MNIST Dataset Analysis

Members: Po-Han Yen, Shih-Siang Lin,

Shu-Ping Chen, Yun-Rong Hsieh

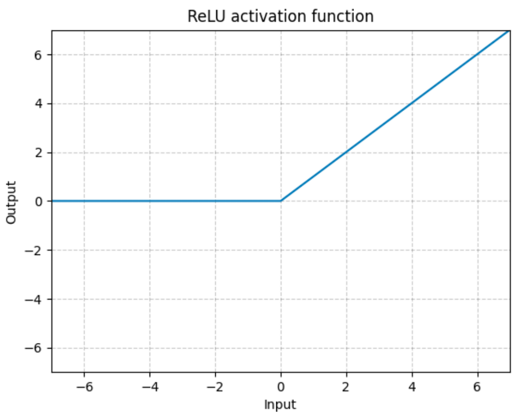
I. Methods used

The following are the steps we used to process the modeling and evaluation:

1. Normalized the values from 0 to 1 with the min-max method for both training and testing datasets.
2. Randomly splitted the training dataset into 80% training and 20% validation.
3. Convolutional neural network (CNN) model:

   (1) Introduction of the function we used

   ReLU function: $ReLU(x) = (x)^{+} = max(0, x)$



   Batch Normalization: $z^N = (\frac{z-m_z}{s_z})$

   m_z: the mean of the neurons' output; s_z: the standard deviation of the neurons' output.

   (2) Introduction of the model we created

   Input

   28x28 pixels images

Convolution Layer 1

Learned 64 (8x8) filters from the input images with 5x5 dimensions of the kernel, and then applied rectified linear unit function (ReLU) on this layer.

Batch Normalization

Normalized the features in the images.

Convolution Layer 2

Learned 64 filters from the input images with 5x5 dimensions of the kernel, and then applied rectified linear unit function (ReLU) on this layer.

Batch Normalization

Normalized the features in the images.

Pooling Layer

Divided the pixels of the feature map into 4 (2x2) patches, and selected the maximum value from each patch in order to contain the most prominent feature of the image.

Drop Layer

Nullified the features with probability 0.25 from the Bernoulli distribution.

Convolution Layer 3

Learned 64 (8x8) filters from the input images with 3x3 dimensions of the kernel, and then applied rectified linear unit function (ReLU) on this layer.

Batch Normalization

Normalized the features in the images.

Convolution Layer 4

Learned 64 (8x8) filters from the input images with 3x3 dimensions of the kernel, and then applied rectified linear unit function (ReLU) on this layer.

Batch Normalization

Normalized the features in the images.

Pooling Layer

Shifted the pixels with strides 3 and 3 for height and width, respectively, to divide the pixels of the feature map into 4 (2x2) patches; selected the maximum value from each patch in order to contain the most prominent feature of the image.

Drop Layer

Nullified the features with probability 0.25 from the Bernoulli distribution.

Convolution Layer 5

Learned 64 (8x8) filters from the input images with 3x3 dimensions of the kernel, and then applied rectified linear unit function (ReLU) on this layer.

Batch Normalization

Normalized the features in the images.

Drop Layer

Nullified the features with probability 0.25 from the Bernoulli distribution.

Flatten

Flattened the feature map into one column.

Dense Layer

Applied rectified linear unit function (ReLU) on this layer, and output arrays of shape (None, 256).

Batch Normalization

Normalized the features in the images.

