Classification on Modified Digits

Team: Definitely need a Titan X

Yangchao Yi
yangchao.yi@mail.mcgill.ca
260742608

Zhiguo Zhang
zhiguo.zhang@mail.mcgill.ca
260550226

Yunhua Zheng
yunhua.zheng@mail.mcgill.ca
260733520

I. Introduction

This project aims to perform a classification task on the modified figures based on the MNIST dataset [1]. Each figure contains two digits and an operator indicated by the first letter of the operation, i.e. 'A' or 'a' for addition and 'M' or 'm' for multiplication. Our task is to build a model that correctly predicts the result of the mathematical operation on each input figure. For this purpose, we have implemented a baseline logistic regression classifier, two fully connected feedforward neuron networks and a convolutional neuron network (CNN). Other techniques have also been attempted at to boost the performance, e.g. cost-sensitive learning and data augmentation.

II. Related Work

On image classification tasks, CNNs outperform fully-connected neuron networks, and even a very simple CNN can achieve a high-level performance, as demonstrated in [2]. The great feature extraction capability of deep learning methods has been recognized; however, a large amount of training data must be available to avoid overtraining [3]. It is thus critical to generate additional data based on what is available. A such method, elastic distortion, has been adopted in [2]. On the other hand, authors of [3] have proposed a novel method to work with limited training samples using Gabor filtering and deep CNN, for hyperspectral images (HSIs) specifically. Techniques like dropout, rectified linear unit (ReLU) and batch normalization (BN) have been incorporated to further improve the performance of the proposed framework [2]. For interested readers, an extensive overview of CNN and related papers can be found in [4].

Despite the exponentially increasing amount of available data, the issue of imbalanced data can greatly weaken the performance of most classifiers with the assumption of balanced class distributions [5]. There have been a large number of publications addressing this issue already and that number sees an exponential increase; the general goal is to ensure balanced accuracies of prediction across different classes [5]. Three main methods to mitigate negative effects imposed by the imbalance are: 1. Sampling methods, e.g. oversampling, under-sampling, and synthetic sampling, where a balanced dataset is produced; 2. Cost-sensitive methods, where a cost matrix is designed such that misclassifying a less frequent outcome receives a greater penalty; 3. Kernel-based methods and active learning methods, which can be applied on support vector machines (SVMs) [5].

III. Problem Representation

The images in the dataset are represented as vectors of length 4096, with each value representing the greyscale of a pixel. All the greyscale values of the pixels form the feature space. As there are various types of shadows in the background, some kind of noise filtering has been performed to reduce such distracting factors when implementing the fully-connected neuron network. Specifically, the greyscale of each pixel in the image is converted to a binary value based on a certain threshold, e.g. 200 for the greyscale ranging from 0 to 255. So, the input becomes 4096 binary values. Though this simplifies the problem, it may have eliminated useful information at the same time. When implementing the other two classifiers, input normalization has been performed instead.

Besides, the 1-hot encoding has been implemented for the output. The possible outputs are 40 discrete integers with two operands ranging from 0 to 9 and two possible operators 'add' and 'multiply'. Consequently, there are 40 output units, each corresponding to a certain class. The desired output of a class is 1 at the corresponding output unit and 0 elsewhere. Usually, a softmax function is conducted as the final step to convert the values at the output into the probabilities that the input figure belongs to different classes.

Moreover, based on the properties of the training set, the following more advanced strategies have been considered and adopted to further improve the performance of CNN.

A. Data Balancing

As the number of examples in each class is not uniform (illustrated as the frequency of occurrence in the training set in Fig. 1), weighting methods have been adopted to penalize more

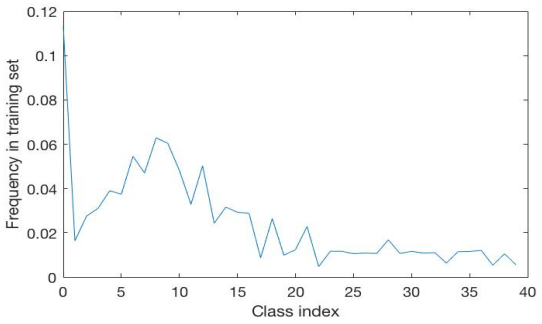


Fig. 1 Class distribution

severely the misclassification of a less probable outcome than that of a more probable one. For example, when implementing the neuron network with cross-entropy loss function, the term associated with a less probable outcome can be weighted more heavily. In the CNN implementation, the class weighting method inspired by [6] is used: for a class A, the weight is specified as

$$weight\ (A) = \frac{\#\ samples}{\#\ classes \times \#\ samples\ in\ class\ A}$$

where $\# \cdot$ denotes "the number of", and we have $\#\ samples = 50000$ and $\#\ classes = 40$ for this case.

Using this weighting method, our CNN model penalizes more the misclassification of minority class. Table I displays the class weights of the training data in an increasing order of class labels (left to right, up to bottom).

TABLE I. CLASS WEIGHT OF TRAINING DATA

0.22063	1.51413	0.91019	0.81286	0.62919
0.66293	0.45694	0.53116	0.39654	0.41088
0.51867	0.76791	0.49712	1.03211	0.79113
0.84969	0.87890	2.94502	0.95826	2.52809
2.01252	1.09863	5.20833	2.09497	2.15105
2.34864	2.29124	2.32919	1.51413	2.32438
2.12665	2.31958	2.28194	3.91986	2.14694
2.17181	2.13472	4.61065	2.32438	4.42913

B. Data Augmentation

A variety of data augmentation methods have been used in this project. Since the training data is constructed using handwritten digits and characters, a good classifier should predict the correct result that is independent of basic 2D and 3D transformations of images.

