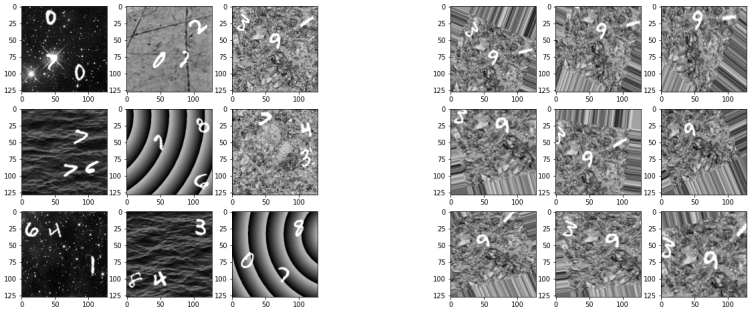


Figure 1: Histogram of Training Set Labels



(a) Modified-MNIST dataset. (b) Augmented Modified-MNIST dataset.

Figure 2: Images from the Modified-MNIST dataset and the augmented version.

3 Dataset and Setup

The original MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems [8]. In this project we used a modified version of the original MNIST dataset called Modified-MNIST. This dataset is a collection of 50000 images with three handwritten digits, randomly oriented, on a noisy background as seen in Figure 2a. The handwritten digits are chosen uniformly from the original MNIST dataset. The label of each image indicates the digit with the maximum value among the three digits. Due to the labeling procedure and the digits being chosen uniformly for each image, we cannot expect to have balanced distribution among labels. As seen in Figure 1 the count of each label is correlated to its digit value, i.e the larger the digit is, the most probable it is to appear in the labels. Therefore, label '9' has the most proportion and label '0' has the least.

For validating our models and hand-picking the best ones, we used 90% of the training data for the training and the remaining 10% for validation. To compare different models with each other, the same data split with a specific random seed was used for each model.

Also as we will also explain in section 4.2, we apply Data Augmentation to the dataset in order to avoid over-fitting and improve the accuracy. The sample images from the augmented version can be seen in Figure 2b.

4 Proposed Approach

We trained several architectures of neural networks for this task. Some of these models were trained from scratch and some were trained using transfer learning on a pre-trained model that was already trained on ImageNet. By validation, we were able to compare their results and choose the best one. Now we explain each of the experimented models briefly.

Table 1: The accuracy of different CNNs on the validation set. The models that used augmented data are specified with *aug* subscript.

Model	$VGG16_{aug}$	$VGG16$	CNN	CNN_{aug}	$ResNet50$	$MobileNet$	$MobileNet_{aug}$
Accuracy	**99.03**%	97.4%	**99.81**%	97.10%	92.74%	94.9%	96.6%

Convolutional Neural Network (CNN) was first presented by Alex Krizhevsky et al. [7] and has improved a lot since then. This model stacks multiple convolutional layers and by using non-linear transformations after each layer and back-propagation for updating the model's weights, learns filters for classifying images. The further architectures discussed in this section are made of the different arrangements of CNN layers.

For constructing our CNN model, we used an architecture with 6 convolution layers which had depths of 64, 128, 256, 256, 512, 512 and finally a dense layer with 10 outputs, representing each class (digit). The activation functions for the hidden convolution layers were *ReLU* and for the output layer, *Softmax* function was used.

ResNet-50 is a CNN based architecture with weights pre-trained on ImageNet dataset and has 50 CNN layers and 25,636,712 parameters. The idea in ResNet-50 is adding "skip" and "residual" connections between layers in order to avoid some problems such as vanishing gradient.

Also, we used a **MobileNet** model [4] with weights pre-trained on ImageNet dataset. We implemented two versions of this network. The first version has 4,253,864 parameters and the second has 3,538,984 parameters. Both versions are composed of 88 layers. MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks. They have a smaller size (15 MB) compared to other similar architectures.

