Two Initial Methods in Convolutional neural network

Li Yuan, Advisor: Professor Roman Kuc

July 25, 2021

1 Abstract

The main aim of this research project is to propose two initialized methods to improve current convolution neural network. I will address this improvement from two prospectives. One is to come up with a new method in image data augmentation. The second one is redesign current median(mode) committee vote method. I will use MNIST hand written digit number database to experiment.

Keywords: image data augmentation, committee vote, MNIST database, convolution neural network

2 Backgound and Introduction

We will initialize two main methods appearing in Neural Network.

1. The first one is a new method in Image Data Augmentation.
2. The second one is a new method in redesigning Committee Vote.

Currently, most image data augmentation techniques are only using one input to shift, rotate, add white noise. However, my new method will incorporate random multiple inputs to extract their common obvious features by committee votes based on each class. Our method performs well on small size data.

The present mature method of committee vote is to choose mode or median as the final decision, but those methods don't consider probability and weights. However, I will come up with a method which make use of probability distribution and appearing frequency as weights to redesign my new committee vote.

2.1 Introduce our Python3.8 environment to reproduce my results[2]

[56]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from keras.datasets import mnist

from keras.preprocessing.image import ImageDataGenerator
import os

import sklearn
```

```
from sklearn.utils import shuffle

from tensorflow import keras
from tensorflow.keras import layers

import copy
%config InlineBackend.figure_format = 'pdf'