Random 2D image transformations including translation, rotation, zooming and shearing [7], have been applied on the training dataset, which significantly increases the size of training data. Additionally, elastic distortion in [2] has been employed to further enlarge the training dataset. The analogy of elastic distortion is that in real life, a handwritten paper may not be flat. Two parameters for elastic distortion is alpha and sigma as described in [8]. In this project, alpha = 36 and sigma = 6, 8, 10 have been chosen, as a larger distortion would lead to an ill-formed image. Fig. 2 visualizes the effect under elastic distortion with alpha = 50 and sigma = 5.

Moreover, perspective skew has been used with the Augmentor package [7]. The rationale is that the classifier should be invariant about perspective changes. There are 12 possible perspective skews used in this project, including tilting left, right, forward, or backward, and skewing around 8 points of the image. Here is an example of a skewing about the middle point of the left edge on the training data (Fig. 3). Both elastic distortion and perspective skew mimic possible real-world variations on handwritten data.

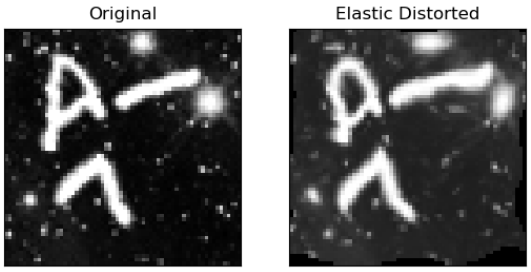


Fig. 2 Elastic distortion

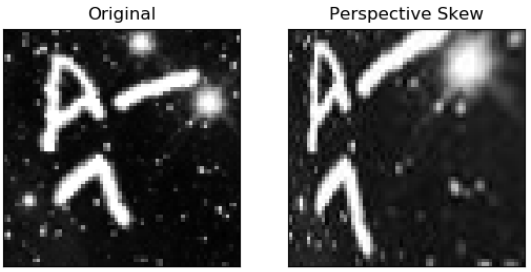


Fig. 3 Perspective Skew

IV. ALGORITHM SELECTION AND IMPLEMENTATION

To begin with, a logistic regression classifier has been implemented as the baseline using the scikit-learn library [9]. Then, two versions of neuron networks have been implemented from scratch. The last classifier is the CNN implemented using packages [10]. In this section, implementation details will be given.

A. Logistic Regression

A baseline learner is built based on logistic regression. The 4096 pixels are considered as 4096 input features of an image. The regularization for logistic regression is L2 penalty. The solver used for this learner is stochastic average gradient descent (SAG). Since our training data is very large, SAG gives a faster convergence than other algorithms. However, SAG is very sensitive to the scale of features. Hence, StandardScaler from the scikit-learn library [9] is employed to standardize the features. Stopping criterion is set to be 0.01 as at 0.1 the results fluctuate significantly, while at 0.001 the convergence is slow. Moreover, cross-entropy loss is chosen for multi-class classification. The training data is split into training data and validation data with 9:1 ratio.

B. Fully Connected Feedforward Neuron Network

The neural network is considered a universal function approximator, which means that with enough nodes, a two-layer neural network can approximate any function within an error bound. Our goal is to build a neural network such that given the training data, it can automatically recognize the numbers and operator in an image, operate the operation and output the result. Most parts of the neural network are trivial and easy to implement; the critical part is computation of the derivative of the loss function with respect to the weights and biases, i.e.

backpropagation. We employed a categorical cross entropy as the loss function, which is given as follows:

$$L(x) = \frac{1}{n}\Sigma_{i=1}^{n}\Sigma_{j=1}^{40}[y_{i,j} \cdot \log\big(h(x_i)_j\big) + \big(1 - y_{i,j}\big) \cdot \log\big(1 - h(x_i)_j\big)]$$

where the function $h(x)$ is the function represented by the fully connected neural network. If we expand the $h(x)$ in terms of the nodes in the last hidden layer:

$$h(x) = \sigma(w_l^T \cdot o_{l-1} + b_l)$$

where l is the number of layers and o_{l-1} is the output of l-th layer. The derivative of o_{l-1} with respect to the above loss function $L(x)$ is

$$\frac{\partial L}{\partial o_{l-1}} = o_{l-1} - y$$

because we employ the sigmoid function as the activation function on all nodes, and we have:

$$o_{l-1} = \sigma(w_{l-1}^T \cdot o_{l-2} + b_{l-1})$$

$$\frac{\partial o_{l-1}}{\partial w_{l-1}} = o_{l-2} \cdot [o_{l-1} \cdot (1 - o_{l-1})]$$

$$\frac{\partial o_{l-1}}{\partial b_{l-1}} = o_{l-1} \cdot (1 - o_{l-1})$$

Using the group of equations above recursively, we can compute the derivatives with respect to parameters on each layer to get the total gradients δW and δB. Then we use the gradient descent equation to updata all the parameters in the network:

$$W = W - \alpha \cdot \delta W$$

$$B = B - \alpha \cdot \delta B$$

In practice, we usually split the whole dataset into many subsets called batches, and update the weights and biases on every batch. Because this can accelerate the convergence of training.

