Modified MNIST Classification of Largest Numerical Digit

Abby Leung
School of Computer Science
McGill University
Montreal, QC H3A 0G4

Jizhou Wang
School of Computer Science
McGill University
Montreal, QC H3A 0G4

Nahiyan Malik
School of Computer Science
McGill University
Montreal, QC H3A 0G4

Abstract

In this project, we analyzed the modified MNIST dataset and used deep learning models to identify handwritten digits contained in each image and find the digit associated with the highest numeric value. Specifically, we analyzed the performance of CNN architectures such as a basic CNN, VGG, ResNet, and EfficientNet, using different hyperparameters such as optimizers, loss functions, learning rates, different preprocessing techniques and augmentation methods. We find that overall, EfficientNet achieved higher accuracies and faster runtimes compared to all the other models.

1 Introduction

In this project, we are given a modified MNIST dataset, a set of images each containing three handwritten digits from the MNIST dataset on a patterned background. The images are in grey scale and represented by a matrix of pixel intensities. Each image is associated with a label, corresponding to the digit with the highest numerical value amongst the three handwritten digits of that image. We are tasked with using deep learning models to predict this label of handwritten digit with the highest numerical value from each image and output the result.

We have trained a simple 3 layer Convolutional Neural Network, VGG, ResNet and EfficientNet on this dataset. Most of the models were pretrained from ImageNet [3]. The best set of hyperparameters found were the Adabound optimizer with learning rate of 0.001 to 1, Cross Entropy loss, batch size of 50, and no preprocessing. A more detailed analysis of each model and hyperparameter selection can be found in the Results section below. We found that the above combination of hyperparameters with data augmentation trained on EfficientNet-B2 produced the highest overall accuracy of 98.76% after 92 epochs.

2 Related Work

Classifying the MNIST dataset is a widely studied problem in machine learning. There have been many previous work on classifying MNIST handwritting digits, including many that have achieved considerable accuracy. In 2012, Ciresan et al. [1] introduced the multi-column deep neural network (MCDNN), the first to achieve near-human performance on the MNIST dataset and outperform humans by a factor of two on the traffic sign recognition benchmark. The MCDNN achieved an error rate of 0.23 on the MNIST dataset with 35-net MCDNN.

More recently, Kowsari et al. [13] from the University of Virginia introduced Random Multimodel Deep Learning (RMDL) for classification. RMDL is an ensemble method that combines Deep Neural Networks (DNN), Convolutional Neural Netwroks (CNN), and Recurrent Neural Networks (RNN). It was able to achieve an error rate of 0.18 on the MNIST with 30 RDLs.

3 Dataset and Setup

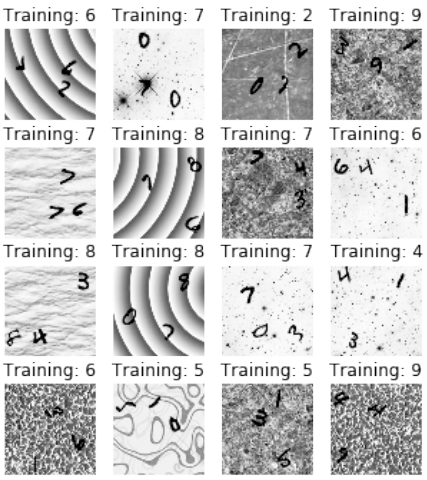

Figure 1: An example of the dataset with their respective training labels.

The dataset given to us were generated with three digits from the original MNIST dataset combined with a set of patterned backgrounds in grey scale. Each image is associated with a label, corresponding to the digit with the highest numerical value amongst the three handwritten digits of that image. Because of the set up of the problem, there is an imbalance in the dataset where the label 0 is the least likely to occur compared to the label 9, which is the most likely to occur as shown in Figure 2.

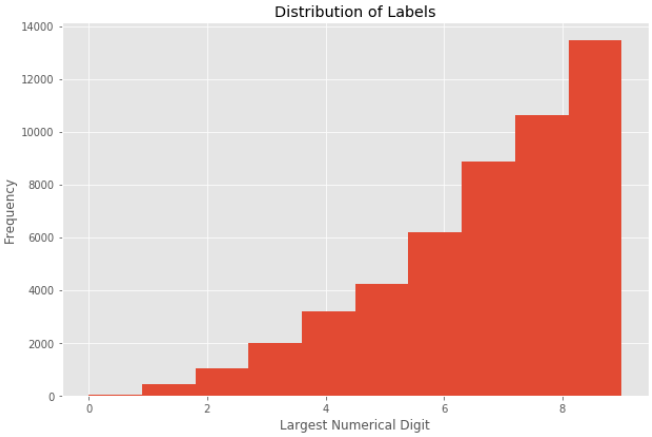

Figure 2: The distribution of all the tagged labels.