from scipy import stats

import warnings
warnings.filterwarnings('ignore')
```

3 The current general image augmentation method review

3.1 Load Digit Written MNIST database as our new method experimental data

[25]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

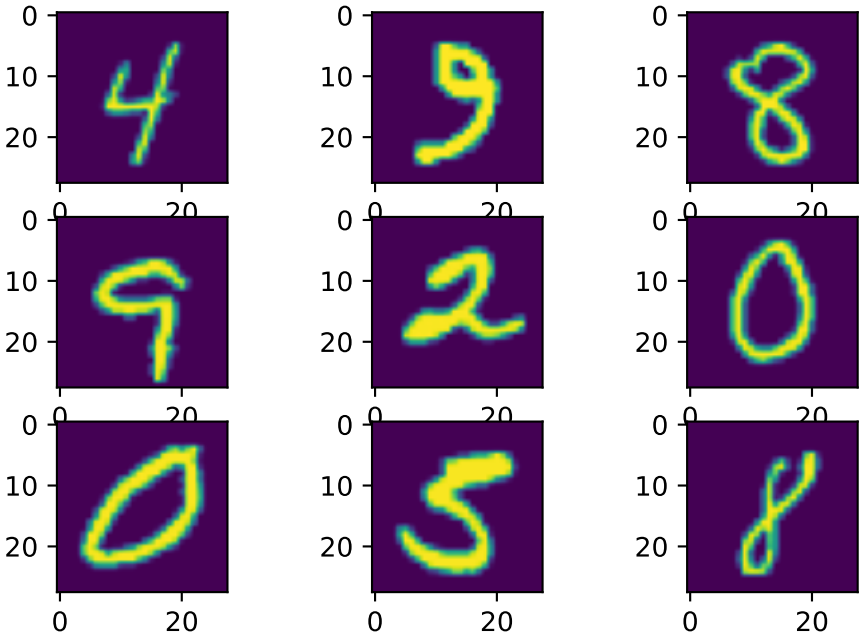
3.2 reshape to be [samples][width][height][channels] and change to float type

[26]:
```
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_train = X_train.astype('float32')
# Set seed
np.random.seed(12345)
```

3.3 Feature Standardization

This method will centralize and standardize the data. The raw data sometimes will benefit from this manipulation because of some outliers and different scales.

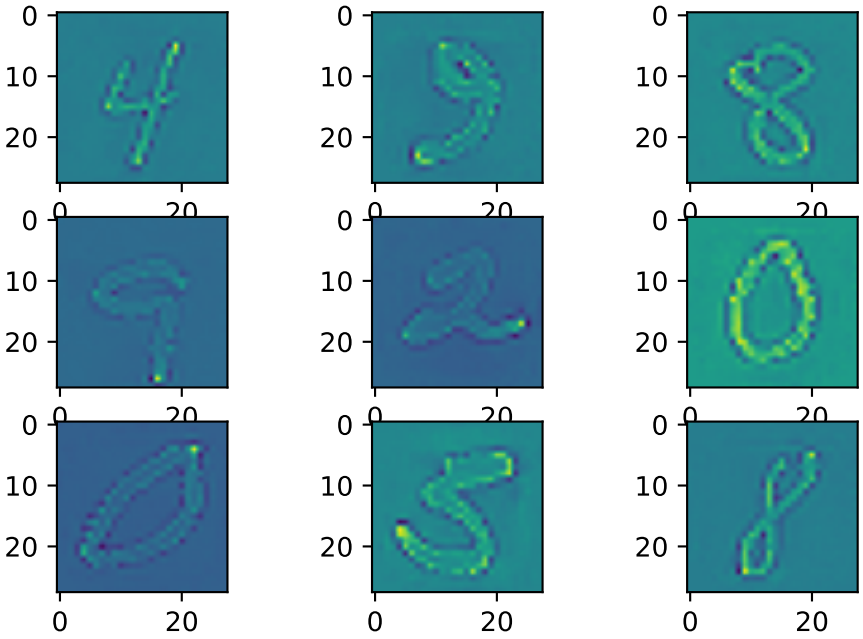
[27]:
```
# define data preparation
datagen = ImageDataGenerator(featurewise_center=True,␣
 ↪featurewise_std_normalization=True)
# fit parameters from data
datagen.fit(X_train)
# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9, seed=123):
    # create a grid of 3x3 images
    for i in range(0, 9):
        plt.subplot(330 + 1 + i)
        plt.imshow(X_batch[i].reshape(28, 28))
    # show the plot
    plt.show()
    break
```



3.4 ZCA Whitening

This technique is used to linearly transform raw data such that covariance matrix of it is identical and make sure to decrease the correlation in data.

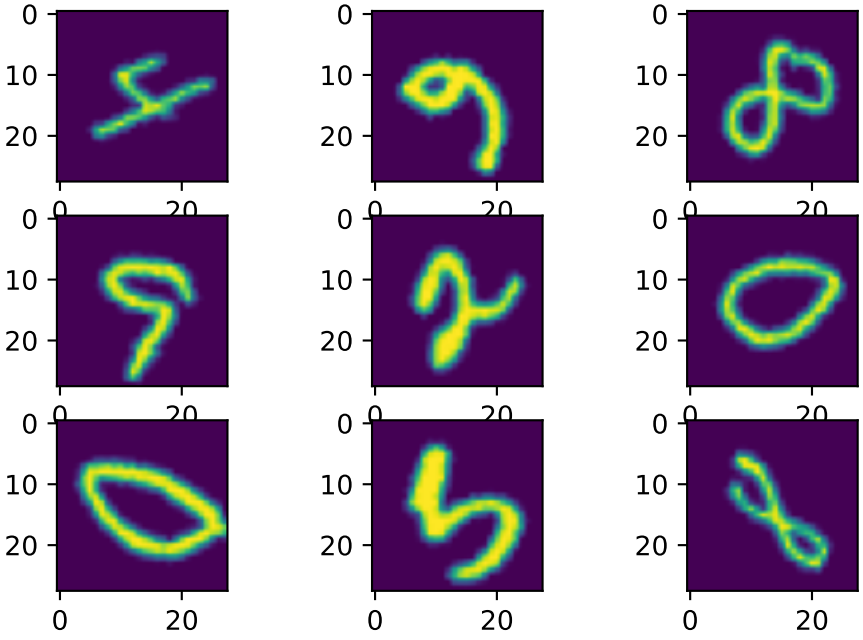
[28]:
```
datagen = ImageDataGenerator(zca_whitening=True)
datagen.fit(X_train)
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9, seed=123):
    for i in range(0, 9):
        plt.subplot(330 + 1 + i)
        plt.imshow(X_batch[i].reshape(28, 28))
    plt.show()
    break
```



3.5 Random Rotations

This method is very straightforward. Some of images are randomly chosen to rotate 90 degree.

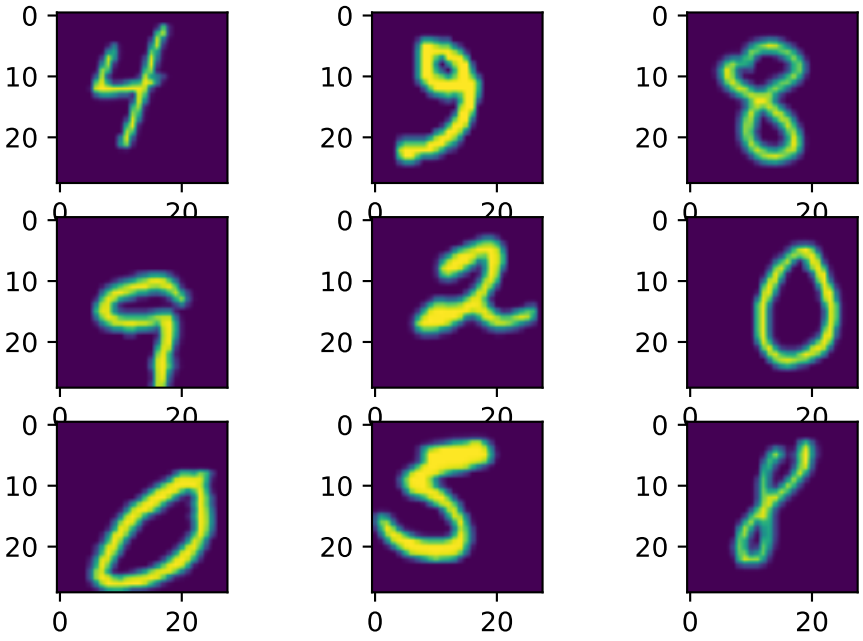
[29]:
```
datagen = ImageDataGenerator(rotation_range=90)
```



3.6 Random Shifts

This method is also very straightforward, some images are randomly shifted vertically or horizontally by different values.

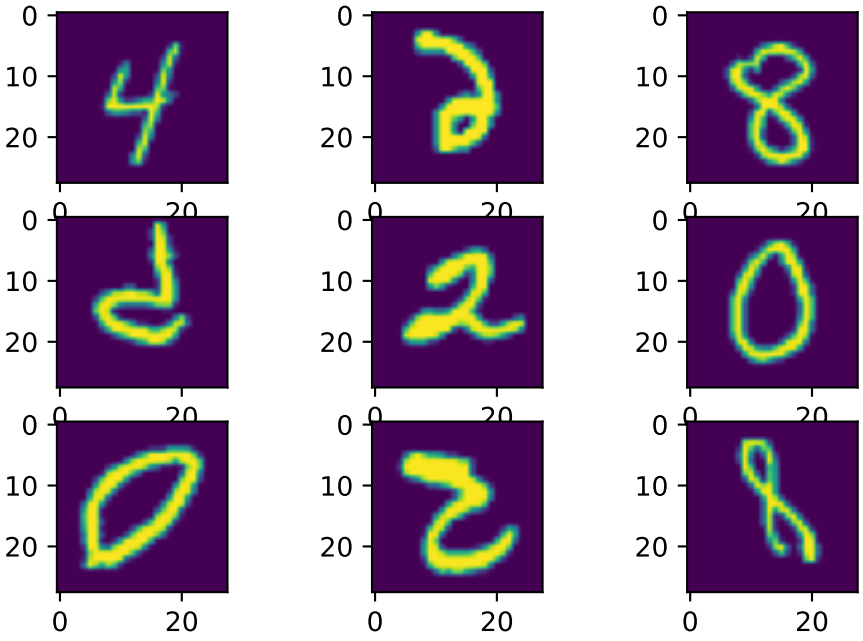
[31]:
```
# define data preparation
shift = 0.2
datagen = ImageDataGenerator(width_shift_range=shift, height_shift_range=shift)