Another implementation uses tanh in place of the sigmoid function and uses the cross-entropy error as the loss function [11]. This implementation is essentially the same as that presented in [11]. With the softmax function at the final layer, considering the input layer as layer 0, the output at the output layer (layer 2) $\hat{y}$ is obtained through following equations:

$$\hat{y} = softmax(z_2)$$

$$z_k = a_{k-1}W_k + b_k, \qquad k = 1, 2$$

$$a_k = \tanh(z1),\ if\ k > 0\ and\ a_k = x, if\ k = 0$$

where W_k is the weight matrix between layer $k-1$ and layer k, b_k is the displacement vector, and x is the input.

Ignoring the influence from other output units introduced by the softmax function, the gradient descent rules for a three-layer neuron network are as follows [11]:

$$\frac{\partial L}{\partial W_2} = a_1^T\delta_3, \qquad \frac{\partial L}{\partial b_2} = \delta_3$$

$$\frac{\partial L}{\partial W_1} = x^T\delta_2, \qquad \frac{\partial L}{\partial b_1} = \delta_2$$

where $\delta_3 = \hat{y} - y$, and $\delta_2 = (1 - tanh^2 z_1) \circ \delta_3 W_2^T$. The gradient descent can be easily extended for four or more layers.

After analyzing the "compute_class_weight" function from scikit-learn [9], it has been found that the returned weight for each category is the ratio of 2 and the number of training examples in that category, so it is easy to implement by hand and it has been incorporated into the code of the neuron network.

C. CNN

Inspired by VGG models [8], our CNN model consists of 7 convolution layers with 3 by 3 kernels, 2 fully connected layers of 512 nodes followed by an output layer of 40 nodes. The details of our model are documented in Appendix A.

We have applied a Lambda layer at the beginning to normalize the input images with the following formula:

$$x = \frac{x - \mu}{\sigma}$$

where μ is the mean value and σ is the standard deviation of the input image. Every two convolution layers, a batch normalization layer, a max-pooling layer with 2 by 2 kernels, and a dropout layer with the dropout probability of 0.25 are inserted. Batch normalization layer scales the features, which makes gradient descent easier. Max-pooling layer reduces the dimension of training parameters. Dropout layer enhances the generalization of the model and reduces the amount of computation. Furthermore, the number of filters in convolution layers is doubled so that more features of the image can be recognized as learning goes deeper.

After 3 blocks of convolution layers, there is a flatten layer added. Flatten layer unfolds a 2D array into an 1D array for fully connected layers. For each fully connected layer, there are also a batch normalization layer and dropout layer serving the same purpose. The activation function used in all middle layers is a ReLU function as it empirically performs better than other activation functions. The output layer uses softmax for activation.

The loss function used in CNN is the categorical cross entropy loss function with class weights applied. The evaluation metric in the learning process is accuracy. Various optimizers, including SGD, momentum, RMSprop and Adam, have been examined and the best found is Adam optimizer given other factors fixed. Adam optimizer improves the performance with sparse gradients [12].

The hyper-parameters in this model includes batch size of each iteration, learning rate of the optimizer, and number of epochs. We have tested the effect on loss and accuracy of a variety of batch sizes. As shown in Appendix B(I), we see the effect of different batch size is quite negligible. However, larger mini-batch size reduces the variance of gradient descent updates [13]. Therefore, 256 has been chosen as our mini-batch size. As for learning rate, as shown in Appendix B(II), for the first 20 epochs, there are enormous fluctuations for learning rate of 0.1, while learning is too slow for learning rate of 0.0001. Therefore, learning rate of 0.001 is chosen for Adam optimizer. Regarding the beta parameters, we believe that the default setting works the best for momentum effect. As for number of epochs, the model may have overtraining issues as the number of epochs increase.

However, due to limited computation power, the overtraining point of our model has not been found. In Appendix B(III), accuracy and loss plots are given for 200 epochs.

At the preprocessing step, random elastic distortion and perspective skew are applied on training data to generate extra data. Extra data of two times the size of the original data has been generated by perspective skew and of three times the size by elastic distortion. In total, we have 300,000 input images for training. During training, an ImageDataGenerator from [10] is used for random translation, rotation, shearing and zooming, which magnifies the training data by 500 times.

V. Test and Validation

The results obtained with the trained models on the training set and a detailed analysis of them will be presented in this section. As we are aware of the limitations of the first two classifiers and have expected that CNN is the best classifier among the three, more efforts have been made on the CNN classifier.

A. Logistic Regression

With the baseline learner, the average precision, average recall, average F1-score and validation accuracy obtained are 0.05, 0.07, 0.06 and 0.0666 respectively. Just for reference, the achieved accuracy on Kaggle is 0.0695.

B. Neuron Network

The original training set is split into 4:1 for training and testing respectively. The learning rate has been set to 0.008, 0.01 and 0.012, and the number of nodes in the hidden layer has been set to 512, 1024 and 2048. After 1000 updates, the achieved training and testing accuracies for the 3-layer neuron network with tanh as the activation function are as illustrated in Fig. 4. Actually, though the learning rate 0.012 and the number of nodes 1024 seem to perform dominantly well, the accuracies are approximately the same in all cases. Another thing to notice is that though the model gets a training accuracy over 95% very quickly, as shown in Fig. 5, the testing accuracy remains low even at the end of 1000 iterations and it cannot be expected that this metric would increase much with more updates. The results obtained using our neuron network with the sigmoid activation function or 4 layers are similar and not included here.

