Recognizing Hand-Written Characters using Tensorflow

Wooseok Kim, James Schallert
Department of Computer Science, University at Albany, SUNY
{wkim3, jschallert}@albany.edu

***Abstract* – The goal of this project was to use Neural Networks to recognize handwritten digits and characters using the eMNIST hand-written dataset. We used 6 different methods to build the neural network models and then calculated their prediction accuracy. The eMNIST dataset has 47 hand-written characters including 10 digits (0~9), 26 lower case letters, and 11 capital letters. The algorithms included single-layer SoftMax Logistic Regression, multi-layer Artificial Neural Network, ANN with Xavier Initialization, ANN with Dropout, Convolutional Neural Network, and Ensemble Neural Network. Ensemble Neural Network was found to have the highest accuracy.**

I. Introduction

The Tensorflow 1.0 Machine Learning library for Python was used to create a model capable of recognizing hand-written characters. When a document is scanned, a computer has difficulty recognizing all of the letters, so we wanted to attempt to improve that accuracy using various neural network strategies. Unlike normalized fonts used by computers, hand-written characters have high variation, making them difficult to be read by a computer. Our goal is to find the most efficient algorithm with the highest accuracy percentage.

Our training and test data are pulled from the eMNIST data set available from Kaggle [1]. We compared the 37 different hand-written characters and 10 digits using the following algorithms: SoftMax single-layer Logistic Regression, a basic artificial neural network, Xavier initialization, Dropout, Convolutional Neural Networks (CNN), and Ensemble CNN. We ran the algorithms on Google Colaboratory[3]. The main motivation for using Google Colaboratory was the availability of a GPU for cloud computing. This allowed us to build our models much faster than we could with local machines.

II. Related Work

Traditionally, the MNIST data set of 10 handwritten integers is used as a common dataset for introduction to image recognition and is even available directly in the TensorFlow library [2]. Accuracies for digit recognition is already well above 99% for the MNIST dataset. For our project, we wanted to do something similar to the MNIST project, but more difficult. We found the eMNIST dataset which is an extension of the MNIST dataset. This allows us to expand our image recognition from handwritten digits to include handwritten characters as well.

III. Methods

The different algorithms we tested were single-layer SoftMax Logistic Regression, basic Artificial Neural Network, Xavier initialization, Dropout, Convolutional Neural Networks (CNN), and Ensemble CNN. The first step, however, was to import the dataset from Google drive and wrangle it into a form useable by TensorFlow. A given image size is 28 x 28. To be useable by TensorFlow, we needed to flatten each image into a 1 x 784 input vector. There are 47 different hand-written characters including 10 digits (0~9), 26 lower case letters, and 11 capital letters.

A. Logistic Regression

For the first model we used single-layer SoftMax Logistic regression. Let's suppose $y = XW$

$$W = \begin{bmatrix} \omega_{11} & \cdots & \omega_{1n} \\ \omega_{21} & \cdots & \omega_{2n} \\ \vdots & \ddots & \vdots \\ \omega_{n1} & \cdots & \omega_{nn} \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Therefore, a given prediction y can be written:

$$y = WX => \frac{e_i^y}{\Sigma_j e_i^y}$$

Since Softmax function [4] is $S(y_i) = \frac{e_i^y}{\Sigma_j e_i^y}$, the output is a vector of probabilities. Tensorflow provides a function to easily create this output. $y = WX + b$ is calculated using TensorFlow's matrix multiplication and then is sent to the SoftMax function. To train the dataset, we use the below cost function:

$$cost(W) = -\frac{1}{m}\sum y \log(H(x)) + (1 - y)\log(1 - H(x))$$

We chose to use Gradient Descent Optimizer to minimize the cost function. The Gradient Descent Optimizer is calculated as:

$$W = W - \alpha \frac{\partial}{\partial W} cost(W)$$

Once the model is created, the highest probability from the probability vector is used as the prediction for image recognition.