Therefore for training and validation, we have stratified our dataset into 90% training and 10% validation set out of 50000 images for a balanced training strategy.

Knowing the number of training data can be insufficient to for training, a data loader is setup such that data augmentation can be applied before the images are loaded into a neural network.

Standardization is applied to each individual image before given as input to a neural network for training and validation. Additionally, the images were modified using OpenCV to isolate the digits from the background. Both the raw and filtered images were used to train the models.

4 Proposed Approach

Given the time constraints and computing resources, we tried to incrementally narrow down the best set of hyperparameters by running controlled experiments while tuning each hyperparameter separately. Conducting cross validation or doing a brute force grid search would be infeasible given time period. We started out with a default set of hyperparameters; specifically, an Adam optimizer with a learning rate of 0.1, Cross Entropy loss, batch size 100, no data augmentation and a simple threshold preprocessing. Testing for the best set of hyperparameters was conducted in the following order: optimizer, model architecture, preprocessing, batch size, loss function, learning rate and data augmentation. For every incremental test, the best hyperparameter and model combination that was found in the previous tests is used as the default set of hyperparameters in the following tests.

4.1 Preprocessing

The original images include three digits and various background images. Several preprocessing steps were used to remove the backgrounds from the images. To start, a basic threshold was used to only display image intensities 254 to 255. This resulted in the digits being isolated, but a lot of the details below the 254 intensity were filtered out as well.

A more complex preprocessing was used to preserve such details. The images were thresholded from intensity region 225 to 255. Then, the three largest connected components were isolated, which indicated the three digits. Additionally, in order to recover some of the lost details, a slight dilation was applied to the connected components.

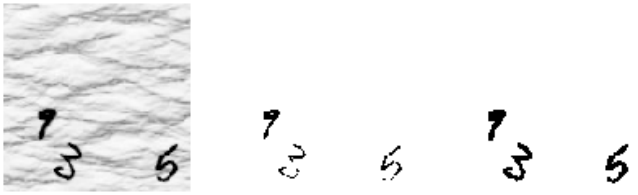

Figure 3: An example of the original (left), basic thresholding (middle) and the connected components (right). The target label is 9.

4.2 Neural Network Architectures

We have used a number of different models for our classification task, including a simple 3 layer CNN, VGG, ResNet and Efficientnet. Pretrained model parameters are always used in the available model architectures as they reduce the time needed for training. All the models use a technique called batch normalization [9] to accelerate the learning process.

4.2.1 Convolutional Neural Network

Convolutional neural networks, or CNN, are deep learning models that takes a 3D tensor as input and at each layer transforms this input tensor to an output tensor using differential equations. [10] The CNN architecture is composed of three types of layers: a convolutional layer, a pooling layer and a fully-connected layer. The convolutional layer parameters consist of a set of spatially small filters. We then slide the filters across the input column and produce a 2-dimensional activation map of the responses of that filter at each spatial position [10]. The activation maps are then stacked to produce the output volume. The pooling layer is used to reduce the size of the representation to reduce the number of parameters and computation in the network, thus controlling overfitting. A fully-connected layer has full pairwise connection between neurons of adjacent layers [11]. The full network is formed by alternating the three types of layers and trained with backpropagation.

Our simple CNN has 2 convolutional layers and 1 fully connected layer. It is abbreviated as CNN-3.

4.2.2 VGG

The VGG network is a deep convolutional network trained by Simonyan and Zisserman [17] at the Visual Geometry Group at Oxford University. It consists of 16–19 weight layers and achieved state-of-the-art results at the time of its release. The VGG models that were used used include VGG-11 and VGG-16, both with and without batch normalization.

4.2.3 ResNet

ResNet is a deep residual learning framework that fits stacked layers to residual mapping instead of the desired underlying mapping. [8] Let $H(x)$ be the desired underlying mapping. The stacked non linear layers are then fit to $F(x) := H(x) - x$. The original mapping is recast into $F(x) + x$. This model seeks to address the degradation of training accuracy problem in deep

networks since it is hypothesized that it is easier to optimize residual mapping than it is to optimize the original unreferenced mapping [8]. It is the first neural network to achieve superhuman results on ImageNet [6].

The ResNet models used include ResNet-18, ResNet-34 and ResNet-50. ResNeXt-50, a wider version of the ResNet-50 model, was also used.