```



3.7 Random Flips

This method is just to flip the random images and keep the remaining the same.

[33]:
```
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)
```



3.8 My new method

All above proposed methods only use one input image. The drawback of this method is that one new image can't learn common characteristics of this class due to one picture. I want to leverage the idea of the committee vote to select multiple input images to generate new images. This committee vote can learn the obvious pattern of this class and spit out this pattern in new image. I will step by step to illustrate my method.

3.8.1 Set a parameter S, size of committee vote, then randomly select S images from the class(instead of the whole data) we want to generate

[ ]:
```
cla = dataset
np.random.shuffle(cla) # shuffle the class
sub_index = np.random.choice(cla.shape[0], S, False) # random subset S data
basis = cla[sub_index, :, :, :] # form the basis
```

3.8.2 Iterate each pixel of image shape, for example we use (28, 28, 1)

[ ]:
```
for i in range(28):
    for j in range(28):
        count = []
```

3.8.3 Iterate each image on the selected pixel in this committee vote

[ ]:
```
S = 9
for q in range(S):
```

3.8.4 Set a parameter thre, threshold of each pixel, then judge if the pixel value of each image is greater than this threshold

[ ]:
```
if basis[q, i, j, 0] >= thre:
    count.append(basis[q, i, j, 0])
```

3.8.5 Set a parameter p, percentage of committee vote needed to calculate the average of values on this pixel

[ ]:
```
if len(count) >= S * p:
    new[i, j, 0] = np.mean(count)
```

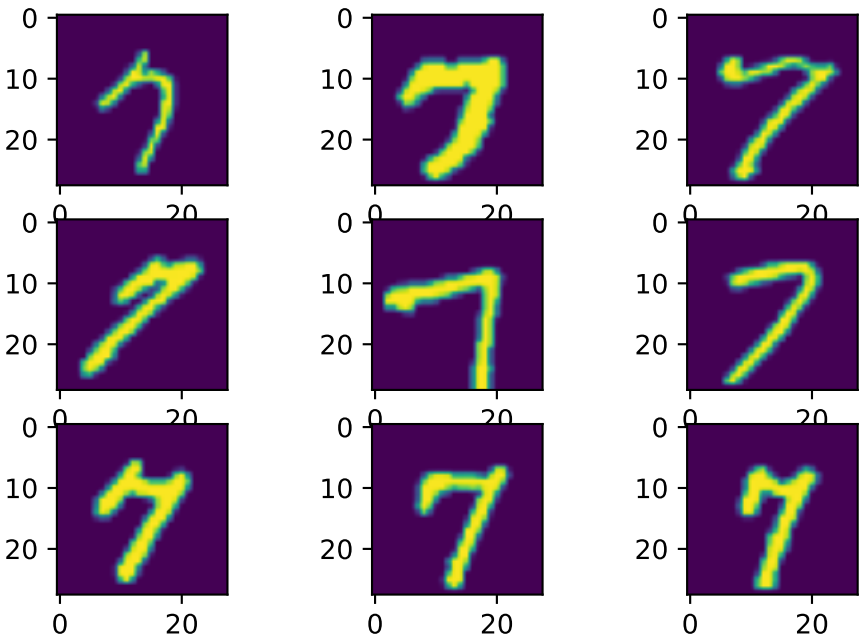
3.9 After these steps, we will combine them together to define a function to process a new image based on multiple inputs

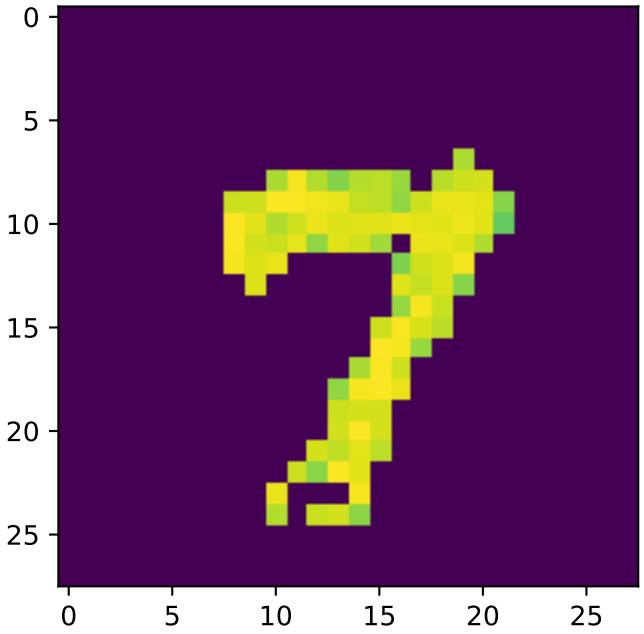
[35]:
```
def generator(dataset, num = 0, S = 50, p = 0.25, thre = 140):
    '''
    by tuning paramaters we generate one new digit image
    '''
    cla = dataset
    np.random.shuffle(cla) # shuffle the class
    sub_index = np.random.choice(cla.shape[0], S, False) # random subset S data
    basis = cla[sub_index, :, :, :] # form the basis

