MiniProject3 – Modified MNIST Challenge

Xiaohui Wang, Yun-Fei Cheng, and Zijun Yu

Abstract - The 3rd mini-project required us to compete in Kaggle once again. The task involved discerning the largest hand-written digit out of 3 digits in a single 128 $\times$ 128 image with random background patterns-hence the "modified MNIST challenge." Our attempt involved 2 approaches after preprocessing data. The first approach consisted of trying different simpler models we have seen previously, such as logistic regression and k-nearest-neighbor. In most cases, the models either did not converge at all, or was not able to perform at any substantial level of accuracy. The second approach consisted of building Convolutional Neural Network (CNN) models of varying complexities. The optimal CNN model was one based on Xception, which produced 97.633% accuracy on Kaggle.

I. INTRODUCTION

IN this miniproject, we developed classifiers that aim to correctly label the largest digit in a 128 x 128 sized image. The complexity of the task was increased by having 3 different hand-written digits from the MNIST dataset in one single image. On top of this, unpredictable background patterns/drawings served as extra noise to our data.

The labels are not balanced, and this is to be expected since we are labelling the largest of the 3 digits as target. Naturally, any image with 9 as one of the three digits would automatically "dominate" whatever other digits are existent and have 9 as the target output. Similarly, 8 would be the target if the same image does not include any 9's, and so on. Following this argument, the target output would be 0 only if all 3 digits within an image are '0's. 42 such instances were found (See Appendix Table III & Fig. 7).

For data preprocessing, three main transformations were experimented. The first involved OpenCV's Canny algorithm[1] for edge detection. The second transformation involved normalizing the data and setting a threshold to decide whether a pixel is kept or omitted. In the last experimentation on data preprocessing, we attempted to draw "boxes" around each digit, essentially extracting 3 individual digits out of each image.

Following data preprocessing, simple machine learning models were tested on the flattened data. Logistic regression, support vector machines, and k-nearest neighbor were tested. However, all three did not perform at any useful level of accuracy. Minor parameter-tuning was performed along with 3-fold cross validation, but it did not improve the model, so the three basic models were forfeited.

Next, a basic Convolutional Neural Network model was built(see Appendix $Fig.8$). Following its success at 94.36% accuracy, different pre-trained models were considered as bases to build upon. Specifically, VGG16[2], ResNetV2[3], Xception[4], NASNet[5] were used as base models for further extension. All 4 were trained on the SGD optimizer, and the model using Xception had the best performance and produced a 97.633% accuracy on Kaggle.

II. RELATED WORK

After the success of basic CNN, an attempt to increase the accuracy further was made. Our aim was to integrate pre-trained models that could be accessed through deep learning library, and add a few layers following the output of the pre-trained models. Essentially, we let the pre-trained models handle the minor details such as edge detection and lower-level feature learning, and then take over the control for high-level abstraction as well as decision making. Intuitively, the extra layers we added correspond to two sub-tasks a human would have to perform to achieve the main task of discerning which of 3 digits is the largest. The first is recognizing 3 separate digits in one image. The second is picking the largest of the 3 recognized digits.

Pre-trained CNN models that are easily accessible from Keras's application library[6] were chosen: VGG16[2], ResNetV2[3], Xception[4], and NASNet[5]. These are all models for image classification with weights pre-trained on the ImageNet[7] dataset.

The MNIST database is a labeled dataset for handwritten digits. Often coined as the introductory dataset for machine learning, it is common to achieve accuracies of 95% and above. The state of art is "Regularization of Neural Networks using DropConnect" and performs near 99.7% [8].

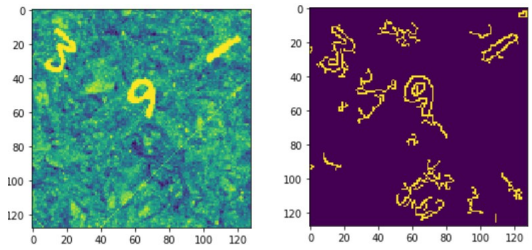
ImageNet is a dataset consisting of more than 14 million hand-annotated images and 20,000 categories. An image is considered correctly classified if any 1 of top 5 guesses by a machine matches the annotation. Every year, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is held where a subset of 1000 non-over lapping classes are chosen, and the state of art is "Self-training with Noisy Student improves ImageNet improves ImageNet classification" and has has 98.2% accuracy[9].