B. Neural Networks

Figure 1 shows a 3-layer neural network. We used a 4-layer neural network for our basic artificial neural network algorithm, including 3 hidden layers. The first hidden layer has 784 inputs and 512 outputs. There are 784 inputs, one for each pixel in our 28 x 28 sized image. The second and third hidden layers both have 512 inputs and 512 outputs. The output layer has 47 outputs, corresponding to the 10 digits, 26 lower-case letters and 11 capital letters. We used ReLU for our activation function at each layer. ReLU stands for Rectified Linear Unit [8]. This provides a still-

non-linear alternative to the sigmoid activation function that helps us to avoid the vanishing and explosion of weights during training.

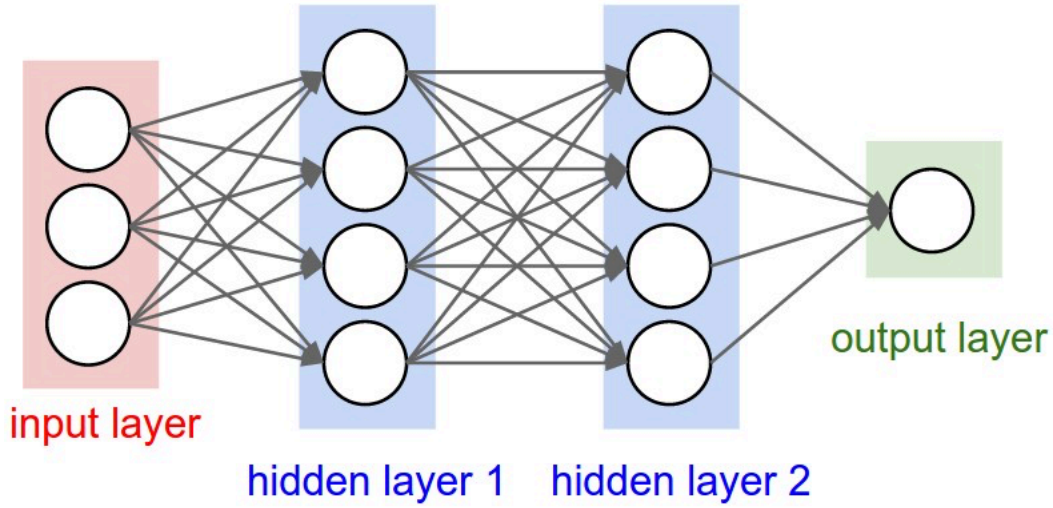

Figure 1. Neural Networks [6]

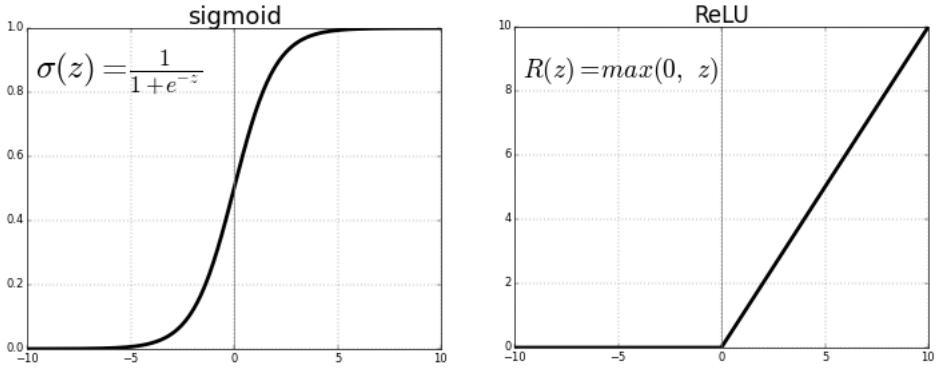

Figure 2. Sigmoid vs ReLU [7]

We used the same cost function as the SoftMax Logistic regression. However, we used the Adam Optimization algorithm instead of gradient descent. There are many optimization algorithms, but Adam optimization shows the best computational performance [9]. Tensorflow also has the Adam optimization algorithm built in [10].