    new = np.zeros_like(basis[1])
    for i in range(new.shape[0]):
        for j in range(new.shape[1]):
            count = []
            for q in range(S):
                if basis[q, i, j, 0] >= thre:
                    count.append(basis[q, i, j, 0])
            if len(count) >= S * p:
                new[i, j, 0] = np.mean(count)
    return new
```

3.10 Next we will use the variant of above function to tune suitable parameters and compare new generated image with committee vote images

3.10.1 For example, we used 7 as our first trial number, 9 as our committee size, 40% as our percentage to pass, 150 as each pixel threshold value

[39]:
```
generator1(num = 7, S = 9, p = 0.4, thre = 150)
```

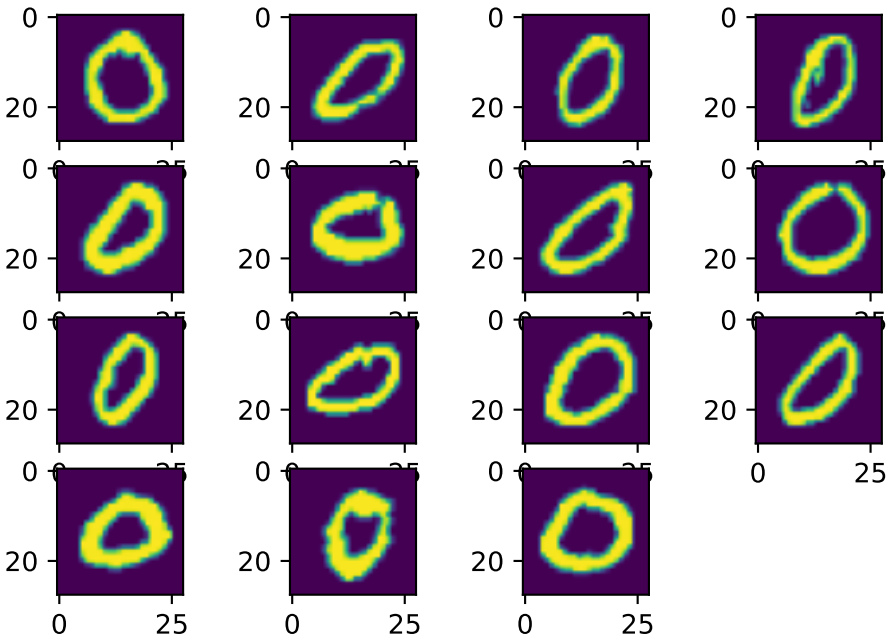


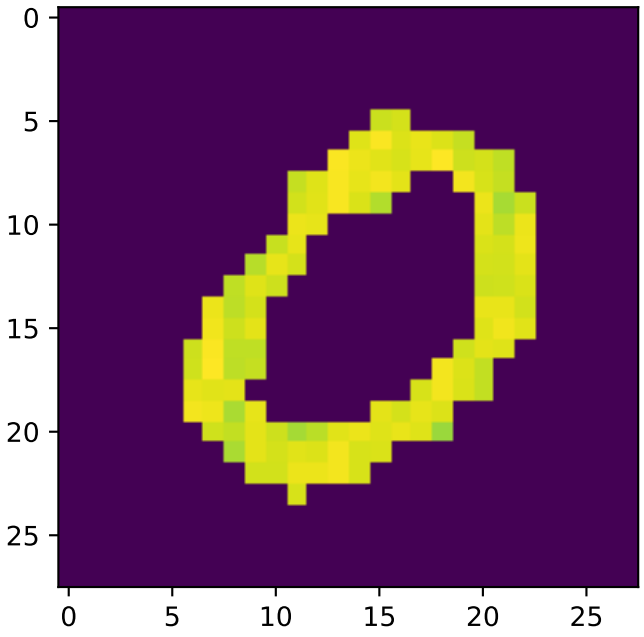


As we see from number = 7, the generated new image capture most common part of digit 7 in this hand-written 9 images. If you want to get better results, you can tune more parameters to find the best combination of it. These three parameters, *S*, *p*, *thre*, are well represented the most features we can manually tune to generate an artificial image based on what we had.

3.10.2 Now we change our number class to 0 and S = 15, p = 0.6, thre = 170

[41]:
```
generator1(num = 0, S = 15, p = 0.6, thre = 140)
```





This 0 image performs better than the previous one 7, which means we find a better combination of parameters to capture its characteristic. Actually, for some classes, the more complex the class is, the harder new image can capture.

3.11 Compare the performance of this method with original raw data

3.11.1 Create a small subset data from MNIST

3.11.2 Split data based on each class from 0 thru 9

[42]:
```
# the data, split between train and test sets
(X_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
X_train = np.expand_dims(X_train, -1)
x_test = np.expand_dims(x_test, -1)