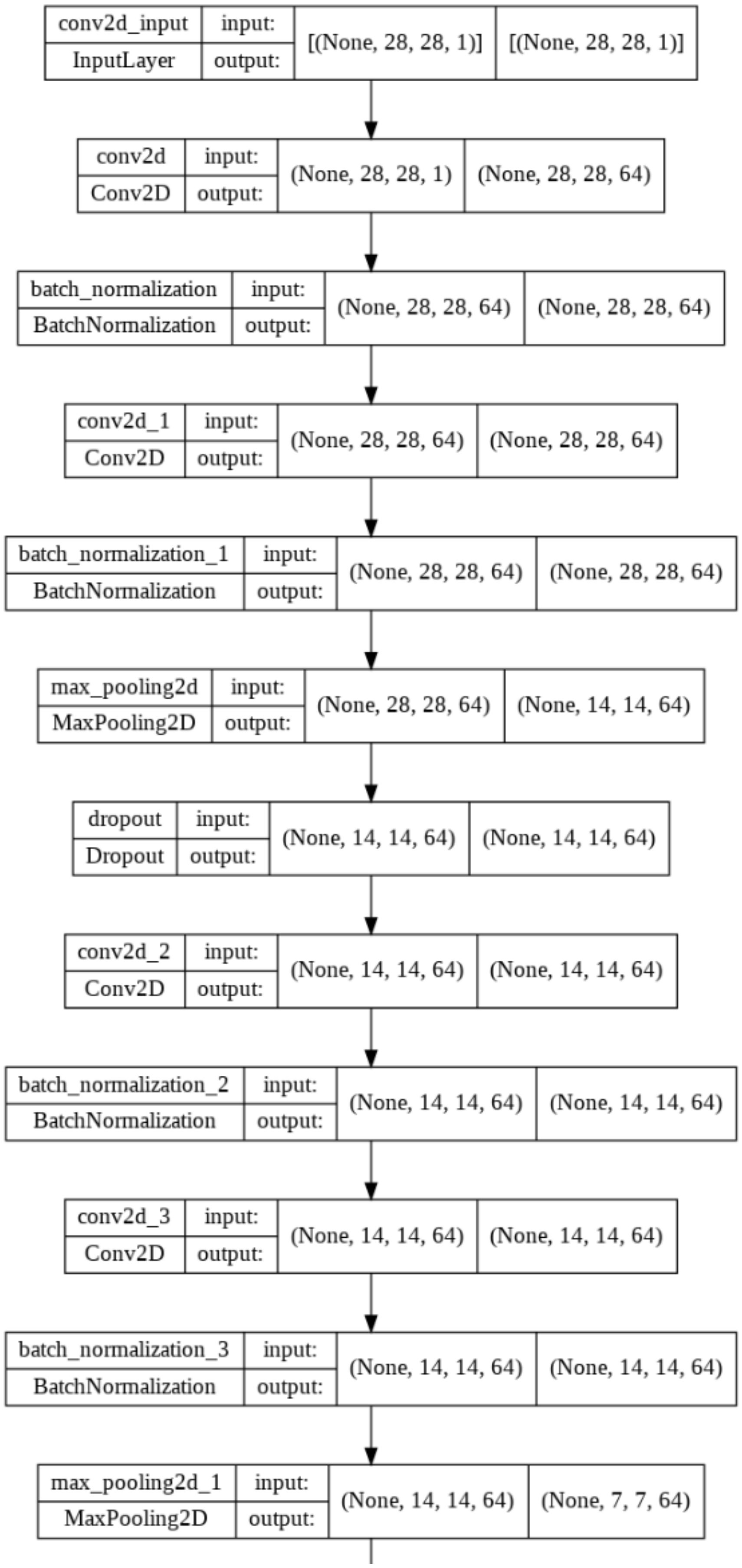
Drop Layer

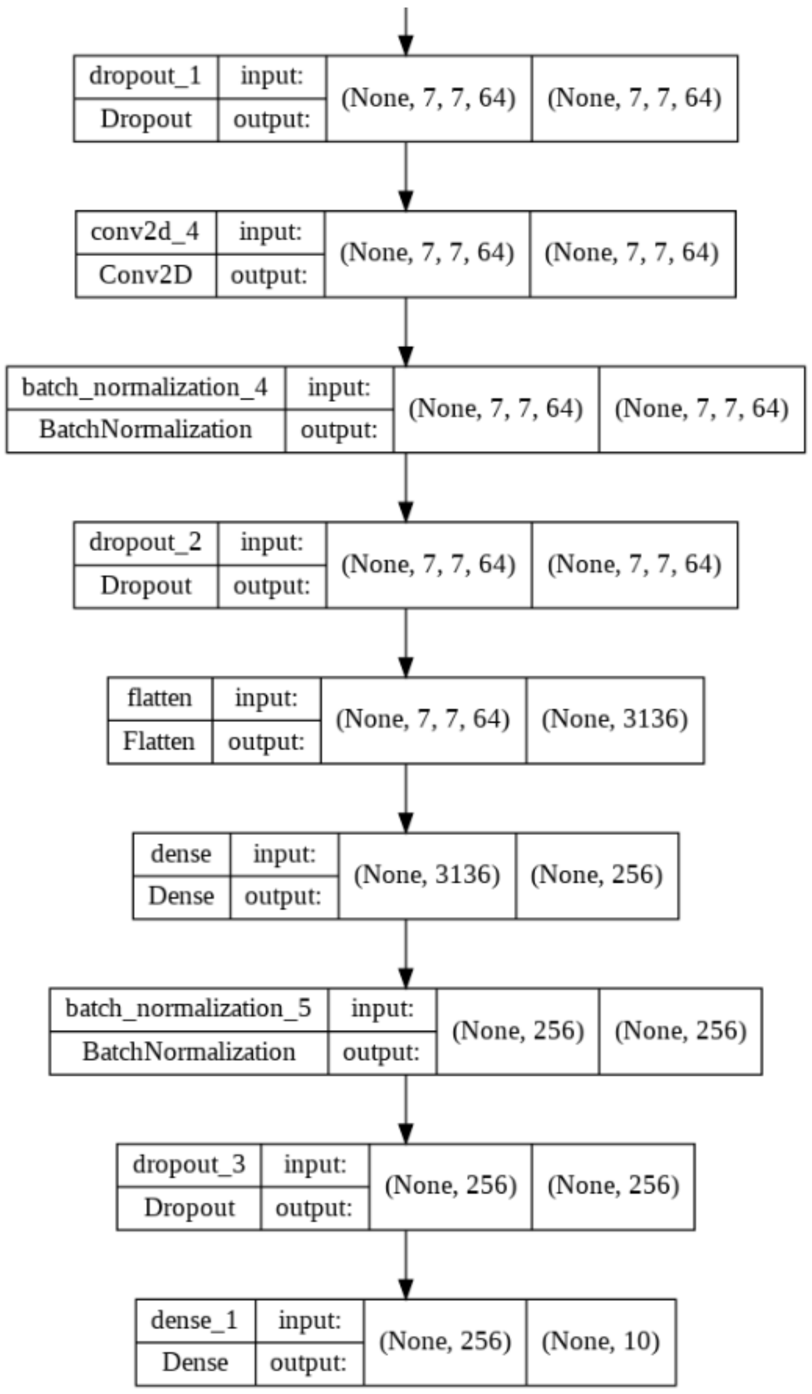
Nullified the features with probability 0.25 from the Bernoulli distribution.

Output

Applied Softmax function on this layer, and output the probabilities from 0 to 1 for each class (0-9) to find which class is most likely our input.

(3) The dimensions of each layer in our CNN model


conv2d_input | InputLayer | input: | output: | [(None, 28, 28, 1)] | [(None, 28, 28, 1)]
conv2d | Conv2D | input: | output: | (None, 28, 28, 1) | (None, 28, 28, 64)
batch_normalization | BatchNormalization | input: | output: | (None, 28, 28, 64) | (None, 28, 28, 64)
conv2d_1 | Conv2D | input: | output: | (None, 28, 28, 64) | (None, 28, 28, 64)
batch_normalization_1 | BatchNormalization | input: | output: | (None, 28, 28, 64) | (None, 28, 28, 64)
max_pooling2d | MaxPooling2D | input: | output: | (None, 28, 28, 64) | (None, 14, 14, 64)
dropout | Dropout | input: | output: | (None, 14, 14, 64) | (None, 14, 14, 64)
conv2d_2 | Conv2D | input: | output: | (None, 14, 14, 64) | (None, 14, 14, 64)
batch_normalization_2 | BatchNormalization | input: | output: | (None, 14, 14, 64) | (None, 14, 14, 64)
conv2d_3 | Conv2D | input: | output: | (None, 14, 14, 64) | (None, 14, 14, 64)
batch_normalization_3 | BatchNormalization | input: | output: | (None, 14, 14, 64) | (None, 14, 14, 64)
max_pooling2d_1 | MaxPooling2D | input: | output: | (None, 14, 14, 64) | (None, 7, 7, 64)