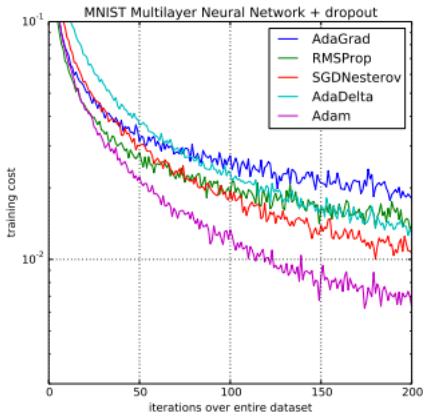

Figure 3. Optimization Algorithms [9]

C. Xavier Initialization

For Xavier Initialization, we use the same basic algorithm as the Neural Network above. The difference between the Xavier initialization algorithm and the Neural Networks model is in the initialization values [11]. Input and output weights are still created randomly, however, the defining characteristic of Xavier initialization is that the variance of the input weights is made equal to the variance of the output weights. Those values are divided by the square root of the input. This helps to normalize the input and output variance by dividing by the input standard deviation.

$$\frac{numpy.random.randn(input, output)}{\sqrt{input}}$$

TensorFlow has the function, prettytensor, built in for the Xavier initialization algorithm [12]. This is available as a library that we used in the hidden layers.

D. Dropout

Figure 4(a) shows an example of a basic neural network. Figure 4(b) shows a phase of dropout, where nodes are randomly disconnected for one pass of training. In practice, we randomly set some neurons' output to zero in the forward pass. This helps to avoid overfitting as well as node co-dependence. Networks can memorize a training data set while training, so dropout can be used when we train our model to introduce variation. Dropout only occurs during training. When the model is being tested or implemented, dropout is no longer used.

In order to implement dropout, Tensorflow has a function, *keep_prob,* that implements dropout by specifying how much of the networks we will keep. According to what we study, 50% ~ 70% of networks are kept while training for maximum accuracy.

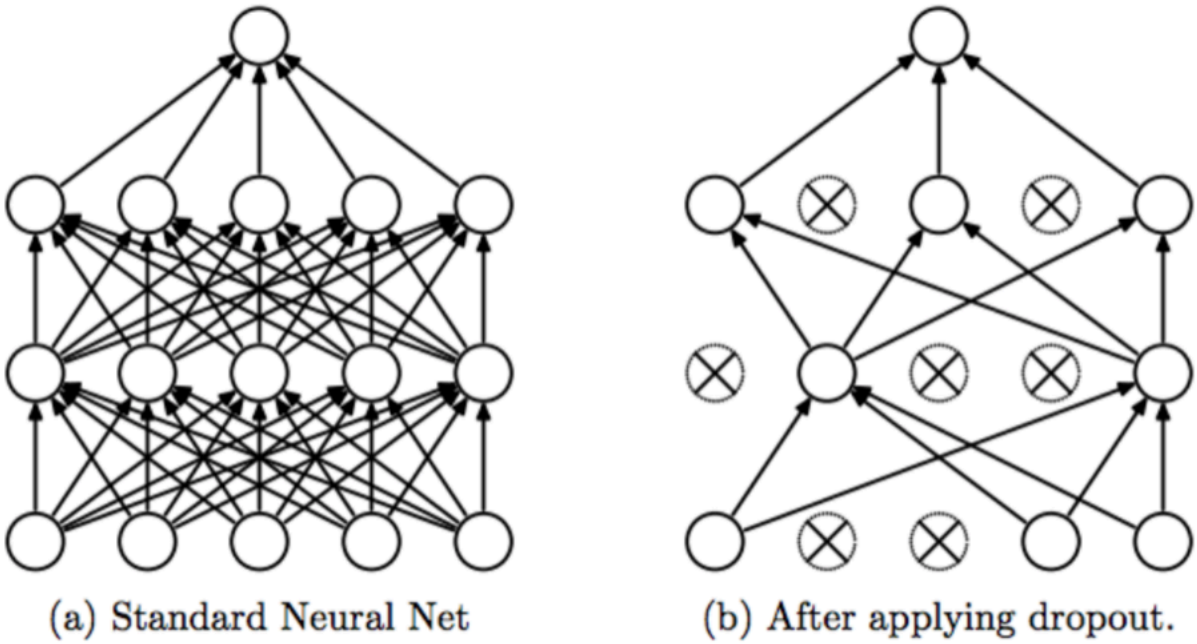

Figure 4. Dropout Neural Network Model [13]