x0 = X_train[y_train == 0]
x1 = X_train[y_train == 1]
x2 = X_train[y_train == 2]
x3 = X_train[y_train == 3]
x4 = X_train[y_train == 4]
x5 = X_train[y_train == 5]
x6 = X_train[y_train == 6]
x7 = X_train[y_train == 7]
```

```
x8 = X_train[y_train == 8]
x9 = X_train[y_train == 9]
```

3.11.3 Randomly choose a small subset of each class from 0 thru 9

[44]:
```
p = 50
np.random.seed(123)
x0_ind = np.random.choice(x0.shape[0], x0.shape[0]//p, False)
x1_ind = np.random.choice(x1.shape[0], x1.shape[0]//p, False)
x2_ind = np.random.choice(x2.shape[0], x2.shape[0]//p, False)
x3_ind = np.random.choice(x3.shape[0], x3.shape[0]//p, False)
x4_ind = np.random.choice(x4.shape[0], x4.shape[0]//p, False)
x5_ind = np.random.choice(x5.shape[0], x5.shape[0]//p, False)
x6_ind = np.random.choice(x6.shape[0], x6.shape[0]//p, False)
x7_ind = np.random.choice(x7.shape[0], x7.shape[0]//p, False)
x8_ind = np.random.choice(x8.shape[0], x8.shape[0]//p, False)
x9_ind = np.random.choice(x9.shape[0], x9.shape[0]//p, False)

x0_train = x0[x0_ind, :, :, :]
x1_train = x1[x1_ind, :, :, :]
x2_train = x2[x2_ind, :, :, :]
x3_train = x3[x3_ind, :, :, :]
x4_train = x4[x4_ind, :, :, :]
x5_train = x5[x5_ind, :, :, :]
x6_train = x6[x6_ind, :, :, :]
x7_train = x7[x7_ind, :, :, :]
x8_train = x8[x8_ind, :, :, :]
x9_train = x9[x9_ind, :, :, :]
```

3.11.4 Combine each small subset class and shuffle them with their labels

[45]:
```
new_train = np.concatenate([x0_train, x1_train, x2_train, x3_train, x4_train,␣
↪x5_train, x6_train, x7_train, x8_train, x9_train], axis = 0)

new_y_train = np.array([0] * x0_train.shape[0] +
[1] * x1_train.shape[0] +
[2] * x2_train.shape[0] +
[3] * x3_train.shape[0] +
[4] * x4_train.shape[0] +
[5] * x5_train.shape[0] +
[6] * x6_train.shape[0] +
[7] * x7_train.shape[0] +
[8] * x8_train.shape[0] +
[9] * x9_train.shape[0])

train, y = sklearn.utils.shuffle(new_train, new_y_train, random_state=0)
```

3.11.5 Define Convolutional neural network model function[1]

[46]:
```
def CNNfit(generated_train, generated_y):
    '''
    train the model
    '''
    # Scale images to the [0, 1] range
    generated_train = generated_train.astype("float32") / 255

    # convert class vectors to binary class matrices
    generated_y = keras.utils.to_categorical(generated_y, 10)

    model = keras.Sequential(
        [
            keras.Input(shape=input_shape),
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(10, activation="softmax"),
        ]
        )

    batch_size = 128
    epochs = 7

    model.compile(loss="categorical_crossentropy", optimizer="adam",␣
↪metrics=["accuracy"])

    history = model.fit(generated_train, generated_y, batch_size=batch_size,␣
↪epochs=epochs,
    validation_split=0.1, verbose = 0)

    return model, history
```

3.11.6 Define evaluation function and calculate PEG value

[47]:
```
def CNNevaluate(model, Xtest, ytest):  # computes CM and PE for test set
    Ntest = Xtest.shape[0] # number of rows
    CM = np.zeros([10,10], dtype = int)
    ypred = model.predict(Xtest, verbose = 1) # predicts entire set
    for i in range(Ntest):
        yclass = np.argmax(ypred[i])
        ytrue = int(ytest[i])
        CM[ytrue,yclass] += 1
```