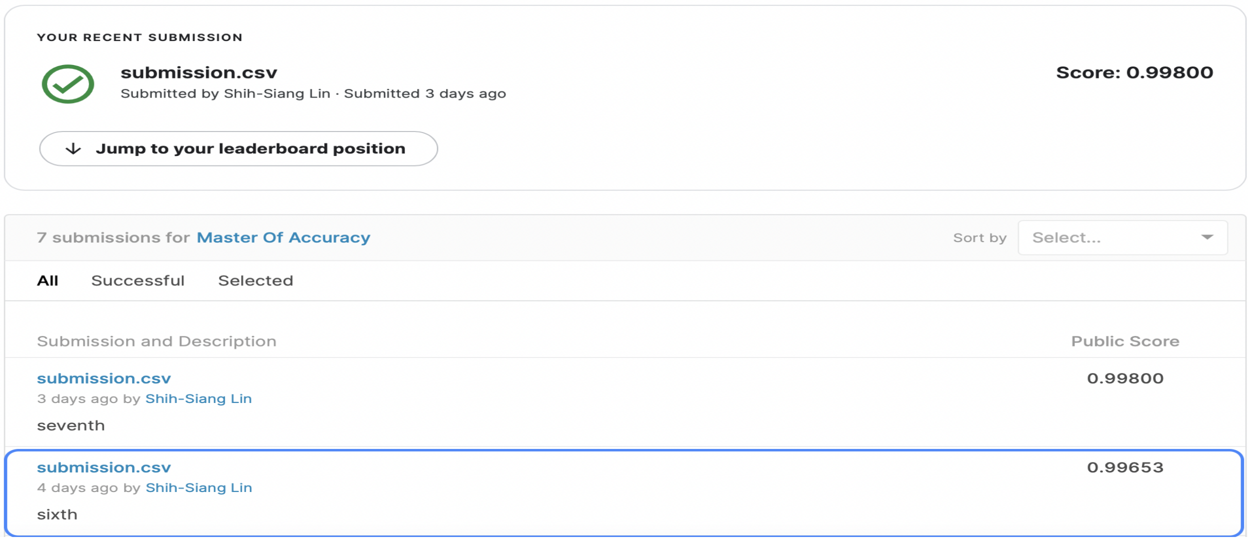
4. Implemented the RMSprop algorithm as optimizer with learning rate 0.001, rho 0.9, epsilon 1e-08, decay 0; used categorical cross-entropy function to compute the categorical cross-entropy loss.
5. Created 5 CNN models, and ensemble all of them to predict on the testing dataset.

II. Results and reports

Training Results:

```
[CNN 1]: Epochs=50, Train Accuracy=0.99845, Validation Accuracy=0.99619
[CNN 2]: Epochs=50, Train Accuracy=0.99839, Validation Accuracy=0.99655
[CNN 3]: Epochs=50, Train Accuracy=0.99839, Validation Accuracy=0.99726
[CNN 4]: Epochs=50, Train Accuracy=0.99833, Validation Accuracy=0.99631
[CNN 5]: Epochs=50, Train Accuracy=0.99764, Validation Accuracy=0.99607
```

Above are the results from five of the CNN models. For each, we print out the history maximum Train and Validation accuracy. Then we ensemble all of them to predict on the test data sets and output the submission.csv file as the same format Kaggle competition requested. Below is the score and ranking on the leaderboard:


YOUR RECENT SUBMISSION

submission.csv
Submitted by Shih-Siang Lin · Submitted 3 days ago

Score: 0.99800

7 submissions for Master Of Accuracy

Submission and Description	Public Score
submission.csv 3 days ago by Shih-Siang Lin seventh	0.99800
submission.csv 4 days ago by Shih-Siang Lin sixth	0.99653

