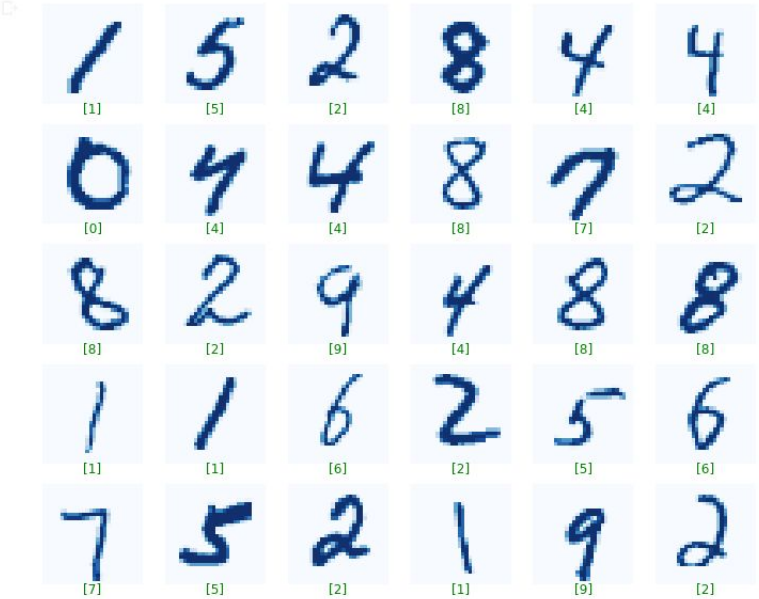
Kaggle - Digit Recognizer

김민규, 김민주, 윤희준

Project Proposal

Topic	Kaggle - Digit Recognizer
Description	MNIST DATA(0~9의 무작위 손글씨 이미지)에서 각 Data가 represent하는 Digit을 인식하는 Model을 생성
Expected Duration	4 weeks
Team Member	김민규, 김민주, 윤희준

What is MNIST DATA?



- **Modified National Institute of Standards and Technology**
- 손으로 쓴 숫자들로 이루어진 대형 데이터셋 **(약 60,000 샘플)**
- **28 x 28 픽셀; [0, 255] 픽셀 값어치; 숫자 0-9** 까지

데이터셋 확인

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel7
29633	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
345	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
36369	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
16624	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
14389	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

label → 0 - 9 숫자 값
pixel 0 - 784 → 28 x 28 픽셀의 값

```
df_train.shape
```

```
(42000, 785)
```

42,000개의 손글씨 이미지 (각 758 Column)

Training and Valid Dataset

```
#settings up validation set
sample_size = df_train.shape[0] # Original Training set size
validation_size = int(sample_size*0.2) # Validation set size
```

Training Set의 20%에 해당되는 Valid Set 지정

```
# train_x and train_y
train_x = np.asarray(df_train.iloc[:sample_size-validation_size,1:]).reshape([sample_size-validation_size,28,28,1])
train_y = np.asarray(df_train.iloc[:sample_size-validation_size,0]).reshape([sample_size-validation_size,1])
```

```
# val_x and val_y
val_x = np.asarray(df_train.iloc[sample_size-validation_size:,1:]).reshape([validation_size,28,28,1])
val_y = np.asarray(df_train.iloc[sample_size-validation_size:,0]).reshape([validation_size,1])
```

Train Set과 Valid Set을 Splice후 [size,28,28,1]의 형태로 Reshape
X = 이미지 자료들
Y = 각 이미지에 대한 label (답)

Test Dataset / Normalizing Pixel Values

```
df_test = pd.read_csv("input/digit-recognizer/test.csv")
test_x = np.asarray(df_test.iloc[:,:]).reshape([-1,28,28,1])
test_x.shape
```

```
(28000, 28, 28, 1)
```

```
# convirting pixel values in range [0,1]
train_x = train_x/255
val_x = val_x/255
test_x = test_x/255
```

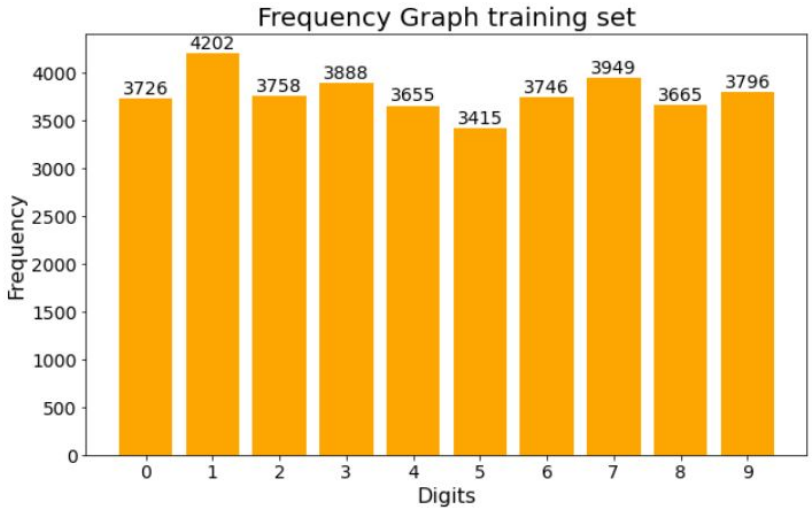
Test set 도 마찬가지로 [size,28,28,1]의 형태로 Reshape