III. DATASET AND SETUP

The train set contains 50,000 samples of 128 x 128 images. The target outputs are unbalanced since larger numbers tend to be the label of an image. Due to the nature of approach_1, where we use simple models to predict the output, extensive feature engineering/data cleaning was required as opposed to approach_2, where we expect the CNN to do all the work for

us.

To prepare our data for approach-1, we took 3 steps. The first step is using the canny algorithm to detect edges. The benefit of canny is that some noise within data is suppressed at the same time. Through empirical testing, the hyper-parameters that perform best on the canny was determined to be (240,700). A sample of the data after the canny transformation is shown below.



(a) Before Canny (b) After Canny

Fig. 1: A figure from training set

We then proceeded to perform the 2nd step, where we further reduced the noise by normalizing the data and determining which pixels to keep or omit according to some threshold. At this point, we were worried about instances of image where edges are connected. For example, two 0's connected together may wrongly be interpreted as an 8. To prevent this problem, we drew boxes around each of the 3 digits so that each image is guaranteed 3 different instances of digits.

The data processing for CNNs are rather straight forward. For a simple CNN(with a structure in appendix), we can just input the raw data. For a pre-trained model from Keras[6], we need to transfer the input data to shape $(n, 128, 128, 3)$ where 3 denoted the image is in RBG setting. To do this we can simply repeat the gray scale pixel three times to fit the input.

Gaussian Filter is a blurring technique to smooth images so that noise and details can be removed. The input images are convolved with a Gaussian kernel. The Conv2D method is imported from the Keras package and we applied the Gaussian smoothing with kernels of size 3×3.

Flipping Images

To address the predefined issue of having very few '0' and '1' labels, we flipped images where the target is either 0 or 1 and added the flipped images into the training set. This was possible because 0 and 1 are both symmetric. Despite the symmetry, 8 was not flipped because flipping an image with 8 as one of the targets will flip other non-symmetric values.

IV. Proposed approach

Add threshold and then detect bounding to cut digits out

To reduce the noise and test the potential of this dataset, we decided to try bounding out the digits within the image. We used OpenCV[10] to implement this task. First, we turned the image into binary image with a threshold of 252, then used the contour function in cv2 to detect the boundaries. To fight noise, we only picked the first three largest contours. Then we extracted the contours from the raw image and transfer it to a size of 28×28 to fit a standard MNIST dataset.

Unfortunately, this method did not scale as it was not applicable to all images. The method failed when digits were too close or when the background noise was too large. A successful instance of digits extraction is shown in $Fig.2$ and $Fig.3$

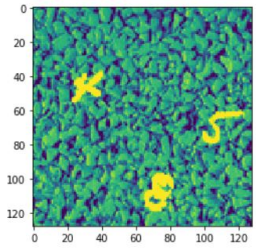
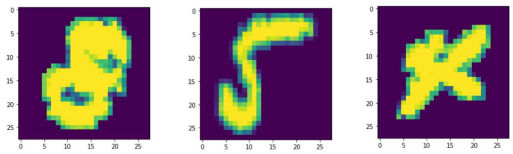


Fig. 2: Raw Image



(a) Digit1 (b) Digit2 (c) Digit3

Fig. 3: The bounded images

We now focus on describing approach_2 since this is where significant success was observed in performing the main task.

A. VGG16

VGG16 is a convolutional neural network model that uses multiple 3×3 kernel-sized filters one after another. The VGG16 network consists of 16 layers where their weights and bias parameters are learnt. Of the 16 layers, 13 are convolutional layers and 3 are dense layers.

B. ResNet

ResNet applies 'identity shortcut connection' that skips one or more layers. By copying the activations from shallower layers and setting additional layers to identity mapping, we can assume that the deeper layers are learning new useful underlying patterns from training data.

C. Xception

Xception is a deep learning method with depth wise separable convolution. As an extension of the inception architecture, Xception replaces the standard inception

modules with depth wise separable convolution.

In each of the above, a 2D-pooling layer was added to the output of the pre-trained models. Then, 4 pairs of dense-and-dropout layers are added. In total, 9 layers were added on top of the pre-trained models.

Optimizer

The usage of optimizers in the training process is of vital importance. Optimizers and their parameters can hugely affect the training time, convergence rate and whether the model can escape a local minima or a saddle point. Among the optimizers provided in Keras[6], we tried SGD, Adam, Adadelta, Nadam. We found that although these adaptive methods can provide a short learning time, they cannot assure the best performance. With a careful parameter control however, the SGD optimizer can give us a good result.

V. Results

A. Linear classifiers

Linear classifiers such as SVM and logistic Regression failed to produce desired results efficiently due to their simple architecture.