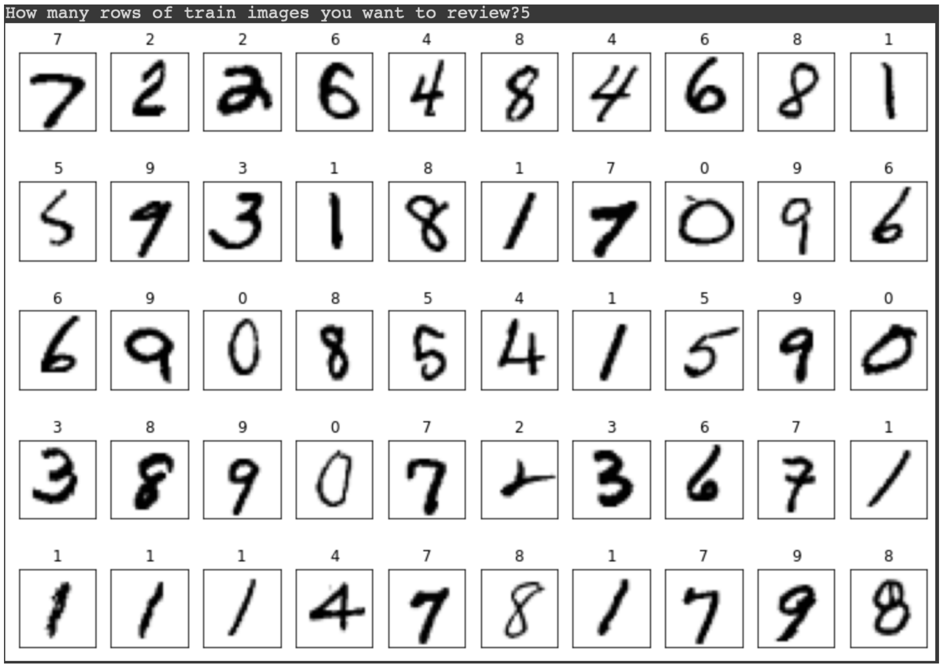
Score 0.99653 ranks around 100 out of 2133 teams on the leaderboard, proving that this CNN model performs pretty well on the image detection of mnist data set.

One thing I would like to mention is that we tried to use the full data set to train the model, and get a higher score of 0.998 which is reasonably expected since the full data sets contain all the data labels, definitely resulting in higher accuracy in the testing data sets. However, this is just for experiment because it is prohibited to do so following the competition instructions. Therefore, score 0.99653 would be our final result.

III. What did you do and what have you learned

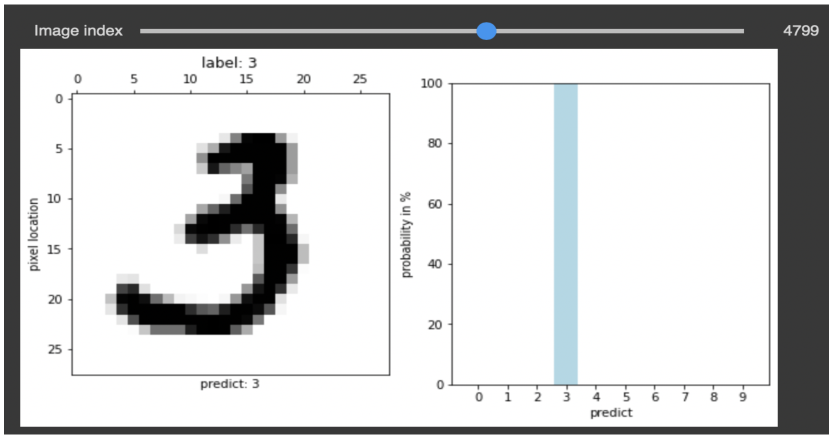
- **Visualization on the training and validation data sets**

1. Created the interactive surface to display the training dataset:
   Ask users how many rows of train images they want to review, and show each image in a row x 10 plots.



```
How many rows of train images you want to review?5
```

2. Visualized the prediction result on validation data set with an interactive widgets: Users can scroll the slider bar to show the prediction result of each validation image.



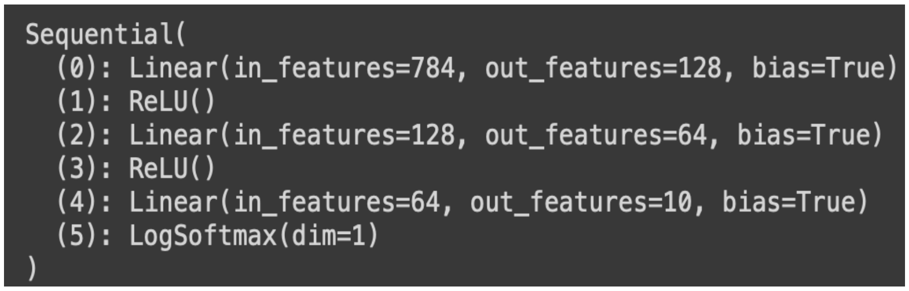
- **The model performance improvement from NN (Neural Network) to CNN (Convolutional Neural Network)**

1. The convolution units are especially beneficial as:

➢ They reduce the number of units in the network (since they are many-to-one mappings). This means, there are fewer parameters to learn which reduces the chance of overfitting as the model would be less complex than a fully connected network.

