Miniproject 3: Analysis of prediction on classifying the largest digit on a modified MNIST

Design and validate a supervised classification model to perform the Modified MNIST prediction task

Shiyun Luo
McGill ID:260778958
shiyun.luo@mail.mcgill.ca

Jiyi Wang
McGill ID:260771384
jiyi.wang@mail.mcgill.ca

Yifei Zhao
McGill ID:260900278
yifei.zhao@mail.mcgill.ca

Abstract
This paper presents the methodology and results obtained when using supervised learning methods to investigate the performance of a modified MNIST dataset. This study used a modified dataset of 50,000 data instances for training, and 10,000 instances as test set. Each training image consists of three digits of random sizes and orientations, which is represented as a matrix of pixel intensity values. The goal is to output the digit in the image with the highest numerical value. Among all supervised learning models based on CNN's variation, an ensemble model with VGG architecture has the best performance with 98.40% prediction accuracy in held-out validation and 98.23% prediction accuracy in Kaggle Competition.

1 Introduction

1.1 Motivation

In recent years, deep learning-based techniques have been gaining significant interest in the research community for solving various problems. One of the most widely used techniques are Convolutional Neural Networks(CNN), which is able to retrieve features from multidimensional inputs, turn them into a very interesting alternative for solving problems within the field of computer vision[1]. In particular, neural networks play an important role in complex image classification tasks. In this paper, we represent several approaches for predicting an associated label in each image relative to its numerical value.

1.2 Methodology

In this project, we implement and compare the performances of different models based on CNN. To predict the largest numerical value of each image on the modified MNIST dataset, we implement various architecture based on the AlexNet, LeNet and VGG models inside the Keras deep-learning library. Accuracy is also improved by changing the hyper-parameters and using the ensemble method. Through the training process, we apply the validation pipeline on dataset to examine and compare its performance in predicting unseen data comments. We then use the predicted accuracy to measure the quality of the models to make the most appropriate prediction.

1.3 Important Findings

VGG architecture obtains better performance than LeNet5 and AlexNet corresponding to our experiments. This phenomenon provides us the motivation to implement datasets on different architecture of VGG models based on its constant kernel parameter. According to our results, VGG8 is the best performer in both the validation and Kaggle test sets, whereas, with ensemble models, the one contains VGG8, VGG9 and VGG11 scored highest in Kaggle test sets.
Different batch size and optimizer also influence the results we obtained through the process of adjusting parameters. With the increasing batch size, the running time of the model and the accuracy will have a slight improvement. For the perspective of optimizer, Adam achieves generally the best performance among others. Batch normalization, dropout layer and ensemble method are important factors that improve the accuracy as well.

2 Related Work

Previous similar work using MNIST dataset based on the image classification task can be found. The current state of accuracy generated by Convolutional Neural Network model on the **original** MNIST dataset is achieved generally high. This gives us the motivation to build and test CNNs on the **modified** MNIST dataset for this work.

• **[Karen Simonyan and Andrew Zisserman, 2015]**([2]) investigated the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. It presented **VGGNet** where a CNN model now has the fifth highest accuracy of ImageNet, despite its architecture is relatively simple and uniform. This implies that CNN play an important role in VGG's appeal, which is what prompted us to base our implementation on the VGG architecture.

• **[David Opitz and Richard Maclin, 1999]**([3]) presented an empirical study based on popular **ensemble methods**: Bagging(Breiman, 1996c) and Boosting(Freund Schapire,1996; Schapire, 1990). Methods were tested on 23 datasets using both neural network and decision trees as our classification algorithm. The results demonstrate that a Bagging ensemble method nearly always outperforms a single classier, and it is probably appropriate for most problems. Therefore, this gives us the motivation to create our best ensemble model which was built from multiple instances of our best standalone models.

3 Dataset and Setup

3.1 Dataset Description

The MNIST (Mixed National Institute of Standards and Technology) database was introduced by LeCun et al. in 1998[4]. The modified dataset used in this analysis contains 50000 instances, and each one represents an image with 3 digits and noise. The goal of our study is to identify the biggest digit in each image. Each image is normalized and centered in a gray-level image with 128×128 matrix of pixel intensity values.