픽셀값이 [0 - 255] 이면 모델이 학습하기 너무 어렵기 때문에 0 아님 1 로 표준화

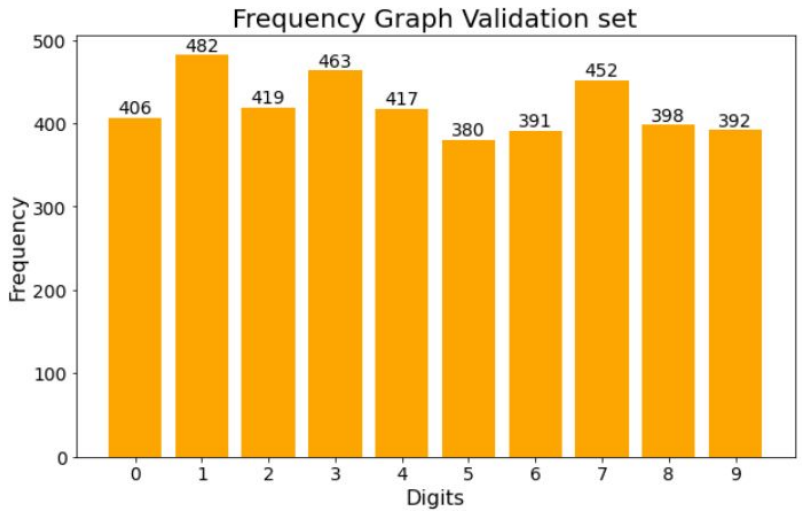
Visualize Digits Dataset

** To check is the frequency of the classes in the dataset

1. Frequency plot for the training



2. Frequency plot for the validation set



Dataset이 특정 숫자에 편향되는 경향이 없기 때문에 별도의 작업을 거치지않고 Dataset을 사용 가능

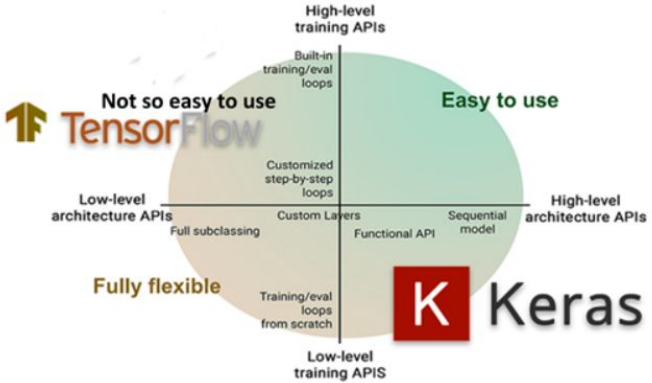
Building Model: Model Using Keras

Why Keras?

(+) Computation speed, Low-level Error
(-) Algorithm Support, Dynamic Charts support, API

Methods

- **Sequential Model**: Direct, Convenient, Simple
- **Function API**: To build a more complicated Model such as for multi-output Models




```
[ ]  model = models.Sequential()
```

