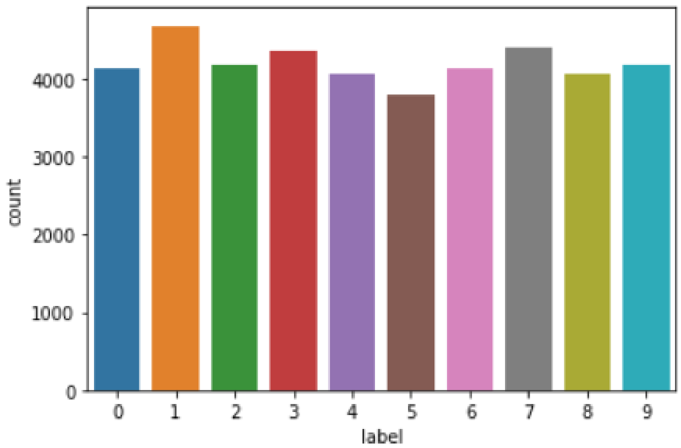
# MNIST data deep learning

Tao Shan

**Describe the data**

The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.




The label has 10 numbers from 0-9, each number's count is represented in this count plot, which shows the number of labels for each number is not sparse.

## Objective

This model aims to find the relationship between the target numbers and the pixel points in the photo. I'm using the accuracy score to estimate the model performance.

## Steps

The following steps I did for preprocessing the data:
**Scaler:** Min-max scaler between 0 and 1 for pixel point's values
**One-hot encoding:** use it for y values to fit with the outputs of deep learning models in python Keras package
**Split train/validation sets:** 80% data for training, 20% data for validation

## Models

The models I have chosen are self-designed CNN models (2 different structures) and one Vgg16 transfer learning model. All models use categorical_crossentropy for loss function, accuracy metrics and adam optimizer.

Model 1: CNN

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 25, 25, 32) | 544 |
| conv2d_1 (Conv2D) | (None, 22, 22, 32) | 16416 |
| max_pooling2d (MaxPooling2D) | (None, 11, 11, 32) | 0 |
| dropout (Dropout) | (None, 11, 11, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 8, 8, 32) | 16416 |
| conv2d_3 (Conv2D) | (None, 5, 5, 32) | 16416 |
| max_pooling2d_1 (MaxPooling2 | (None, 2, 2, 32) | 0 |
| dropout_1 (Dropout) | (None, 2, 2, 32) | 0 |
| flatten (Flatten) | (None, 128) | 0 |
| dropout_2 (Dropout) | (None, 128) | 0 |
| dense (Dense) | (None, 256) | 33024 |
| dropout_3 (Dropout) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 10) | 2570 |

Total params: 85,386
Trainable params: 85,386
Non-trainable params: 0

I use this as a base line approach. Within an epoch of 100, the best accuracy for validation set is 0.994 when epoch = 94.

Model 2: CNN

| Layer (type) | Output Shape | Param # |
|---|---|---|
| zero_padding2d (ZeroPadding2 | (None, 30, 30, 1) | 0 |
| conv2d_4 (Conv2D) | (None, 26, 26, 5) | 130 |
| conv2d_5 (Conv2D) | (None, 22, 22, 5) | 630 |
| max_pooling2d_2 (MaxPooling2 | (None, 11, 11, 5) | 0 |
| batch_normalization (BatchNo | (None, 11, 11, 5) | 20 |
| zero_padding2d_1 (ZeroPaddin | (None, 13, 13, 5) | 0 |
| conv2d_6 (Conv2D) | (None, 9, 9, 7) | 882 |
| conv2d_7 (Conv2D) | (None, 5, 5, 7) | 1232 |
| max_pooling2d_3 (MaxPooling2 | (None, 2, 2, 7) | 0 |
| batch_normalization_1 (Batch | (None, 2, 2, 7) | 28 |
| flatten_1 (Flatten) | (None, 28) | 0 |
| dense_2 (Dense) | (None, 49) | 1421 |
| dropout_4 (Dropout) | (None, 49) | 0 |
| dense_3 (Dense) | (None, 10) | 500 |

Total params: 4,843
Trainable params: 4,819
Non-trainable params: 24

By adding more layers, including zero padding 2d layers and batch normalization layers, the model learns more slowly and have a better accuracy when number of epochs grows. Within an epoch of 100, the best accuracy for validation set is 0.979 when epoch = 72.

Model 3: Vgg16

Layers for Vgg16:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_2 (InputLayer) | [(None, 56, 56, 3)] | 0 |
| block1_conv1 (Conv2D) | (None, 56, 56, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 56, 56, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 28, 28, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 28, 28, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 28, 28, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 14, 14, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 14, 14, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 14, 14, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 14, 14, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 7, 7, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 7, 7, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 7, 7, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 7, 7, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 3, 3, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 3, 3, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 3, 3, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 3, 3, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 1, 1, 512) | 0 |

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Layers for transfer learning:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| vgg16 (Functional) | (None, 1, 1, 512) | 14714688 |
| flatten_2 (Flatten) | (None, 512) | 0 |
| dense_4 (Dense) | (None, 512) | 262656 |
| activation (Activation) | (None, 512) | 0 |
| batch_normalization_2 (Batch | (None, 512) | 2048 |
| dropout_5 (Dropout) | (None, 512) | 0 |
| dense_5 (Dense) | (None, 10) | 5130 |
| activation_1 (Activation) | (None, 10) | 0 |

Total params: 14,984,522
Trainable params: 268,810
Non-trainable params: 14,715,712

Since total parameter is large, it requires GPU to train it faster. Also, I add Early Stopping to reduce the over fitting problem. Within an epoch of 30, the best accuracy for validation set is 0.968 when epoch = 29.

**Findings**

When number of epochs increases, the loss for each steps decreases, and the accuracy increases until the peak, then decreases a little bit. Previously I though the model should increase its accuracy when the model becomes more complex, but the solution is not. The base line version of CNN wins (with accuracy 0.993), and the other two versions lose. It's not necessary that more complex model has better accuracy.

**Revisit**

I can try more options such as Resnet, Alexnet, also Lenet-5 is based on this dataset, which I should also spend time on this. Also, I can try more structures on the existing models, and use auto encoder to generate more data.

**Appendix**

Data: https://www.kaggle.com/competitions/digit-recognizer

Code: https://www.kaggle.com/taos2000/mnist-cnn-vgg16-lenet-5