```
    Nerr = sum(sum(CM))-np.trace(CM)
    Ntotal = sum(sum(CM))
    PE = Nerr/Ntotal
    return Nerr, Ntotal,CM, PE
```

3.11.7 Define a function that creates new images using my method

[48]:
```
def create(dataset, mul = 3, num = 0, S = 50, p = 0.25, thre = 140):
    '''
    create a whole new data set
    '''
    x_new = copy.deepcopy(dataset)
    for i in range(x_new.shape[0] * mul):
        new = generator(x_new, num = num, S = S, p = p, thre = thre)
        x_new = np.append(x_new, np.expand_dims(new, axis=0) , axis = 0)
    return x_new
```

3.11.8 Display each class size from 0 thru 9 after my image augmentation method

```
(826, 28, 28, 1)
(938, 28, 28, 1)
(833, 28, 28, 1)
(854, 28, 28, 1)
(812, 28, 28, 1)
(756, 28, 28, 1)
(826, 28, 28, 1)
(875, 28, 28, 1)
(819, 28, 28, 1)
(826, 28, 28, 1)
```

3.11.9 Display the raw small subset size and generated big data size after my image augmentation method

[50]:
```
print(train.shape, y.shape)
print(generated_train.shape, generated_y.shape)
```

```
(1195, 28, 28, 1) (1195,)
(8365, 28, 28, 1) (8365,)
```

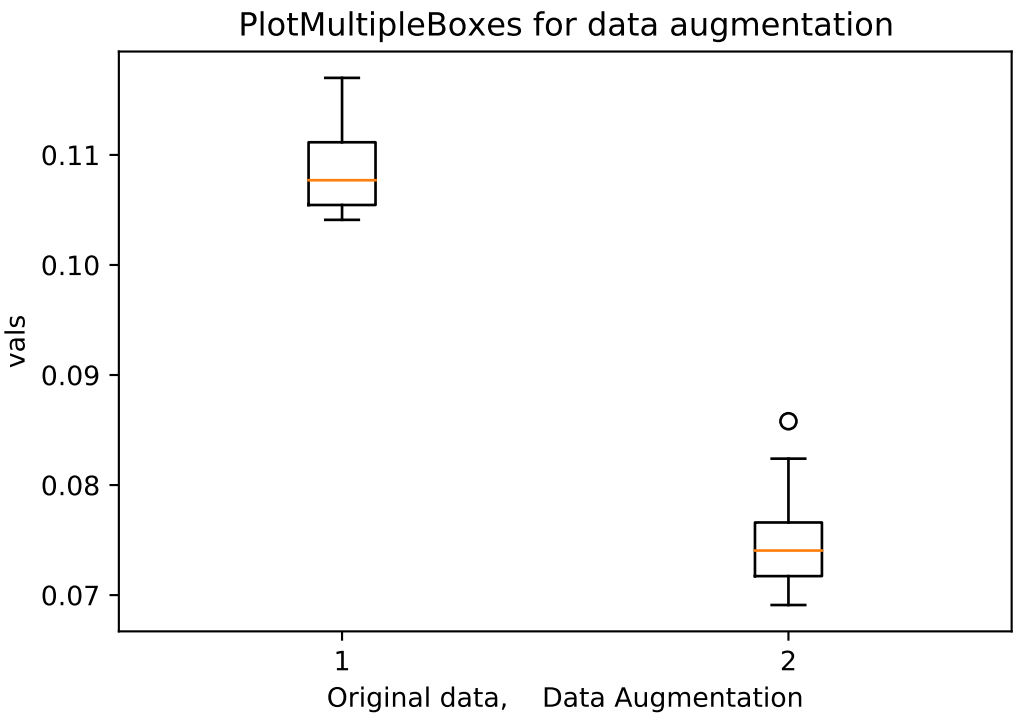
3.11.10 Now we use 10 restarts to compare the prediction PEG on raw small subset versus expanded bigger data

[ ]:
```
Nrestarts = 10
num_classes = 10
input_shape = (28, 28, 1)
PEG_vals_a = np.zeros(Nrestarts)
```

```
PEG_vals_o = np.zeros(Nrestarts)

# the data, split between train and test sets
(X_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
# Scale images to the [0, 1] range
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_test = np.expand_dims(x_test, -1)

for restart in range(Nrestarts):

    model_a, history_a = CNNfit(generated_train, generated_y)
    model_o, history_o = CNNfit(train, y)

    Nerr_a, Ntotal_a,CM_a, PEG_a = CNNevaluate(model_a,x_test,y_test)
    Nerr_o, Ntotal_o,CM_o, PEG_o = CNNevaluate(model_o,x_test,y_test)

    PEG_vals_a[restart] = PEG_a
    PEG_vals_o[restart] = PEG_o

boxes = [PEG_vals_o, PEG_vals_a]
```

3.11.11 Let's compare their box plots in one figure

[52]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.boxplot(boxes)
ax.set_title('PlotMultipleBoxes for data augmentation')
ax.set_xlabel('Original data,    Data Augmentation')
ax.set_ylabel('vals')
plt.show()
```



As you can see the expanded image data performance much better than raw small data when using the same model setting. I used the very standard model, 2 convolutional layers plus two max pooling layers, then a dropout. These model set is thought as the excellent model on minist data. The median PEG of raw data is about 0.108 whereas PEG of my expanded data is about 0.074. My method improved about 31%. What's more, in this example, even outlier of the expanded data is lower than the low whisker bar of raw data. In light of this analysis, if you find a good combination parameters for most class regardless of this minist data, your image augmentation expanded data will help a lot.