3.2 Data Pre-processing

We load the dataset as an object using ReadPickle function.
We then transform the loaded data type of all features to float. We also do normalization by dividing 255 in order to re-scale all features to the range of 0 to 1.
Since our image inputs are grey-scale instead of RGB(red, green and blue). Therefore, we reshape the x set by inserting a new dimension which represents the channel of color.
Besides that, we use one-hot encode to fit data type better for training as well.
For the final step, pipeline validation is used to create training and validation sets where they are done randomly for each model, with an 80:20 train to validation ratio.

3.3 Data Augmentation

To increase the generalizability of the model, we implement data augmentation to learn more robust features, and generalize example data points which are not included in the training set. Through the implementation of Keras's ImageDataGenerator function, we obtain augmented data from the original images by applying simple geometric transforms: rotation, width shift, height shift, horizontal and vertical flip.

3.4 Task Setup

Since the neural network is a cyclic training process, then we can use it to get an output in each epoch based on 80% training data, and 20% validation data can also be used to obtain the validation accuracy. Through the back propagation, we can infer the loss function. We then minimize this function by using the gradient descent in order to update the network weight and bias for entering a new round of epoch. We eventually obtain the model with the highest validation accuracy after several epochs.

4 Proposed Approach

The following are the proposed approaches and some general discussions about setups and algorithm implementations from building and complementing the supervised model on this particular dataset and considering the prediction performances as well as computational efficiency. Readers should also keep in mind that we may not always follow the rules in exploring the optimal model due to the computational limitation in terms of time for this machine.
Before starting to build and train the model, we studied and referred to several classic CNN models, and finally chose AlexNet, LeNet and VGG as the initial directions. By using Keras as the automatic differentiation framework. The model type that we use is Sequential. It allows us to build a model layer by layer. To fit our dataset, we continuously refine each model under the standard architecture, and finally get some suitable models and compare their performance.

4.1 Model Description

Convolutional Neural Network(CNN, or ConvNet): This is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other[5]. The preprocessing required in a ConvNet is much lower as compared to other classification algorithms as it has the ability to learn these filters/characteristics with enough training. The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

• **Convolutional Layer**: It is the first layer to extract features from an input image. Convolution is a mathematical operation that preserve the relationship between pixels by learning image features using small squares of input data.

• **Strides**: It is the number of pixels shifts over the input matrix.

• **Non Linearity(ReLU)**: The output of ReLU is $f(x) = max(0, x)$. It introduces non-linearity in our ConvNet as the dataset want ConvNet to learn non-negative linear values.

• **Pooling Layer**: This layer reduces the number of parameters of parameters when the images are too large. Especially for the max pooling layer, it takes the largest element from the rectified feature map.

• **Fully Connected Layer**: This layer we call as FC layer, we flatten our matrix into vector and feed it into a fully connected layer like neural network. It purposes to connect every neuron in one layer to every neuron in another layer.

• **Batch Normalization**: This is a method we can use to normalize the inputs of each layer, in order to fight the internal covariate shift problem.

• **Dropout Layer**: It is a technique to prevent a model from overfitting by ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random.

4.2 Pipeline Validation

We use the held-out method to split the dataset at the ratio of 8:2. We use 80% training data to train the model, and 20% validation data to predict the features and make further improvement.

For LeNet5 and AlexNet, we use their standard architecture to train our model. Their architecture is shown in the Table 1.

Table 1. Standard architecture of VGG, LeNet5 and AlexNet

Model	Conv Layer	Pooling Layer	FullC Layer
VGG16	13	5	3
LeNet5	2	2	3
AlexNet	5	3	3

Conv Layer stands for "Convolutional Layer"

FullC Layer stands for "Fully Connected Layer"

4.3 LeNet5

In LeNet model, its convolutional layers have 5×5 kernel size and stride 2, then use max pooling layers to reduce the size of inputs.

4.4 AlexNet

For AlexNet, its kernel size decreases from 11×11 to 5×5 for next two convolutional layers, and 3×3 kernel size is used for the last three layers. Also, each convolutional layer has different strides, which is 4,1,1,1,1 respectively.