VGG16 [11] is a CNN-based architecture with 16 layers and 138,357,544 parameters. It was developed by a group at University of Oxford and is an improvement over AlexNet. This model was tested with multiple number of dense layers at the end which will be explained in Section 5.

4.1 Fine-tuning (Freezing Initial layers)

As the size of the dataset is relatively small (50,000 samples), we can keep some of the initial layers pre-trained on ImageNet and freeze their weights to make use of their pre-learned representations. Also as the similarity of training dataset to pre-trained dataset (ImageNet) is not very high, we can train the deeper layers, which extract more abstract features of the input data, to customize them to the new data. This procedure is called fine-tuning.[3]

We tested this approach on the VGG16 model by freezing the initial CNN layers of block 1 to block 4, and training the fifth block convolution layers as well as the final dense layers on the dataset. While freezing the network we set the learning rate to 10^{-5}, so that we limit the magnitude of the modifications to the representations of the layers we're fine-tuning.

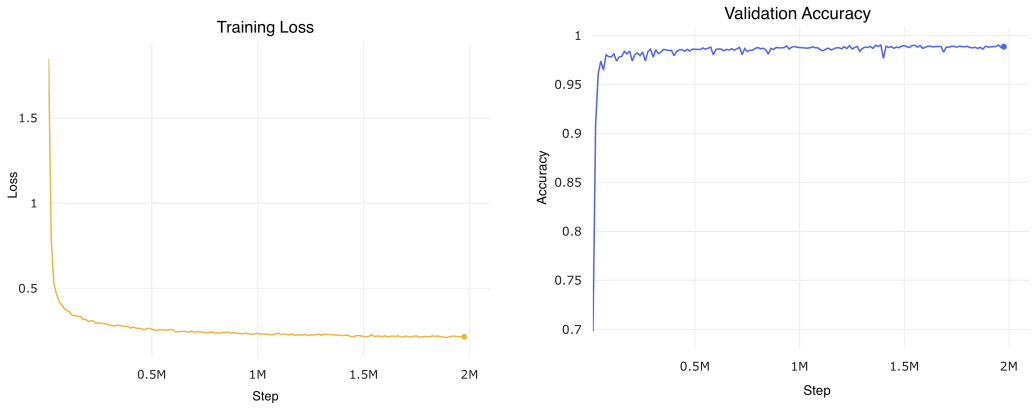
However, we did not find any improvement and our model's accuracy did not change significantly. Hence, the version with no frozen layers was used for the final version of the model.

4.2 Data Augmentation

In order to further improve the accuracy of the model, we use a technique called Data Augmentation. In Data augmentation we increase the amount of training data by adding minor alterations, such as rotation, re-scaling, shifting, zooming, and shearing to the images of our training data. The results of the augmentation process for a single image can be seen in Figure 2b. Data augmentation is helpful in reducing the over-fitting of the model by adding more variation to the training data [10]. We adjusted the setups of data augmentation, such as the amount of rotation, zoom and etc, based on the nature of the problem and our models, which can be seen in details in the code.

5 Results

At the beginning, all models were trained on the original data (not augmented) and validated according to the data splits. The parameters in VGG16, ResNet-50, and MobileNet were initialized with their



(a) Training loss of VGG16.

(b) Validation accuracy of VGG16.

Figure 3: Plots from the VGG16 with data augmentation ($VGG16$). Comet-ml was used for plotting these figures.

pre-trained weights on the ImageNet dataset and the CNN model was initialized randomly. We tested both stochastic gradient descent (SGD) and Adam optimizers for all models; SGD worked best for VGG16 and CNN, while Adam lead to a better performance for MobileNet and ResNet-50. We used the default learning rates of 0.01 for most of the models and we also tested 0.003 learning rate for VGG16 model, which decreased the performance. We have used the default values for decay rate and momentum. Due to the nature of the problem, which is a multi-class classification task, we used *categorical cross-entropy* loss function. Since CNN, VGG16, and MobileNet had better validation accuracy, they were experimented once again with the augmented training data. The results of these experiments can be seen in Table 1. We also tested MobileNetV2 (a second version) but it had a lower validation accuracy compared to the first version of the model. Furthermore, we tested these models with various hyper-parameters and chose the best ones for further experiments. Particularly we realized that a batch size of 4-8 for both training and validation data works best on VGG16 architecture. With batch size of 32 and above, VGG16 sometimes over-fit to training data. Decreasing the batch size solves the over-fitting problem and works like adding regularization to the model [6].