3.11.12 The advantages of our new data augmentation

1. Use random multiple original inputs to create one new data (stochastic)
2. Create new data based on each category instead of the whole new dataset to improve feature extraction (classify)
3. New data can be randomly selected to create another new data (self-adaptive)
4. Focus on feature extraction to create new data (committee vote)
5. Give multiple tune parameters to find best new data (tune parameter)

3.11.13 The disadvantaged of our new data augmentation

1. Separating into each class data requires more manipulation
2. Tune different parameters on each class

4 The second method is how to redesign committee vote to have a better final decision.

I was considering that frequency of mode candidate is greater than some predefined threshold, for instance, 7 out of 10, we are confident enough to choose this mode candidate to be our final decision, otherwise, for example mode 5 out of 10, we don't have confidence in this model candidate. We are supposed to redesign our committee vote method. I noticed that the current methods don't consider frequency weights and probability distribution when deciding the final candidate. So, as you will see next, I integrate the two new factors into my new method.

4.1 If the mode vote is greater than number of model * (tune parameter p), then we choose this vote as our prediction for each new test point.

4.2 If the mode vote is less than number of model * (tune parameter p), then we use below method to choose vote. I will step by step go through this.

4.2.1 As an example, we randomly generate 10×10 ranged between 0 and 1 probability 2D matrix, simulating 10 different models giving 10 probability from 0 thru 9.

[54]:
```
np.random.seed(123)
a = np.random.rand(10,10)
a
```

[54]:
```
array([[0.69646919, 0.28613933, 0.22685145, 0.55131477, 0.71946897,
        0.42310646, 0.9807642 , 0.68482974, 0.4809319 , 0.39211752],
       [0.34317802, 0.72904971, 0.43857224, 0.0596779 , 0.39804426,
        0.73799541, 0.18249173, 0.17545176, 0.53155137, 0.53182759],
       [0.63440096, 0.84943179, 0.72445532, 0.61102351, 0.72244338,
        0.32295891, 0.36178866, 0.22826323, 0.29371405, 0.63097612],
       [0.09210494, 0.43370117, 0.43086276, 0.4936851 , 0.42583029,
        0.31226122, 0.42635131, 0.89338916, 0.94416002, 0.50183668],
       [0.62395295, 0.1156184 , 0.31728548, 0.41482621, 0.86630916,
        0.25045537, 0.48303426, 0.98555979, 0.51948512, 0.61289453],
       [0.12062867, 0.8263408 , 0.60306013, 0.54506801, 0.34276383,
        0.30412079, 0.41702221, 0.68130077, 0.87545684, 0.51042234],
       [0.66931378, 0.58593655, 0.6249035 , 0.67468905, 0.84234244,
        0.08319499, 0.76368284, 0.24366637, 0.19422296, 0.57245696],
       [0.09571252, 0.88532683, 0.62724897, 0.72341636, 0.01612921,
        0.59443188, 0.55678519, 0.15895964, 0.15307052, 0.69552953],
       [0.31876643, 0.6919703 , 0.55438325, 0.38895057, 0.92513249,
        0.84167   , 0.35739757, 0.04359146, 0.30476807, 0.39818568],
       [0.70495883, 0.99535848, 0.35591487, 0.76254781, 0.59317692,
        0.6917018 , 0.15112745, 0.39887629, 0.2408559 , 0.34345601]])
```

4.2.2 Vote Results and probability for each voted digit number then compute mean of lists of highest probability given each number

[58]:
```
print("Vote results:", vote, "\n")
pri_avg(ind_new)
```

```
Vote results: [6. 5. 1. 8. 7. 8. 4. 1. 4. 1.]


 1 :  [0.8494317940777896, 0.8853268262751396, 0.9953584820340174] / 3 =
0.9100390341289822

 4 :  [0.8423424376202573, 0.9251324896139861] / 2 = 0.8837374636171217

 5 :  [0.7379954057320357] / 1 = 0.7379954057320357

 6 :  [0.9807641983846155] / 1 = 0.9807641983846155

 7 :  [0.985559785610705] / 1 = 0.9855597856107050

 8 :  [0.9441600182038796, 0.8754568417951749] / 2 = 0.9098084299995273
```

4.2.3 Compute average probability of each digit number multiplied by their frequency weights

[60]:
```
pri_mul(ind_mean, ind_num)
```

```
 1 :  0.9100390341289822 * (3 / 10) = 0.2730117102386947

 4 :  0.8837374636171217 * (2 / 10) = 0.1767474927234243

 5 :  0.7379954057320357 * (1 / 10) = 0.0737995405732036

 6 :  0.9807641983846155 * (1 / 10) = 0.0980764198384615

 7 :  0.9855597856107050 * (1 / 10) = 0.0985559785610705

 8 :  0.9098084299995273 * (2 / 10) = 0.1819616859999054
```