Generally, the performances of LeNet5 and AlexNet are worse than that of VGG architecture. Thus, we will focus more on VGG in our project.

4.5 VGG

We worked on the standard VGG16 architecture at the beginning. However, since our modified MNIST dataset has only a single channel of color, the size mismatch occurs. Therefore, we reduced the number of layers to 11 which contains 8 convolutional layers and 3 fully connected layers to make our dataset fit in VGG and reduce the training time and computational complexity. Four max-pooling layers are also used to downsampled or pooled feature maps that highlight the most present feature in the patch. We also halve the number of kernels in all layers relative to that number in all layers of standard architecture. We follow this concept to test VGG10 to VGG6 in sequence, these allow us to explore the effect on the performance of diversity of layers (see Table 2 for the specific architecture). It turns out that the eight-layer VGG has the highest accuracy in the validation set.

In terms of hyper-parameter setting, we keep the core parameter setting of the VGG model unchanged (i.e. We keep the kernel size be 3×3, stride be 1 and max-pooling kernel be 2×2). On this basis, we select the ReLU nonlinear function as the activation function of all layers except for the final fully connected layer. This is based on the fact that ReLU's gradient is non-saturation, which greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid/tanh functions provided. Meanwhile, the ReLU function tends to make part of the neuron output to be 0, which causes the sparse nature of the network. The interdependence of parameters is reduced and the overfitting problem is alleviated as well.
For the output layer, the number of neurons is changed to 10, and the probability is output through softmax function to achieve the final classification. It also shows that inserting the batch normalization layer and drop-out layer can improve the results to some extent. Based on the situation for each structure that 100 epochs are carried out, we set an early stop in all architecture. This method would terminate the training in advance if 10 epochs are implemented while the results are not improved. This significantly reduced the running time for training data.

In the backpropagation, we try multiple optimizers on loss functions, and Adam method does the best job. We also try to find out the impact of batch size on accuracy, and the results will be elaborated in the next section. The annealer method is also used when there is no improvement after 3 epochs, the learning rate will decrease to 0.85 times of current learning rate and will accelerate its convergence.

4.6 Ensemble Method

In our study, we also use ensemble learning to improve the performance. Ensemble construct multi models(i.e. In our case, it represents models of multi layers in VGG architecture) to fit and predict. In order to obtain

[1] Cov Layer stands for "Convolutional Layer"
[1] MaxP Layer stands for "Max Pooling Layer"
[1] FullC Layer stands for "Fully Connected Layer"
[1] BatchN Layer stands for "Batch Normalization Layer"
[1] DropO Layer stands for "Dropout Layer"

Table 2. Different architectures of VGG [1]

Model	Cov Layer	MaxP Layer	FullC Layer	BatchN Layer	DropO Layer
VGG6	4	3	2	4	1
VGG7	5	3	2	5	1
VGG8	6	3	2	6	1
VGG9	7	3	2	7	1
VGG9_2	6	3	3	6	1
VGG10	8	4	4	8	2
VGG11	8	4	3	8	2

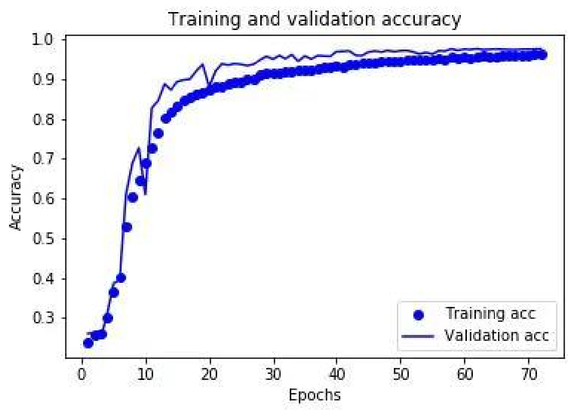


Fig. 1. VGG8 training and validation accuracy

higher accuracy in this method, we need select several models with high accuracy and diversity between them. They can then combine to form a better model.
By experimenting VGG models of each different layer on the training dataset, we compare the prediction accuracy generated and conclude the best three models with different neural network architecture: VGG8(8 layer VGG variant), VGG9, VGG11. As parameters are randomized when running the models each time, we then implement these three models three to four times approximately, and eventually summarize the best seven to eight models to combine them as an ensemble model. This model is able to achieve the highest validation accuracy of 98.4%.