E. Convolutional Neural Networks (CNN)

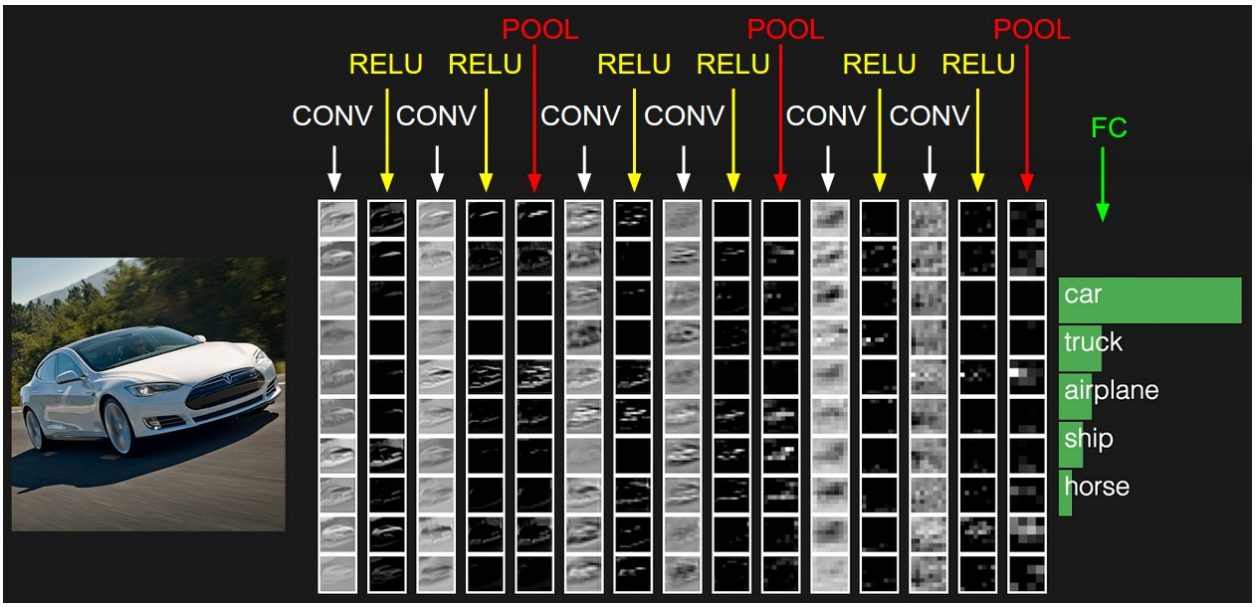

Figure 5. Convolutional Neural Network Architecture [6]

Convolutional Neural Networks are made up of a sequence of layers like Neural Networks. However, there are three types of layers: Convolutional (CONV) Layer, Pooling (POOL) layer, and Fully-Connected (FC) Layer. The three layers are used in order to build a Convolutional Neural Network like the one pictured in figure 5. The CONV layer computes an output which is connected to local regions in the input by passing a filter over the image, then applies a ReLU activation

function. POOL layer makes a smaller image than the original image by performing a down sampling operation. FC layer computes the classes, which are 47 different hand-written characters. The highest probability is then chosen as the image class.