As a severe drawback of the fully connected feed forward neuron network has been noticed after performing basic training and validation for 3-layer and 4-layer networks, no more layers have been attempted at.

C. CNN

All of our tests in this section are based on a 9:1 training/validation data split. One of the most noticeable observations during testing is that the training data size plays a dominant role in terms of classifier performance. At the first attempt, ImageDataGenerator is not used during the training process. 200 epochs are performed on the original data and the validation accuracy is approximately 73% on average. Extra epochs do not significantly improve the validation accuracy at this time. Then, 2D image augmentation is added. The model is trained on original data with 50 epochs and generated data with 200 epochs. The validation accuracy is then improved to 89.5%. Furthermore, increasing the number of training epochs on

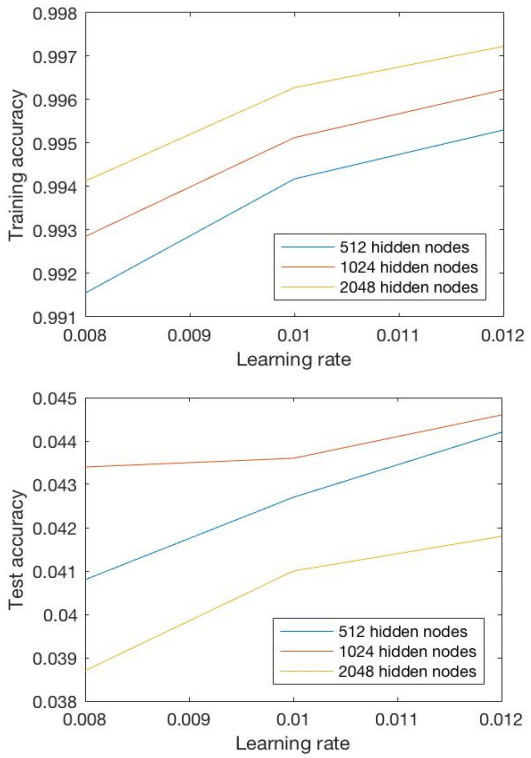


Fig. 4 Training accuracy (upper) and test accuracy (lower) after 1000 epochs for a 3-layer neuron network with different learning rates (0.008, 0.01 and 0.012) and different numbers of nodes in the hidden layer (512, 1024, 2048).

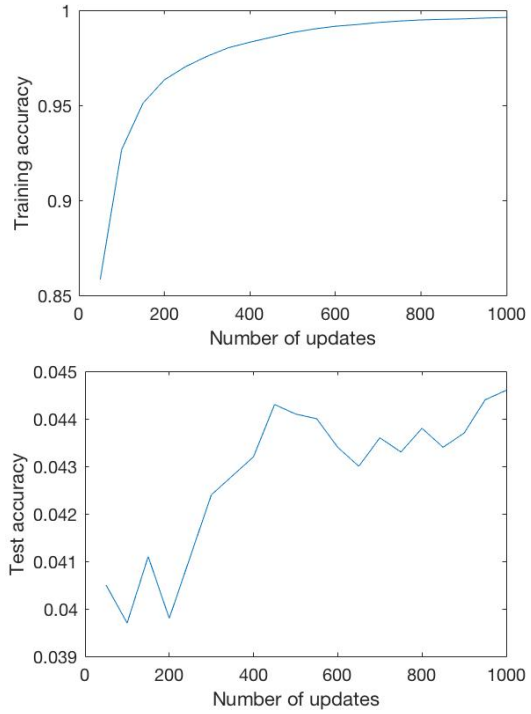


Fig. 5 The training (upper) and test (lower) accuracies with the number of updates.

augmented data to 500 further improves the validation accuracy to 92.9%. Since data augmentation makes a great contribution to the model performance, 3D image augmentation is performed on the original training data. With 3 times more training data generated from each original image by elastic distortion, we trained the model with the training data for 50 epochs and 2D augmented data for 500 epochs, and the validation accuracy has increased to 96.7%. Finally, we added random perspective skew on training data. Perspective skew alone increases the size of training data by 100,000. A similar training process is performed with new training data. Surprisingly, the validation accuracy reaches 99.28%. Table II shows the Kaggle test results.

TABLE II. TEST RESULTS OF CNN MODELS

Training data	Validation Accuracy	Kaggle Score
Original data	89.0%	90.325%
2D augmented	92.9%	92.350%
Elastic distortion	96.7%	95.575%
Perspective Skew	99.3%	98.675%

VI. DISCUSSION

From the last section, it is obvious that the CNN classifier performs much better than the other two classifiers. The logistic regression classifier is essentially a single neuron; obviously, it is too simple to learn any complicated structure in the image. In consequence, it generally cannot perform better than the neuron network, though it only takes a small amount of time to train. Besides, the results of fully connected neural network imply that the original neural network is good at fitting the data that it has seen, since it is called the universal function approximator; however, it performs badly on new data, i.e. it does not generalize well, which is understandable because the original neural network is not shift-invariant. For example, an image would seem totally new to a fully connected neural network if all the pixels of the image are shifted left by one pixel. This justifies the better performance of CNN as CNN essentially extracts the features from local areas of the image regardless of where the symbols are located. In a CNN, many filters are implemented on several layers to capture the features of the image at different levels. In the shallow layers, these filters learn some simple features, e.g., lines and angles. In the deep layers, the filters combine the simple features and represent more abstract and complicated features, e.g. some filters may try to match if a number '8' appears in some part of an image. This way, CNN is more robust to transformation of image and can generalize well, which is consistent with the validation results. Unfortunately, CNN has a much higher time complexity than the other two methods and requires much more time to train.