5 Results

5.1 Model Performance

Regarding on Table 3, our best performing single model is VGG8 with batch normalization and dropout after training for 73 epochs. The plot of the validation and training accuracy can be found in Figure 1. We also build and implement another CNN model based on standard AlexNet. After training for 87 epochs, we achieve 95.8% accuracy on 20% validation set and 95.12% on test set. Some small modifications are made on the standard LeNet5 model as well where we achieve 85.58% accuracy on validation set and 85.21% on test set after training for 51 epochs.

When we start encountering difficulties with improving this single model, we decide to attempt an ensemble of a few examples of this model in order to see whether we can improve our accuracy. Thus, we train our best three models with different VGG layers(VGG8, VGG9 and VGG11) on the same dataset. Then, this ensemble model gives us the highest prediction accuracy 98.4% in validation set and 98.23% in Kaggle Competition, where we use 80% dataset to test the model on Kaggle. These results are shown in Table 4.

Table 3. Results for tested single models[2]

Model	training_acc	val_acc	runtime/epo	batch size	epochs
VGG6	0.9388	0.9650	49s	64	75
VGG7	0.9434	0.9692	79s	64	100
VGG8	0.9436	0.9760	82s	64	73
VGG9	0.9412	0.9743	85s	64	46
VGG9_2	0.9431	0.9697	46s	64	67
VGG10	0.8998	0.9041	81s	64	23
VGG11	0.9508	0.9722	46s	64	56
LeNet5	0.9580	0.9510	67s	64	87
AlexNet	0.8558	0.8521	48s	64	51

Table 4. Results for tested ensemble models[3]

Model	training_acc	val_acc	test_acc
Ensemble8	-	0.9840	0.9823
Ensemble5	-	0.9831	0.9790
Ensemble3	-	0.9813	0.9767

5.2 Batch Size

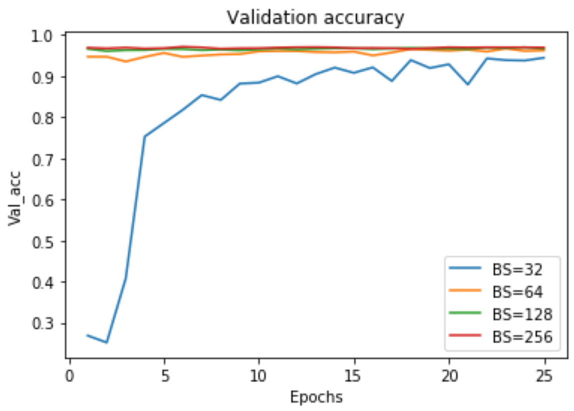
Although Batch Gradient Descent converges directly to minimum, SGD(Stochastic Gradient Descent) converges faster for larger dataset. However, SGD only considers one example at a time to take a single step, and we cannot implement the vectorized implementation on it as well which lead to slowness of the computations. To tackle this problem, a mixture of Batch Gradient Descent and SGD is applied. Therefore, we consider mini-batch gradient as main method on our large-scale dataset to minimize the loss function. Under this basis, we vary the batch size and train over 25 epochs. According to our results, along with the increase of batch size, the more accurate the descending direction is, the smaller the training fluctuation will be. When the batch

[2]training_acc stands for "training accuracy"
[2]val_acc stands for "validation accuracy"
[2]runtime/epo stands for "runtime per one epoch"
[3]test_acc stands for "test accuracy"

size increases to a certain extent, however, its determined decline direction will hardly change. Our figures illustrate these properties clearly.