4.2.4 EfficientNet

EfficientNet is the current State-of-the-Art neural network model for image classification on ImageNet [18]. EfficientNet, aptly named, is a very efficient neural network that's capable of achieving a much higher accuracy (76.3% to 82.6%) than ResNet with up to 21x fewer model parameters than existing CNNs.

A neural network model can be scaled in terms of its width (input/output channels), depth (number of layers) and resolution (image resolution). EfficientNet tries to address the problem of scaling within a neural network model by proposing a *compound scaling method*. In conventional practice, models are scaled arbitrarily. EfficientNet uniformly scales all 3 parameters with a fixed scaling coefficient which can be found through reinforcement learning.

The generation of scaled models from B0 to B7 come from Neural Architecture Search [19] since the effectiveness of model scaling depends heavily on the baseline network.

4.3 Optimization

We have compared three different optimizers including the Adam, stochastic gradient descent (SGD) and Adabound and tested them with three different model architectures specifically: CNN-3, VGG-16 and Resnet-18.

4.3.1 Adam

Adam is a first-order gradient-based optimizer for stochastic objective functions. It only requires the first-order gradients and little memory requirement. It takes the estimates of the first and second moments of the gradients and computes individual adaptive learning rates for different parameters. It works well with sparse gradients and in on-line and non-stationary settings. It is largely inspired by AdaGrad (Duchi et al., 2011), which works well with sparse gradients, and RMSProp (Tieleman & Hinton, 2012) [12].

4.3.2 Stochastic Gradient Descent

A standard gradient descent takes into account the cost and gradient over the full training set, approximates the expectation below and updates the parameters θ of the objective $J(\theta)$ as

$$\theta = \theta - \alpha \nabla \theta E[J(\theta)]$$

Stochastic Gradient Descent (SGD) however, instead of using the entire training set, it only computes the gradient using a single or a few training examples. The update rule then becomes,

$$\theta = \theta - \alpha \nabla \theta J(\theta; x^{(i)}, y^{(i)})$$

where $(x^{(i)}, y^{(i)})$ is an example from the training set.

4.3.3 Adabound

Adabound is a variant of the Adam optimizer. The lower and upper bound are initialized as zero and infinity and then smoothly converge to a constant final step size. It is an adaptive method at the beginning of training, and then gradually transforms to SGD, or with momentum, as time step increases. Adabound improves on the Adam optimizer so it does not suffer from the negative impacts of extreme learning rates. [15]

4.4 Loss functions

We have compared cross entropy loss and focal loss as our loss functions.

4.4.1 Cross Entropy Loss

The cross entropy loss function measures the performance of a model that has outputs of probability value 0 to 1.[4] It is the negative log-likelihood of the logistic function defined by [7]

$$Loss(D) = -\sum_{i=1}^{n} y_i ln(\sigma(W^T X_i)) + (1-y_i) ln(1-\sigma(W^T X_i))$$

where $\sigma(W^T X_i)$ is the probability that the output y_i=1 and $1-\sigma(W^T X_i)$ is the probability that the output y_i=0, $y \in \{0, 1\}$, n is the number of training examples.

4.4.2 Focal Loss

The focal loss function reshapes the standard cross entropy to better address the problem of class imbalance. It dynamically scales the cross entropy loss such that scaling factor can automatically down-weight the easy examples during training and rapidly focuses the model on hard examples. [14]

5 Results

Our best score on the Kaggle leaderboard was achieved with the following settings: EfficientNet-b2 model architecture, Adabound optimizer with learning rate of 0.001 to 1, no preprocessing, batch size of 50, cross entropy loss function and data augmentation including random affine transformations with 10% translation, 40° rotation and a scaling factor of 70% to 100%. It yielded a validation set accuracy of 98.76%.

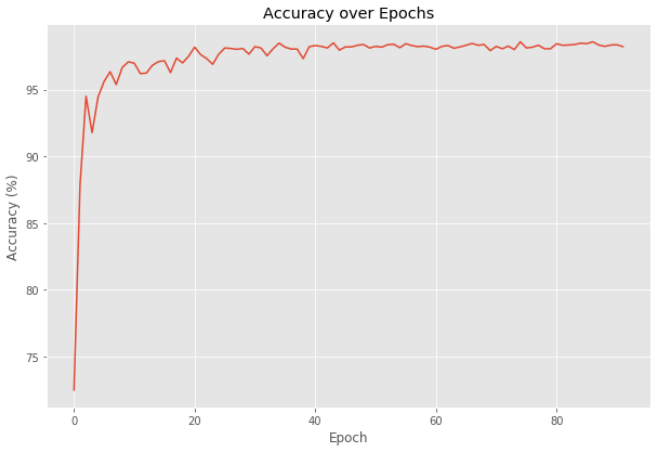

Figure 4: Accuracy of the model over 92 epochs.