STATEMENT OF CONTRIBUTIONS

Zhiguo Zhang has implemented both the logistic regression and the CNN using packages. Yangchao Yi and Yunhua Zheng each implemented a neuron network from scratch.

We hereby state that all the work presented in this report is that of the authors.

REFERENCES

[1] Y. LeCun, C. Cortes and C.J.C. Burges, *The MNIST Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 12-Nov.-17].

[2] P.Y. SimardBest, D. Steinkraus and J.C. Platt, "Best Practices for Convolutional Neural Networks applied to Visual Document Analysis", *Proc. Int. Conf. Document Analysis and Recognit.*, Aug. 2003.

[3] Y. Chen, L. Zhu, P. Ghamisi, X. Jia, G. Li and L. Tang, "Hyperspectral Images Classification With Gabor Filtering and Convolutional Neural Network," *IEEE Geosci. Remote Sens. Lett*., vol. PP, no. 99, pp. 1-5, Nov. 2017.

[4] A. Deshpande, *The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3)*, [Online]. Available: https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html. [Accessed: 13-Nov.-17].

[5] H. He and E.A. Garcia, "Learning from Imbalanced Data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, Sept. 2009.

[6] G. King and L. Zeng, "Logistic Regression in Rare Events Data," *Political Analysis*, pp. 137-163, 2001.

[7] M.D. Bloice, C. Stocker, and A. Holzinger, "*Augmentor: An Image Augmentation Library for Machine Learning*," arXiv preprint arXiv:1708.04680**,** 2017. [Online]. Available: https://arxiv.org/abs/1708.04680. [Accessed: 13-Nov.-17].

[8] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *Comput. Vis. and Pattern Recognit.*, Sept. 2014.

[9] Pedregosa *et al, Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011.

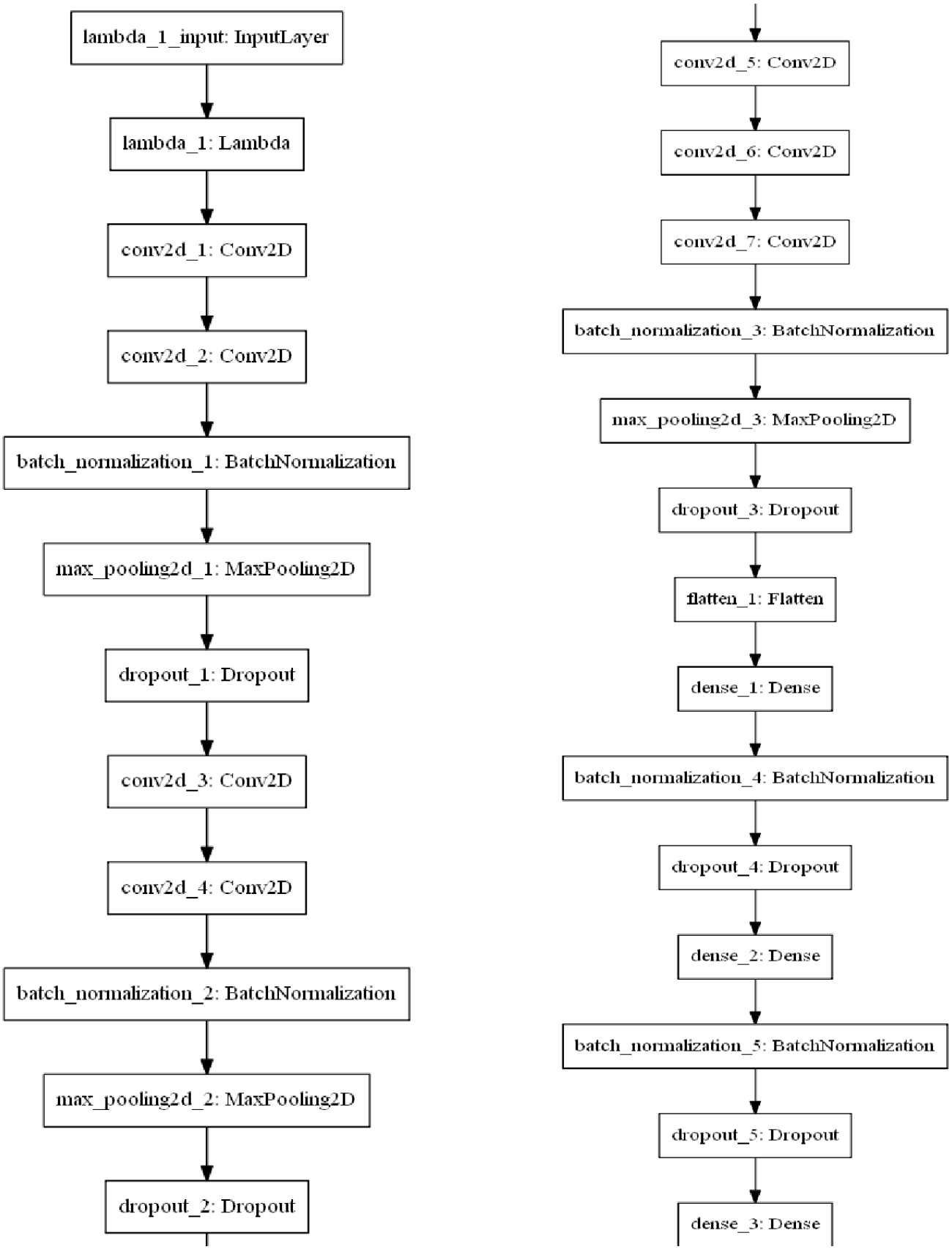
[10] F. Chollet et al, *Keras*, Github, 2015. [Online]. Available: https://github.com/fchollet/keras. [Accessed: 13-Nov.-17].