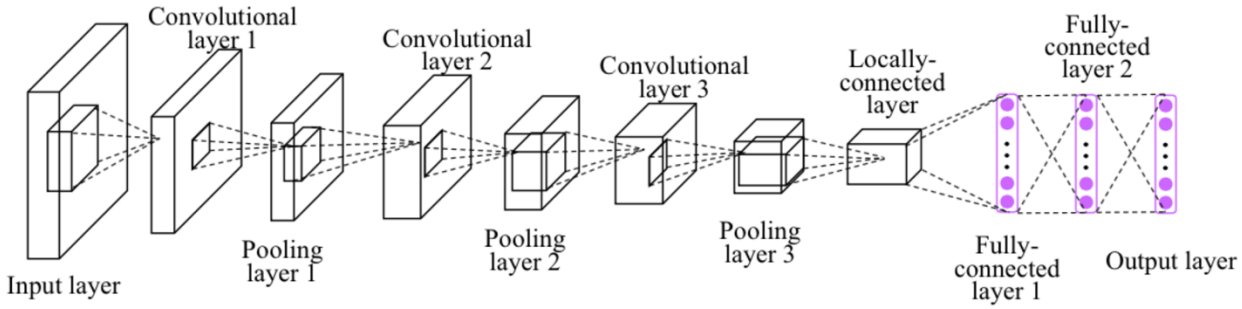

Figure 6. CNN Structure

We used a Convolutional Neural Network structure similar to figure 6 [14]. The Input layer is our 28 x 28 hand-written image. In the Convolutional layer 1, we chose 32 filters with a size of 3x3, resulting in 32 images. Next, a 1x1 filter with a stride of 1 is passed over the 32 images. The last step of the first convolutional layer is then the ReLU activation function. In the Pooling layer 1, the input is the matrix of ReLU values from the Convolutional layer. We then use a 2x2 kernel with a 2x2 stride to reduce the dimension of the original image. In this model, the 28x28 image is reduced to 14x14. 32 of these images are passed on because of the 32 filters used in the first convolutional layer. The final stage of the pooling layer is dropping the least relevant of these 32 images. Convolutional layer 2 and pooling layer 2 are very similar to the first two, further reducing the dimensionality while increasing the number of images. The image becomes smaller and smaller as it passes through each pooling layer. However, as the image passes through each convolutional layer, the number of images is increased. For example, layer 2 has twice the output images of layer 1. Later, in the locally-connected layer, the images are reshaped and combined. Next, the fully-connected layer uses ReLU and dropout, similar to the convolutional and pooling layers. finally, the second fully-connected layer outputs the prediction probability for each of the 47 different characters.

F. Ensemble

In the Ensemble algorithm, several neural network models are trained at the same time. These are called Classification models. When new data is added, each class makes a prediction accuracy value. The prediction values are then combined to make a final prediction. Each classification model and each prediction have their own Convolutional layer, Pooling layer, and Fully-Connected layer like the Convolutional Neural Network above. We essentially used four copies of Convolutional Neural Network describe above. TensorFlow has built-in functionality to run these CNNs in parallel and then combine their final predictions into a single prediction.

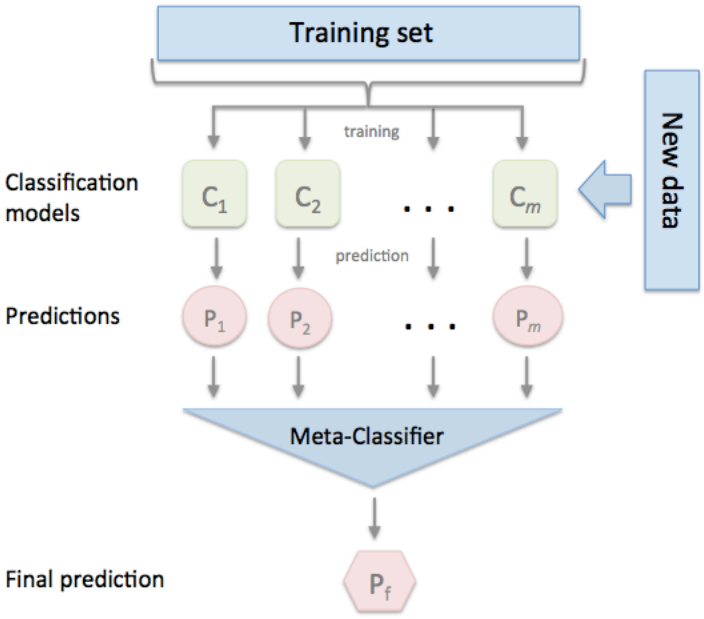

Figure 7. Ensemble [15]

IV. Experiments

Before we even started testing, we needed to determine our training and testing parameters. We needed to choose the number of Epochs, each being a single round of optimization using all of the training data. We also needed to choose the batch size: the number of training examples used during one forward pass and backward pass through the neural network. The trade-off is that the higher the batch size is, the more memory space we need. We also needed to choose the number of iterations: the number of passes using the batch size number of examples[5]. The eMNIST dataset was already separated into training and testing sets, so we simply used those as our split ratio of training to test data.

A. SoftMax Logistic Regression

In order to check if our model was correct, we passed the data from the test set through our model and compared its predictions to the training labels. When we trained our model, we set the epoch size to 20 and batch size to 100. After running all the test data through the model, we can see our accuracy.

After the training is done on the training dataset, we should use a different dataset to test it. It is important to use a test dataset as the model may memorize the given dataset while training. This overfitting is revealed when the model's prediction rate of the training dataset is high, but its prediction accuracy is low on the test dataset. To prevent that, we imported 2 different datasets, the training dataset and the test dataset.