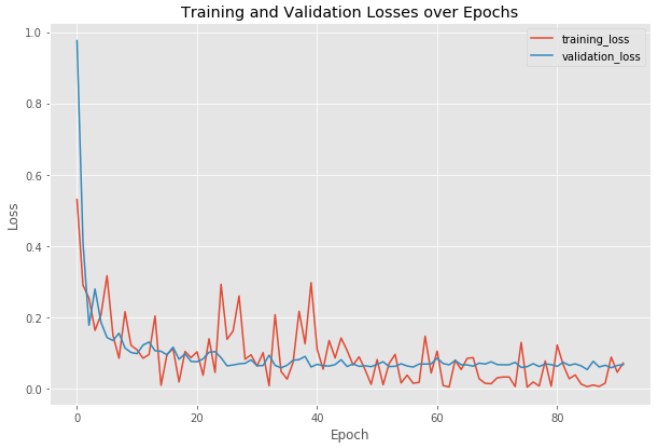

Figure 5: Training and validation losses of the model over 92 epochs.

A more detailed discussion on each hyperparameter selection can be found below.

5.1 Optimizers and Learning rates

Using a default learning rate of 0.1, we tested the Adam, SGD and Adabound optimizers. Using the CNN-3 and ResNet18 architectures, we found Adabound resulted in the best accuracy with 42.78% and 94.72% respectively for the CNN-3 and ResNet18 models. For all further experiments, Adabound with a final learning rate of 0.1 and a SGD learning rate of 0.001 was used (0.001, 0.1).

5.2 Model Architectures

Using the hyperparameters stated above, various architectures were tested, including CNN-3, VGG-11, VGG-16, ResNet-18, ResNet-34, ResNet-50, ResNeXt-50 and EfficientNet-b0. We acquired the best validation set accuracy from ResNet-34 (97.14%), ResNet-50 (97.34%), ResNeXt-50 (97.86%) and EfficientNet-b0 (96.26%) and therefore proceeded with them. All the models were also pretrained.

5.3 Loss Functions

Cross entropy loss and focal loss were tested against EfficientNet-b0. Cross entropy loss yielded a higher validation set accuracy with 96.26% compared to focal loss which resulted in 95.56%. Therefore, cross entropy loss was used for all further experiments.

5.4 Preprocessing

Preprocessing was then tested to determine the efficacy of removal of the background images, the hypothesis being that models would potentially function better with less noise in the form of the background. However, we found that the more involved the preprocessing, the lower the validation accuracy. All of the models exhibited the highest validation accuracy using the raw images, with the accuracy decreasing as the complexity of the proprocessing increased. Given enough layers, it appears the removal of the background is done more effectively by the model itself. For example, when tested against EfficientNet-b0, no preprocessing resulted in a validation accuracy of 96.26%, the basic thresholded preprocessing resulted in 94.64% and the more involved connected components resulted in 93.66%. Therefore, for all further experiments, the the raw images were used.

5.5 Batch Sizes

The default batch size used was 100 samples per iteration except ResNet-50 and ResNext-50 due to memory constraints. In 2017, Radiuk [16] investigated the effects of batch size on CNN based model accuracies. The MNIST dataset was one of the datasets tested against. The results showed that increasing the batch size could result in faster convergence and greater testing accuracies. As such, the batch sizes were increased according to the models and their memory constraints. ResNet-34 achieved a validation set accuracy of 97.76% using a batch size of 200. EfficientNet-b0, in contrast, had a validation set accuracy of 95.42% with a batch size of 200, 96.26% with a batch size of 100 and 96.79% with a batch size of 50.

Therefore, all EfficientNet models henceforth used a batch size of 50. ResNet-50 and ResNeXt-50, due to memory limitations of the compute machine, used a batch size of 50. ResNet-34 used a batch size of 200.

5.6 Data Augmentation

Following the previous experiments, the best residual network model, ResNeXt-50, and EfficientNet-b0 were chosen to apply data augmentation. Many different combinations of data augmentation parameters were used, with parameters yielding the best results being random affine transformations with 10% translation, 40° rotation and a scaling factor of 70% to 100%. While applying the data augmentation, it was also found that the learning rates for the models needed to be changed to work best with the augmentation settings. Specifically, for ResNeXt-50 the learning rate was set to (0.001, 0.01) and for EfficientNet-b0, it was set to (0.001, 1). Using the stated hyperparameters along with data augmentation, ResNeXt-50 achieved a validation set accuracy of 97.98% and EfficientNet-b0 achieved an accuracy of 98.24%.