[11] D. Britz, *Implementing a Neural Network from Scratch in Python – An Introduction*, 2015. [Online]. Available: http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/. [Accessed: 12-Nov.-17].

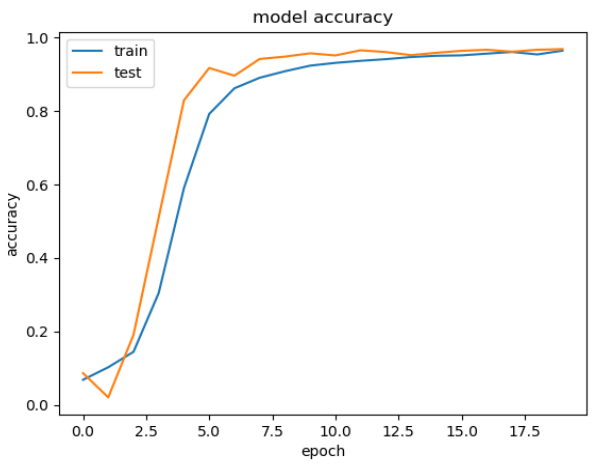
[12] D.P. Kingma and J.L. Ba, "Adam: A Method for Stochastic Optimization," *ICLR*, Dec. 2014.

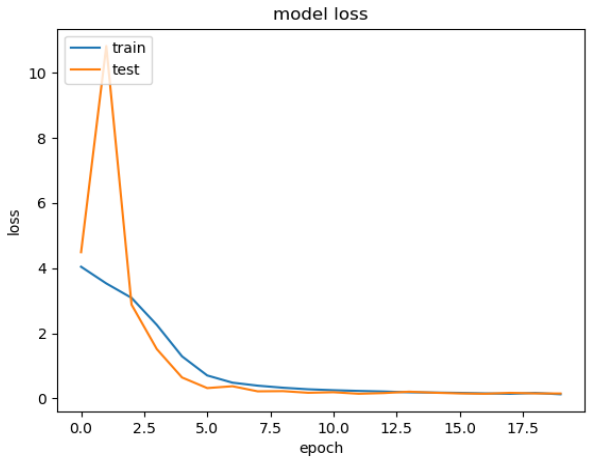
[13] L. Bottou, F.E. Curtis and J. Nocedal, "*Optimization Methods for Large-Scale Machine Learning*," arXiv preprint arXiv:1606.04838, Jun. 2017. [Online]. Available: https://arxiv.org/pdf/1606.04838.pdf. [Accessed: 13-Nov.-17].

Appendix A

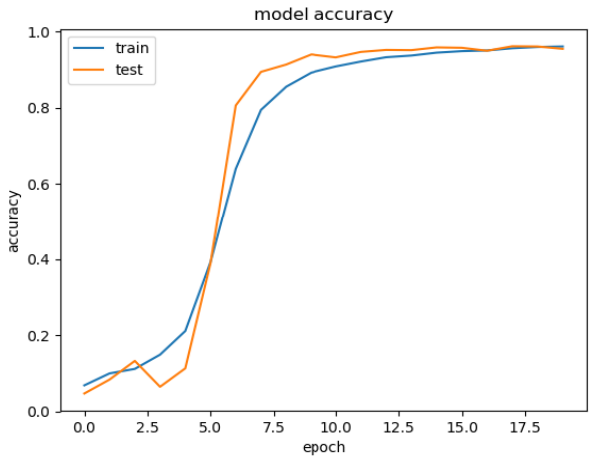

lambda_1_input: InputLayer
lambda_1: Lambda
conv2d_1: Conv2D
conv2d_2: Conv2D
batch_normalization_1: BatchNormalization
max_pooling2d_1: MaxPooling2D
dropout_1: Dropout
conv2d_3: Conv2D
conv2d_4: Conv2D
batch_normalization_2: BatchNormalization
max_pooling2d_2: MaxPooling2D
dropout_2: Dropout
conv2d_5: Conv2D
conv2d_6: Conv2D
conv2d_7: Conv2D
batch_normalization_3: BatchNormalization
max_pooling2d_3: MaxPooling2D
dropout_3: Dropout
flatten_1: Flatten
dense_1: Dense
batch_normalization_4: BatchNormalization
dropout_4: Dropout
dense_2: Dense
batch_normalization_5: BatchNormalization
dropout_5: Dropout
dense_3: Dense