During the training time, we used *Checkpoint* callback feature in Keras, to save the new model in each epoch when its validation accuracy improves, and ignore the model if its accuracy decreases.

Although the CNN model gained a high validation accuracy (99.81%), the accuracy on Kaggle test set was 97.63%. This showed that the weights may have been partially over-fitted on the local training and validation data. After training this model with the augmented dataset, we observed a decrease in the validation accuracy of the CNN model (97.10%). This indicated that due to the augmented data and the regularization effects it had on the model, the weights achieved were more generalized and were not over-fitted on the local validation set and therefore, the local validation accuracy score was more close to Kaggle test set accuracy score.

Furthermore, after training the VGG16 model with the augmented data, we observed an increase in the local validation and Kaggle data accuracy. The $VGG16_{aug}$ model (with one dense layer) achieved 98.57% on the Kaggle dataset. To improve our model, we also added different number of dense (fully-connected) layers to the end of the model and also trained the model with 2 different learning rates and compared their accuracies and losses together. As it can be seen in Table 2, we observed that the model with four dense layers has the highest score and lowest loss compared to the others. This model also achieved an accuracy of 98.933 % on the Kaggle dataset. Figure 3a and 3b depict the training loss and local validation set accuracy, respectively.

As a summary, Table 1 shows the performance of some of the approaches that were tried for the image classification task.

Table 2: Different VGG16 configurations used. The accuracy and loss of the final model used is bold.

Number (and Depth) of Dense Layers	Learning Rate	Accuracy	Loss
1 (10)	0.01	98.39%	0.1822
3 (512, 256, 10)	0.01	99.04%	0.0514
4 (512, 256, 128, 10)	0.01	**99.03%**	**0.0411**
4 (512, 256, 128, 10)	0.003	98.99%	0.0535

6 Discussion and Conclusion

In this project by examining different deep learning models and other helpful methods in image classification and evaluating their performances in the case of our specific problem, we were able to find the best performing models on the Modified-MNIST dataset. In this process there are many challenges such as how to set different attributes and hyper-parameters in the models and methods so they can perform properly in the classification task considering the nature of our problem and dataset.

For the future works, one can work on using *Spatial Transformers Networks* (STN) in the CNN architecture. Spatial Transformer modules can be included into a standard neural network architecture to provide spatial transformation capabilities. Here a spatial transformer that crops out and scale-normalizes the appropriate region can simplify the subsequent classification task, and lead to superior classification performance [5].

We were also inspired by this paper [1] to implement an ensemble model of VGG16 and VGG19 on this problem; however, due to time constraint we were not able to test this method.

7 Statement of Contributions

In this project all the group members contributed to both coding and write up of the report equally.

References

[1] Hammam Alshazly, Christoph Linse, Erhardt Barth, and Thomas Martinetz. Ensembles of deep learning models and transfer learning for ear recognition. *Sensors*, 19(19), 2019.

[2] Angelo Cardoso and Andreas Wichert. Handwritten digit recognition using biologically inspired features. *Neurocomputing*, 99:575–580, 2013.

[3] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2017.

[4] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[5] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks, 2015.

[6] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2016.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Xiao-Xiao Niu and Ching Y Suen. A novel hybrid cnn–svm classifier for recognizing handwritten digits. *Pattern Recognition*, 45(4):1318–1325, 2012.

[10] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.