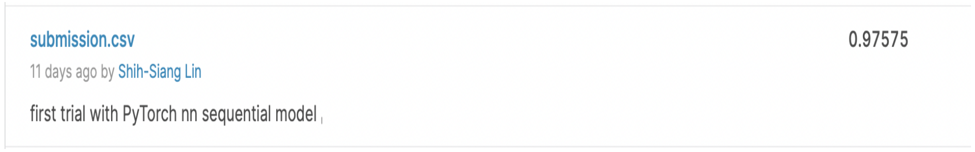
➢ They consider the context/shared information in the small neighborhoods. This future is very important in many applications such as image, video, text, and speech processing/mining as the neighboring inputs (e.g. pixels, frames, words, etc) usually carry related information.

2. The structure and results using the nn sequential model



```
Sequential(
  (0): Linear(in_features=784, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=10, bias=True)
  (5): LogSoftmax(dim=1)
)
```

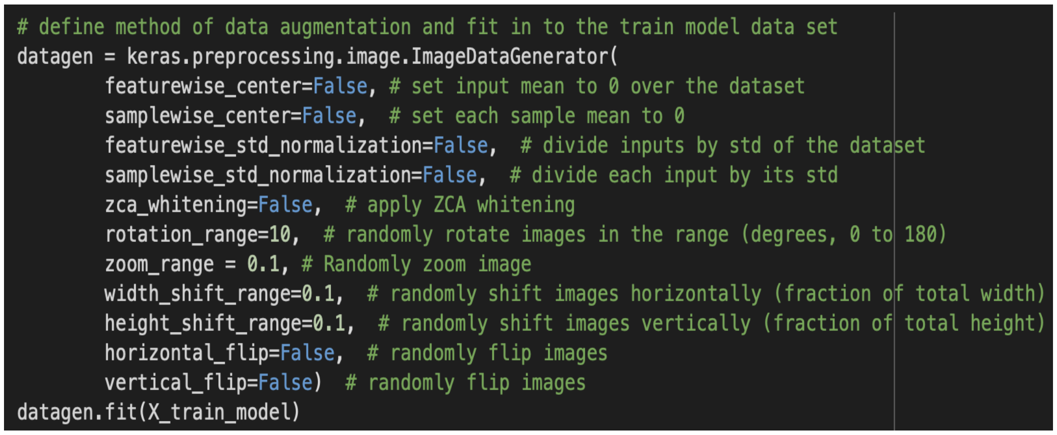
Below is the score on the Kaggle using Pytorch nn sequential model:



submission.csv 0.97575
11 days ago by Shih-Siang Lin
first trial with PyTorch nn sequential model

The result was the first trial using the nn sequential model, and we found that the performance of ensembling five CNN models was way better than the nn sequential model from the scores on Kaggle competition.

- **Data Augmentation on Image**



```
# define method of data augmentation and fit in to the train model data set
datagen = keras.preprocessing.image.ImageDataGenerator(
        featurewise_center=False, # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
datagen.fit(X_train_model)
```

As we have mentioned above, model trained with the full data set can get higher accuracy on the test data set, meaning that with more data, model can perform better. In the real world scenario, we may have a dataset of images taken in a limited set of conditions. But our target application may exist in a variety of conditions, such as different orientation, location, scale, brightness etc. We account for these situations by training our neural network with **additional synthetically modified data**. This is when the technique of Data Augmentation does the magic. By performing augmentation, we can make minor alterations to our existing dataset, and our neural network would think these as distinct images, thus essentially boosting overall performance. The augmentation skills we applied were: Flip, Rotation, Shift, adjust the center mean and normalize it with the standard deviation, and ZCA whitening. With these adjustments, our performance did improve on the test data sets. In brief, Data Augmentation can help model identify more aspects of images with limited training data since natural data in real world can still exist in a variety of conditions.

IV. References

https://www.kaggle.com/competitions/digit-recognizer/discussion/61480

https://ai.stackexchange.com/questions/5546/what-is-the-difference-between-a-convolutional-neural-network-and-a-regular-neur

https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

https://www.kaggle.com/code/chingiznurzhanov/digits-keras-cnn-acc-0-99564

HW2 and HW3 from Big Data and Artificial Intelligence for Business (BUDT737)