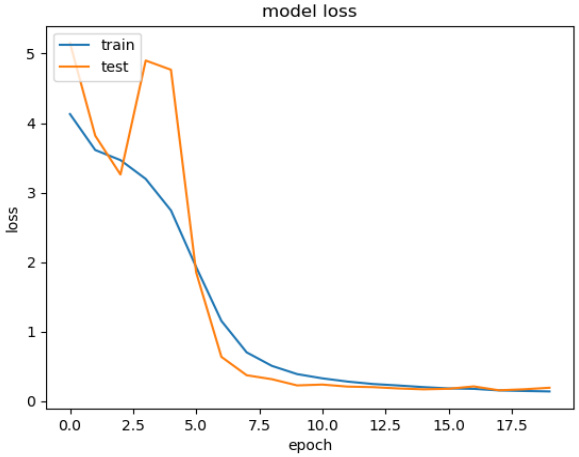
Appendix B(I)



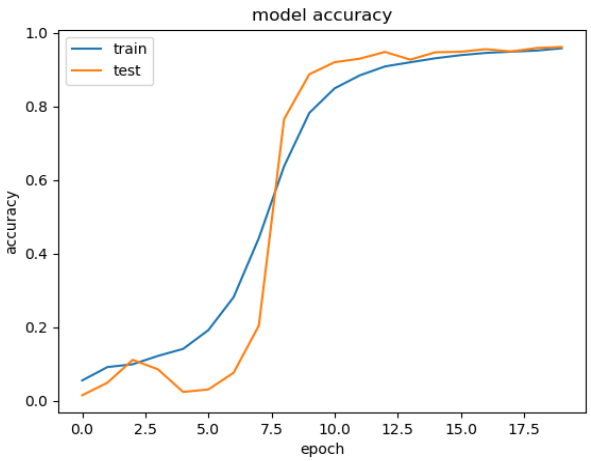


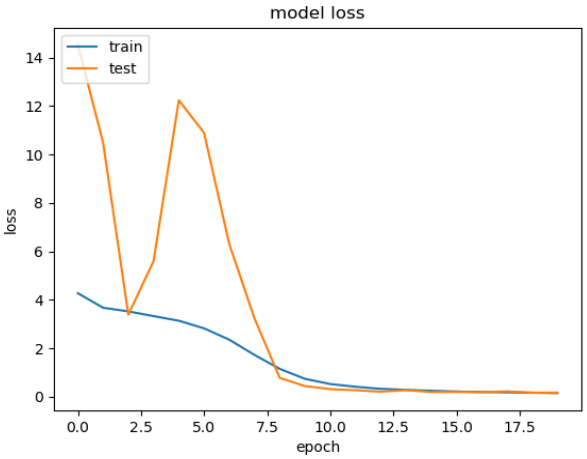
Batch size = 64, epoches = 20, learning rate = 0.001.





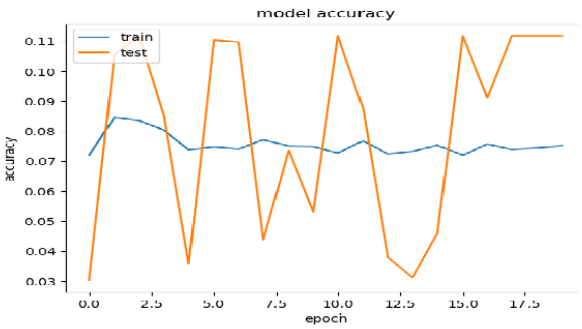
Batch size = 128, epoches = 20, learning rate = 0.001.

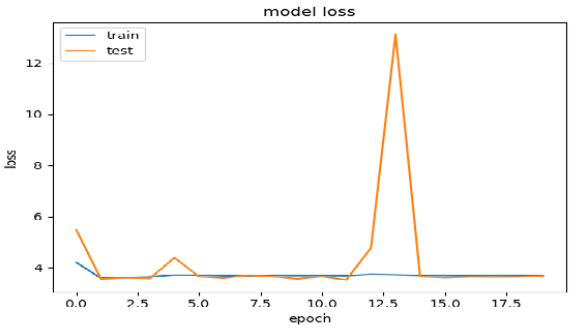




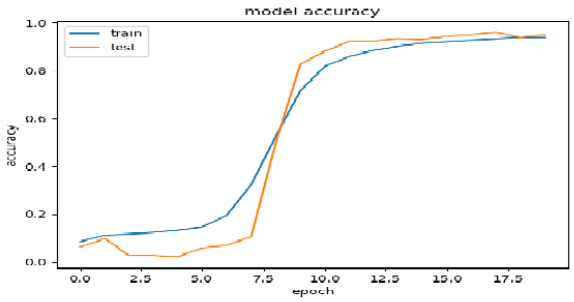
Batch size = 256, epoches = 20, learning rate = 0.001.