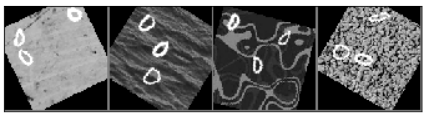

Figure 6: An example of data augmentation using the parameters stated.

Building onto the EfficientNet-b0 model, EfficientNet-b1 and EfficientNet-b2 were also trained and tested with the respective validation set accuracies being 98.56%, 98.76%.

5.7 Final Results

Based on the experiments stated above, all hyperparameters were chosen in a controlled manner. Each model required fine-tuning hyperparameters separately for the best results. The final results can be found in Table 1.

Model	Accuracy	Epochs	Runtime/Epoch (sec)
ResNet-34	97.76%	8	238
ResNet-50	97.34%	13	400
ResNeXt-50	97.98%	14	821
EfficientNet-b0	98.24%	48	102
EfficientNet-b1	98.56%	50	149
EfficientNet-b2	98.76%	92	218

Table 1: Final results - EfficientNet-b2 yielded the best validation set accuracy with 98.76%.

6 Discussion and Conclusion

In general, we found that the EfficientNet model architecture gives the best performance along side its fine-tuned hyperparameters. Given more time and resources, an exhaustive grid search could be conducted for all the hyperparameters. For the given problem we may be able to generate synthetic datasets using GANs [5] that could reduce the biases in prediction of the given dataset as there may not be enough information to represent all the possible combinations of three digits with different backgrounds. This could create a more general model that detects the largest digit given any number of digits on any kind of backgrounds.

With more computing resources, cross validation could help in better fine-tuning each model. For this project, we used a held out validation dataset to make changes. Similarly, ensembling various models could also help in increasing overall accuracy.

We could also explore an automated method for generating data augmentation [2] instead of fine-tuning them by hand. Additionally, transfer learning could be used from other types of MNIST datasets.

7 Statement of Contributions

Jizhou worked on the implementation of testing pipeline and researched on the different models and hyperparameters that could improve our testing performance.

Nahiyan worked on image preprocessing as well as testing models and hyperparameters to maximize accuracy.

Abby worked on VGG implementation and the report.

References

[1] D. Ciregan, U. Meier, and J. Schmidhuber. "Multi-column deep neural networks for image classification". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. June 2012, pp. 3642–3649. DOI: 10.1109/CVPR.2012.6248110.

[2] Ekin D. Cubuk et al. *AutoAugment: Learning Augmentation Policies from Data*. 2018. arXiv: 1805.09501 [cs.CV].

[3] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[4] Brendan Fortuner. *Loss Functions¶*. URL: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

[5] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[6] William L. Hamilton. *COMP 551 - Applied Machine Learning Lecture 15 — Convolutional Neural Nets*. URL: https://cs.mcgill.ca/~wlh/comp551/slides/15-conv_nets.pdf. Last visited on 2019/11/06. 2019.

[7] William L. Hamilton. *COMP 551 - Applied Machine Learning Lecture 4 — Linear Classification*. URL: https://cs.mcgill.ca/~wlh/comp551/slides/04-linear_classification.pdf. Last visited on 2019/11/12. 2019.

[8] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[9] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[10] Andrej Karpathy. *CS231n: Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/convolutional-networks/. Last visited on 2019/11/06. 2019.

[11] Andrej Karpathy. *CS231n: Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/neural-networks-1/. Last visited on 2019/11/06. 2019.

[12] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[13] Kamran Kowsari et al. "RMDL: Random Multimodel Deep Learning for Classification". In: *CoRR* abs/1805.01890 (2018). arXiv: 1805.01890. URL: http://arxiv.org/abs/1805.01890.

[14] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *CoRR* abs/1708.02002 (2017). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002.

[15] Liangchen Luo, Yuanhao Xiong, and Yan Liu. "Adaptive Gradient Methods with Dynamic Bound of Learning Rate". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bkg3g2R9FX.

[16] Pavlo M. Radiuk. "Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets". In: *Information Technology and Management Science* (2017).

[17] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[18] Mingxing Tan and Quoc V Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *arXiv preprint arXiv:1905.11946* (2019).

[19] Barret Zoph and Quoc V. Le. *Neural Architecture Search with Reinforcement Learning*. 2016. arXiv: 1611.01578 [cs.LG].