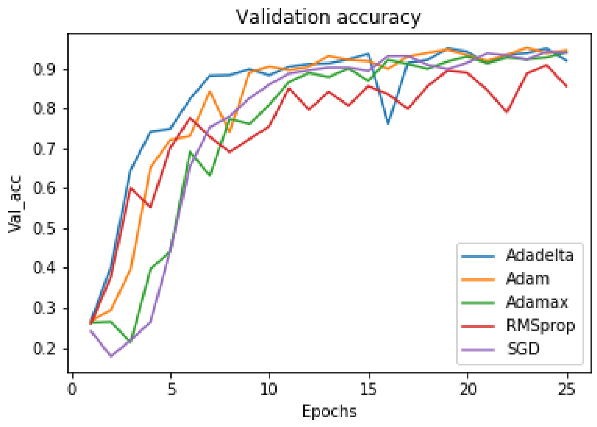
Table 5. Batch Size

Batch Size	Accuracy	Running time/epoch
32	0.9445	49s
64	0.9623	45s
128	0.9667	43s
256	0.9693	42s



5.3 Optimizer

Optimizer is one of the two arguments required for compiling a Keras model. To find the best optimizer suitable for this task, we explored the effects of the implementation of 5 different optimizers on the performance, the results shows that Adam is the best optimizer in terms of the validation accuracy and speed of convergence. AdaDelta converges faster in the first 10 epochs, while Adam, Adamax and SGD reach to the highest validation accuracy. Indeed, Adam combines the best properties of both AdaGrad and RMSProp to provide an optimization algorithm that can handle sparse gradients on noisy problems.



6 Discussions and Conclusion

6.1 Key Takeaways from this Project

In this project, we explore the application of Convolutional Neural Networks towards a multi-class classification problem based on the modified MNIST dataset. We have researched previous state of art models on various image datasets, and finally decide to implement models based on the standard architecture of VGG16.
We find out that it is the best model for its layers to handle feature extraction. We simply normalize our training data before feeding into the network. Kera's data augmentation feature helped a lot in increasing our model's accuracy given our relatively limited training data of only 50,000 images.
We also discover that using techniques such as batch normalization and dropout could help improve our model's ability to generalize other data and to combat overfitting.

Since our dataset contain images that are only 128×128 of grey-scale channel, we have to cut down the original VGG16 model. We experiment with various numbers of convolutional and hidden layers, and find that 3 block of convolutions with max-pooling and batch normalization, followed by one hidden layer with dropout before our softmax layer gives us the best performance. We also observe that the choice of optimizer and batch size can effect not only speed of training but also the model's final accuracy.

6.2 Further Improvements

As for future investigation, we can evaluate the performances of different architectures of VGG. Also, we can explore more other deep learning models such as SSD (Single Shot Detector) and YOLO(You Only Look Once). We can also consider and implement some non-deep learning models such as SVM(Support Vector Machine) and MultinomialNB(Multinomial Naive Bayes) to find the best model for our prediction of classification image by comparing their accuracy.

7 Statements of Contributions

- **Y.Zhao**: Model design, training, validation and testing, write-up contribution.
- **J.Wang**: Model design, write-up contribution
- **S.Luo**: Model design, write-up contribution

References

[1] Rikiya Yamashita, "Convolutional neural networks: an overview and application in radiology". In:(2018). URL: https://link.springer.com/article/10.1007/s13244-018-0639-9

[2] Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/ 1409.1556.

[3] Richard Maclin and David W. Opitz, "Popular Ensemble Methods: An Empirical Study". In: *CoRR* abs/1106.0257 (2011). arXiv: 1106.0257. URL: http://arxiv.org/abs/1106.0257.

[4] Alejandro Baldominos, Yago Saez and Pedro Isasi, *A Survey of Handwritten Character Recognition with MNIST and EMNIST*, July.2019

[5] Sumit Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way". In:(2018). URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[6] F. Pedregosa et al, "Scikit-learn: Machine Learning in Python ". In: Journal of Machine Learning Research 12 (2011), pp. 28252830.

[7] Nitish Srivastava et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: Journal of Machine Learning Research 15 (2014). URL: http://www.jmlr.org/papers/volume15/ srivastava14a/srivastava14a.pdf.

[8] Li Wan et al, "Regularization of Neural Networks using DropConnect". In: Proceedings of the 30th International Conference on Machine Learning. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 10581066. URL: http://proceedings.mlr.press/v28/wan13.html.