As a control, we tried an experiment where we had the model guess the character from our hand-written dataset by randomly selecting a number. The random number decided the label of the test dataset, and then the computer would compare our hypothesis value and test data which we just picked. Finally, we printed the actual image, like below.

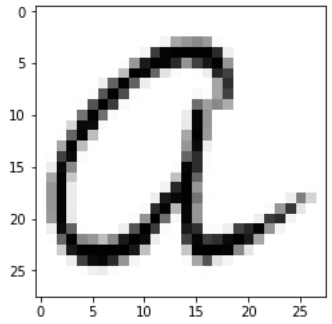

Figure 8. Hand-Written Image

B. Neural Networks

The testing method in the simple Neural network algorithm was the same as the SoftMax Logistic regression above. However, if we changed the number of nodes in each layer, the accuracy is different. When we tested the algorithm, 512 nodes showed the best accuracy, but it took much longer to finish training the model than 256 or 128 nodes.

C. Xavier

We used the identical cost function and Adam optimization algorithm as the simple Neural Network. The process of testing it was also the same. The only thing we changed were the initialization values, but we were still able to see an accuracy improvement over the simple Neural Network.

D. Dropout

A training process is very similar to Neural networks and Xavier initialization algorithm except for adding the TensorFlow function, *keep_prob*. We kept 70% of nodes during each phase of the training process. As discussed above, it is necessary to keep 100% of nodes while testing even if we applied dropout during the training process.

E. Convolutional Neural Networks (CNN)

Although the Convolutional Neural Network structure is different from simple Neural networks, the way to test and train the model is identical to the dropout algorithm. When we trained the model, we continued to keep 70% of the nodes. As before, we still needed to use all the nodes when we tested the model.

F. Ensemble

We used 4 CNN models to build the Ensemble CNN. We trained each model individually, the same as the previous CNN. When we fed the test dataset through the 4 models, we needed the output of each model to be an array of 47 elements: a prediction percentage value for each of the 47 hand-written characters. The 4 prediction arrays are the added and averaged. The highest probability is chosen as the prediction and compared with the test data labels.

V. Results and discussion

	Alphabet / Digits	Digits
SoftMax Logistic Regression	0.6085106	0.7697
Neural Networks	0.6993085	0.9823
Xavier	0.8368085	0.988775
Dropout	0.84957445	0.990575
CNN	0.89090425	0.996125
Ensemble	0.89521277	0.997225

Table 1. Algorithm Comparison

We compare 47 characters and 10 digits using 6 algorithms we discussed above. The SoftMax algorithm was not good for recognizing hand-written characters. The accuracy was the lowest of the six models, hovering around 60%. The simple Neural Network improved accuracy to 10% higher than SoftMax's accuracy for the eMNIST dataset. Although a marked improvement, it wasn't as huge of an accuracy boost as it was for the base MNIST dataset. The Xavier Initialization process further increased the accuracy for the eMNIST dataset greatly, adding another 14 percentage points. This showed just how important it is to set correct initialization values even before the dataset begins training. The Dropout algorithm performed even higher than the Xavier process. Changing the model to a Convolutional Neural Network enhanced accuracy even further, almost to 90%. In the end, the Ensemble algorithm showed the highest accuracy of all the methods, although only half of a percentage point higher than the single CNN.

VI. Conclusion

In our project, Ensemble showed the highest accuracy. Accuracy for prediction using the eMNIST dataset showed around 89.5%. Additionally, accuracy for 10-digit recognition in the base MNIST dataset is around 99.7%, showing great detection performance. We were able to improve accuracy by 29% for the eMNIST dataset 23% for MNIST compared to SoftMax by using the Ensemble Convolutional Neural Network.

Recently, many new algorithms are being developed such as AlexNet, ZFNet, GoogleNet, VGGNET, ResNet, and etc. We are sure that we can improve the accuracy of 47 characters by using one of these newer algorithms. In our next work we would seek to apply ResNet [18], which shows low error rates like figure 9. If ResNet algorithm was used in this project, we predict that our accuracy on the eMNIST dataset would be above 90%. Alternatively, we could try applying many of the algorithms together at once, as many of them are not mutually exclusive. One example would be an Ensemble Convolutional Neural Network implementing Xavier Initialization.