We experimented multinomial logistic regression with solver 'lbfgs' and max_iter=4000 with the original training data, and the model failed to converge. We experimented SVM with C=1 and C=2 with radial basis function (RBF) kernel, and the models took very long time to converge. We will put the emphasis on exploring CNN architecture.

B. ResNet

In this experiment, we used ResNet-50, which is often used as a starting point for transfer learning.

With validation_split=0.01 and batch_size=32, the following results were observed with increasing number of epochs. With increasing number of epochs, the validation accuracy increases.

TABLE I: Validation accuracy vs epochs

Number of epochs	Validation accuracy
6	0.9480
8	0.9520
12	0.9740
16	0.9908

$Fig.4$ is a visualization of validation loss and validation accuracy learning curves derived from epoch = 16.

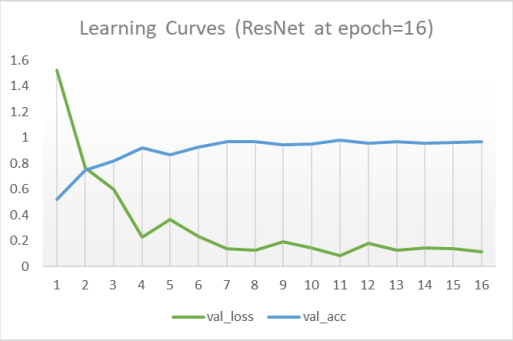


Fig. 4: ResNet learning curves

C. Xception

In this experiment, we used an Xception model.
We experimented on the model without and with a customized optimizer. We tested two different parameter settings in SGD optimizer. The first one is using a learning rate and a decay factor related to iterations. The second one is with a initial learning rate which will decay over epochs, and a nesterov momentum. (See $Fig.6$)

The self-defined optimizer we applied is SGD with lr = 0.1 and momentum = 0.8. With the improved optimizer, we were able to reach a slightly higher accuracy than the one without. Meanwhile, the model with the customized optimizer stabilizes earlier than the one without.

TABLE II: Xception validation accuracies

	Validation accuracy
default optimizer	0.9900
customized optimizer	0.9920

$Fig.5$ is a visualization of validation accuracy learning curves with and without the customized SGD optimizer.

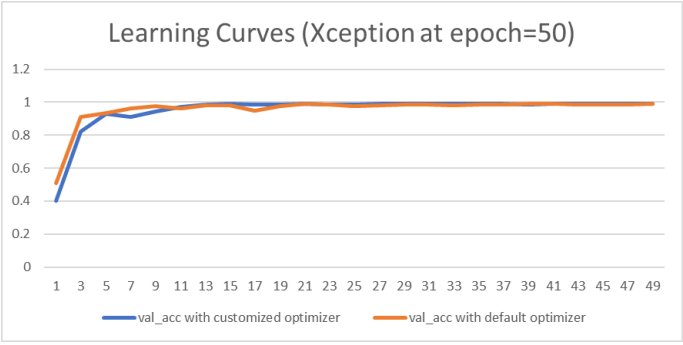
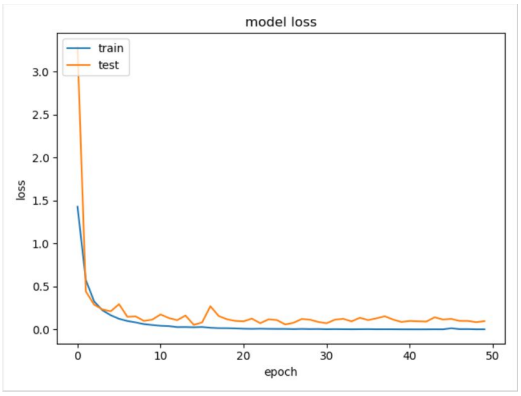
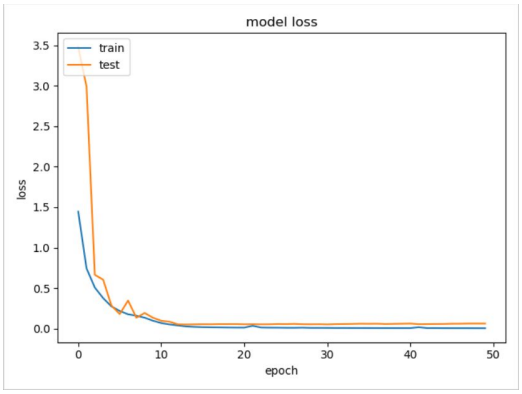


Fig. 5: ResNet learning curves

We can see from $Fig.6$ that the model can reach a better state with learning rate decay over time.