4.3 Define the whole function consisting of all above processes

[61]:
```
def committe_vote(p, **kwargs):
    '''
    redefine a new committe vote modality
    '''
    con = []
    for key, value in kwargs.items():
```

```
        value = np.expand_dims(value, 0)
        con.append(value)
    new = np.concatenate(con, axis = 0)
    num_test = new.shape[1]
    num_model = new.shape[0]
    num_class = new.shape[2]
    pred = np.empty(shape=num_test)

    for i in range(num_test):
        ma = new[:, i, :]
        vote = np.empty(num_model)
        ind_pro = {0: [], 1: [],2: [],3: [],4: [],5: [],6: [],7: [],8: [],9: []}
        for j in np.arange(num_model):
            vote[j] = np.argmax(ma[j, :])
            ind_pro.get(vote[j]).append(np.amax(ma[j, :]))
        m, n = stats.mode(vote, axis=None)
        if n >= num_model * p:
            pred[i] = m
        else:
            ind_new = {}
            for key, value in ind_pro.items():
                if value != []:
                    ind_new[key] = value
            ind_mean = {}
            ind_num = {}
            for key, value in ind_new.items():
                ind_mean[key] = np.mean(value)
                ind_num[key] = len(value)
            ind_final = {}
            for key in ind_new:
                ind_final[key] = ind_mean[key] * ind_num[key] / num_model
            need = max(ind_final.values())
            for key, value in ind_final.items():
                if value == need:
                    pred[i] = key
    return pred
```

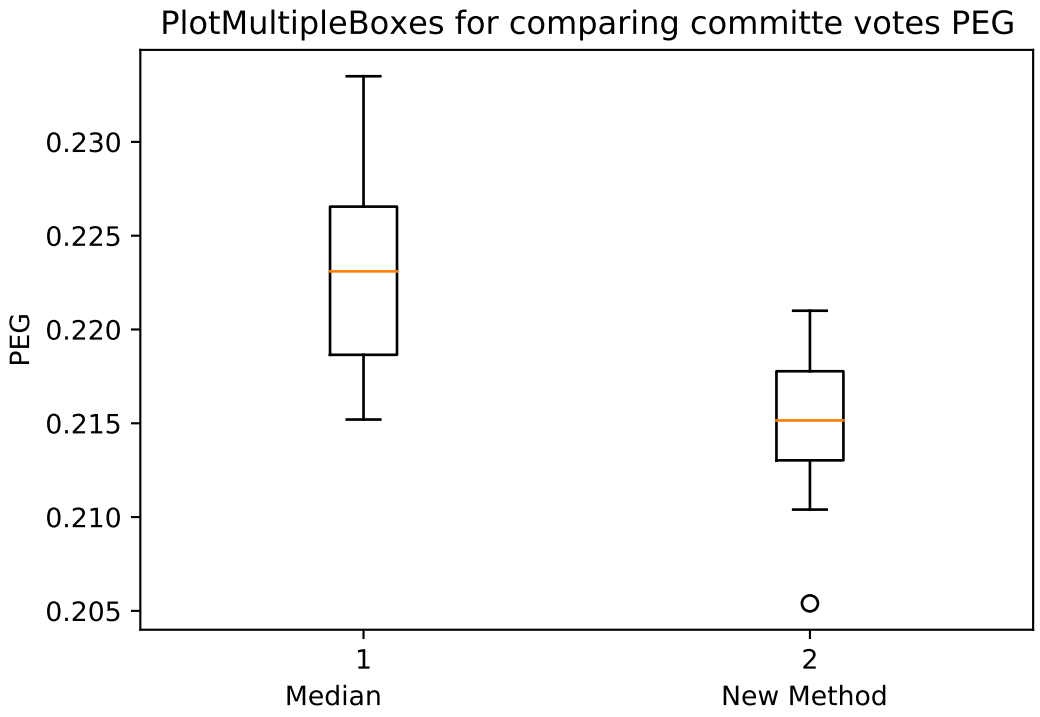
4.4 Use 10 restarts and 70% percentage as committee pass rate to compare median method with my new method

```
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
```

```
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
 64/313 [=====>…] - ETA: 0
```

4.4.1 Draw a box plot to directly compare them

[69]:
```
boxes2 = [PEG_median, PEG_pro_wei]
fig = plt.figure()
ax = fig.add_subplot(111)
ax.boxplot(boxes2)
ax.set_title('PlotMultipleBoxes for comparing committe votes PEG')
ax.set_xlabel("Median                                                New Method")
ax.set_ylabel('PEG')
plt.show()
```



4.5 Conclusion

As you can from this box plot, our new method improved somehow compared to median method(mode). The median PEG of median method is about 0.224 while median PEG of my new method is about 0.215. My method improved about 4% in general and average, which means when we consider more information to make decisions, we can perform well on predicting unknown thing. This philosophy applies to other things in the world. Frequency weights and probability distributions are two important factors when making decisions.

By doing this research project, I have learned how to design our initial algorithm to improve current algorithms and think to push the boundary of current research work. By conducting a self-paced research, it encourages me to think independently and learn new things. Use boxes plot or other methods to test our initial methods in general instead of one lucky test.

References

[1] François Chollet, *Keras*, Github, GitHub repository, https://github.com/fchollet/keras, **2015**.

[2] Jupyter Development Team, *Jupyter Notebooks – a publishing format for reproducible computational workflows*, IOS Press, 87-90, **2016**, https://jupyter.org/, https://nbconvert.readthedocs.io/en/latest/index.html.

[3] Arun Gandhi, *Data Augmentation, How to use Deep Learning when you have Limited Data—Part 2*, Nanonets Automate Data Capture, **2021**, https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.

[4] Jason Brownlee, *How to Configure Image Data Augmentation in Keras*, machine learning mastery, **2019**, https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/