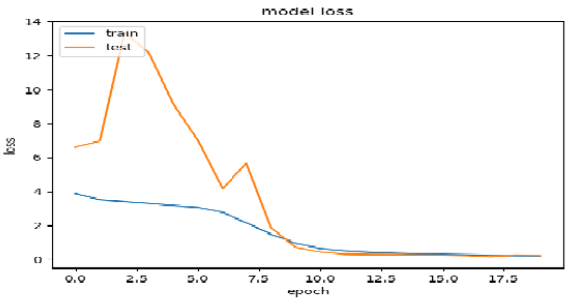
Appendix B(II)



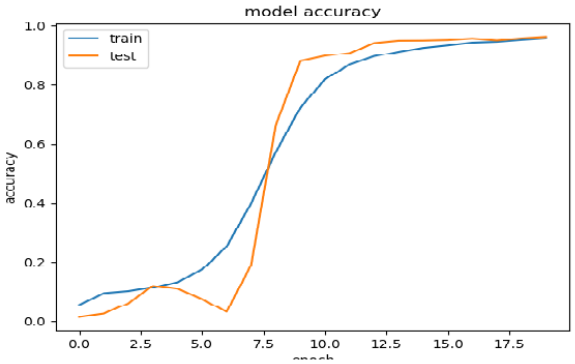


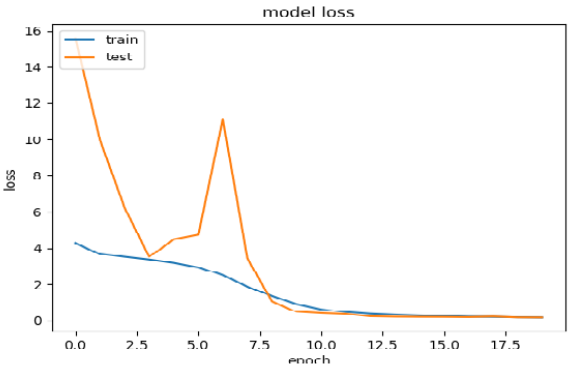
learning rate = 0.1, batch size = 256, epoch = 20.



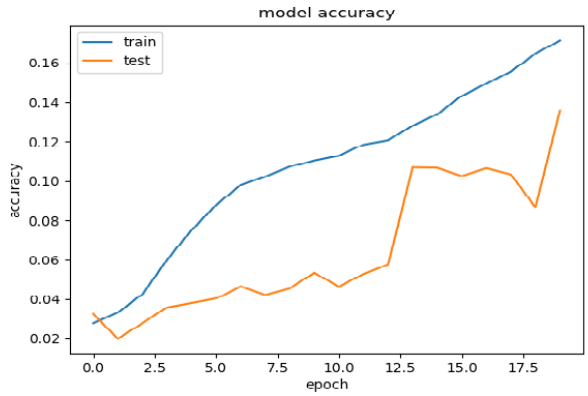


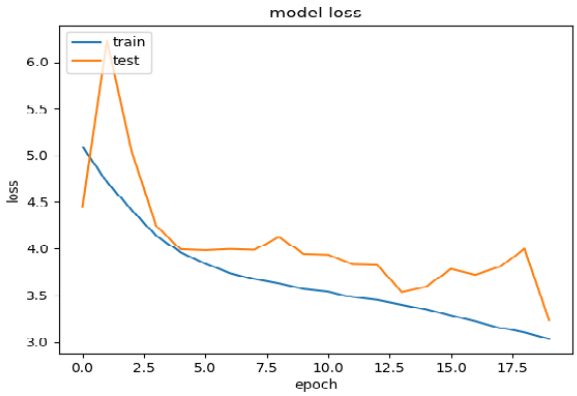
learning_rate = 0.01, batch size = 256, epoch = 20.





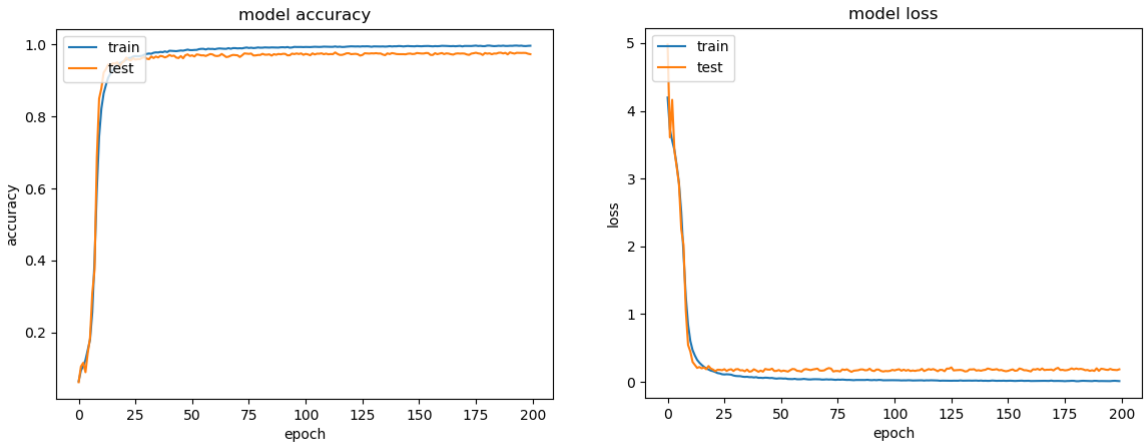
learning_rate = 0.001, batch size = 256, epoch = 20.





learning_rate = 0.0001, batch size = 256, epoch = 20.

Appendix B(III)



epoch = 200, batch size = 256, learning rate = 0.001.