(a) Decay over time



(b) Decay over epoch PLUS momentum

Fig. 6: Two SGD settings

VI. Discussion and Conclusion

In this project, both simple models and more complex CNN models were built to classify the value-wise largest digit of an image. Success was observed in the CNN models as expected. This was true especially when the number of layers were increased. The best-performing Xception-based model has over 100 layers spanning over 14 blocks, and 40,007,986 parameters. It demanded approximately 4.5 hours running on one NVIDIA GeForce GTX 1070 GPU.

Slight overfitting is also observed in the CNN models. In less than 10 epochs, the train loss becomes less than the validation loss. However, an important and interesting observation to note is that increasing epoch and training for longer, which theoretically leads to overfitting, does not necessarily increase the validation loss. In other words, the marginal increase in validation loss is so small that we can, in all practicality, ignore the effects of overfitting. Obviously however, this does not mean the model should be trained indefinitely, because over-training does not lead to any increase in performance and is simply a waste of time and resource.

Future work

1) First, after analyzing the data we can see that the distribution of training data on different labels is not uniform. This could be a drawback when we are training our model. To make the dataset balanced, one can add more training data on the labels with less data by rotation, cropping, adding noise or generation of new images using MNIST dataset.
2) When we are trying to bound the digits within the images, the result is not so satisfying. To improve the performance, we can have more information on the training set by knowing the three digits in the image separately. Training a single digit CNN should largely increase the accuracy.
3) A fine-tuning technique is to use the pre-trained weights on a model. First fix the CNN part of the model and only train the fully connected part. Then open the last two or three layers in the CNN and train the model with a small, exponential decay learning rate. We did not reach a high accuracy using a pre-trained weights for now, but this could be a crucial technique in increasing the performance.

Statement of Contributions

All team members have made certain contributions to the project, works were equally divided. Detailed division of work is as follows:

- Xiaohui Wang: Data preprocessing and experimenting basic approaches, writeup contribution
- Yun-Fei Cheng: Related work researching, report writeup
- Zijun Yu: Writing and training for divergent models and approaches, writeup contribution

Appendix

TABLE III: The count of targets in train set

Target	Count
9	13452
8	10621
7	8866
6	6184
5	4223
4	3171
3	1991
2	1029
1	421
0	42

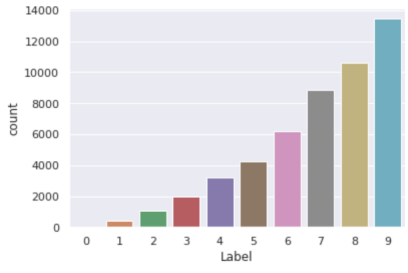


Fig. 7: Graph of target vs count

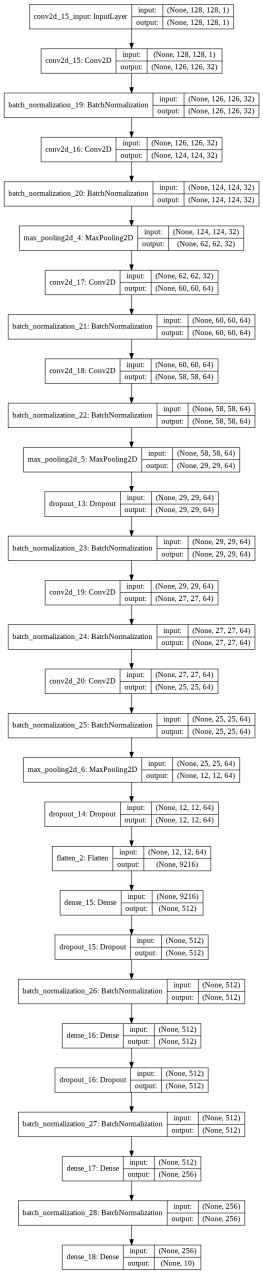


Fig. 8: The structural graph of a simple CNN with Kaggle score 94%

REFERENCES

[1] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, pp. 679–698, June 1986.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[3] K. He *et al.*, "Identity mappings in deep residual networks," 2016.

[4] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2016.

[5] B. Zoph *et al.*, "Learning transferable architectures for scalable image recognition," 2017.

[6] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[7] J. Deng *et al.*, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[8] L. Wan *et al.*, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. III–1058–III–1066, JMLR.org, 2013.

[9] Q. Xie *et al.*, "Self-training with noisy student improves imagenet classification," 2019.

[10] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.