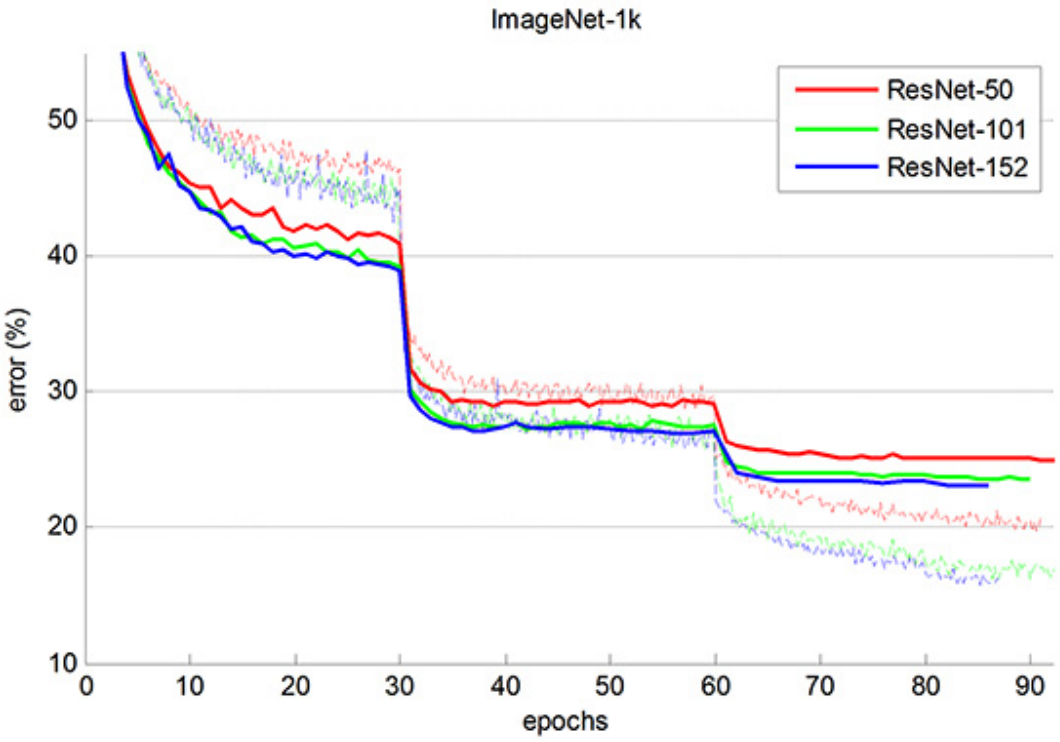

Figure 9. ResNet [17]

References

[1] https://www.kaggle.com/crawford/emnist

[2] https://www.tensorflow.org/get_started/mnist/beginners

[3] https://colab.research.google.com/

[4] https://en.wikipedia.org/wiki/Softmax_function

[5] https://stackoverflow.com/questions/4752626/epoch-vs-iteration-when-training-neural-networks

[6] http://cs231n.github.io/convolutional-networks/

[7] https://cdn-images-1.medium.com/max/1600/1*XxxiA0jJvPrHEJHD4z893g.png

[8] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[9] Diederik P. Kingma, Jimmy Lei Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION", ICLR 2015

[10] https://www.tensorflow.rog/api_docs/python/tf/train/AdamOptimizer

[11] Xavier Glorot, Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks", 2010

[12] https://github.com/google/prettytensor/blob/a69f13998258165d6682a47a931108d974bab05e/prettytensor/layers.py

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfiting", Journal of Machine Learning Research, May 2014

[14] Shuo Yang, Ping Luo, Chen Change Loy, Kenneth W. Shu, Xiaoou Tang, "Deep Representation Learning with Target Coding", AAAI Conference on Artificial Intelligence, 2015

[15] http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier_files/stackingclassification_overview.png

[16] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre´ van Schaik, "EMNIST: an extension of MNIST to handwritten letters", March 2017

[17] https://github.com/KaimingHe/deep-residual-networks

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep Residual Learning for Image Recognition”, arXiv, 2015