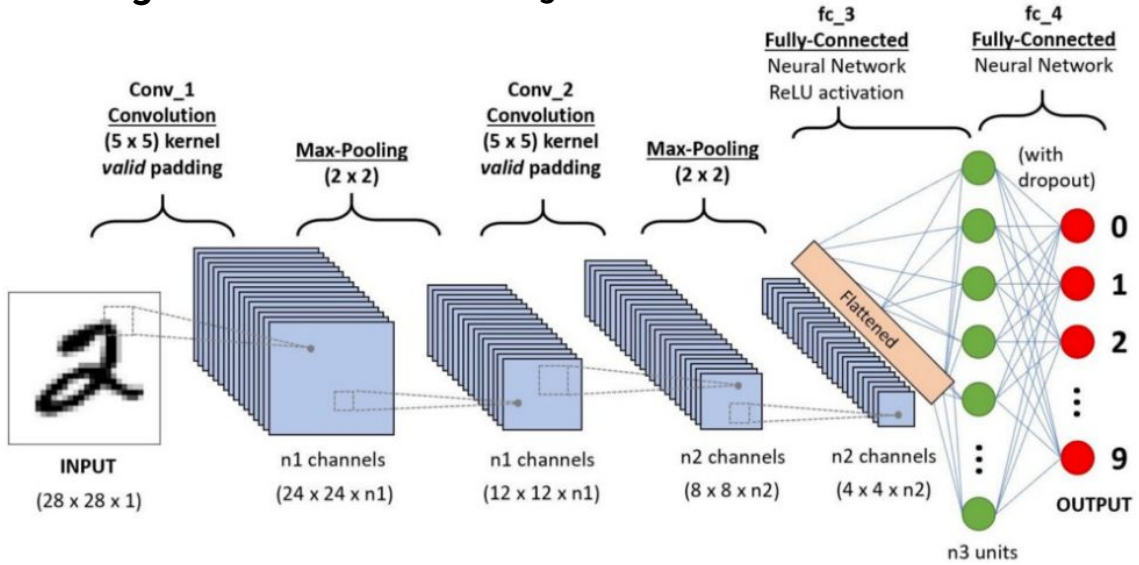
```
[ ]  # Block 1
     model.add(Conv2D(32,3, padding  ="same",input_shape=(28,28,1)))
     model.add(LeakyReLU())
     model.add(Conv2D(32,3, padding  ="same"))
     model.add(LeakyReLU())
     model.add(MaxPool2D(pool_size=(2,2)))
     model.add(Dropout(0.25))

     # Block 2
     model.add(Conv2D(64,3, padding  ="same"))
     model.add(LeakyReLU())
     model.add(Conv2D(64,3, padding  ="same"))
     model.add(LeakyReLU())
     model.add(MaxPool2D(pool_size=(2,2)))
     model.add(Dropout(0.25))

     model.add(Flatten())

     model.add(Dense(256,activation='relu'))
     model.add(Dense(32,activation='relu'))
     model.add(Dense(10,activation="sigmoid"))
```

Building Model: Model Using Keras



*** ReLU(rectified linear unit)**
: CNN 학습시 많이 사용,
but 0이상의 값은 그대로 출력해주지만
0이하의 값은 전달이 안됨. 이 부분을
개선하기 위해 활성화함수를 많이 사용함
ex. LeakyReLU

In our project
: Two Convolutional Blocks
Each block consists of 2 Conv2D layers with **LeakyReLU activation layers**.
-> **MaxPool2D layer**(to reduce the size of the image)
-> **Flatten layer ->** Dense Layers & Output layer
-> **Dropout Layer** (to drops the few activation nodes while regularization training)

Next Phase

- **Building Model**
  - Compiling Model
  - Training
  - Training Performance
  - Confusion Matrix
- **Improving Result by Image Augmentation**
- **Understand the Intermediate Layers of the Model**

Contents

Building Model
- Training
- Training Performance
- Confusion Matrix

Improving Result by Image Augmentation
- Augmentation Using Keras
- Learning Rate

Building Model: Training

한번의 epoch에 대하여 trainset 전체를 한번 학습하고 validset으로 검증이 이루어지는 단계

```
[25] epochs = 20
     batch_size = 256
     history_1 = model.fit(train_x,train_y,batch_size=batch_size,epochs=epochs,validation_data=[val_x,val_y])

     Epoch 1/20
     148/148 [==============================] - 120s 809ms/step - loss: 0.3499 - accuracy: 0.8874 - val_loss: 0.0969 - val_accuracy: 0.9700
     Epoch 2/20
     148/148 [==============================] - 110s 744ms/step - loss: 0.0764 - accuracy: 0.9772 - val_loss: 0.0600 - val_accuracy: 0.9812
     Epoch 3/20
     148/148 [==============================] - 110s 742ms/step - loss: 0.0501 - accuracy: 0.9841 - val_loss: 0.0559 - val_accuracy: 0.9800
     Epoch 4/20
     148/148 [==============================] - 110s 743ms/step - loss: 0.0396 - accuracy: 0.9875 - val_loss: 0.0467 - val_accuracy: 0.9848
     Epoch 5/20
     148/148 [==============================] - 110s 741ms/step - loss: 0.0338 - accuracy: 0.9891 - val_loss: 0.0401 - val_accuracy: 0.9881
     Epoch 6/20
     148/148 [==============================] - 110s 747ms/step - loss: 0.0247 - accuracy: 0.9923 - val_loss: 0.0422 - val_accuracy: 0.9869
     Epoch 7/20
     148/148 [==============================] - 111s 748ms/step - loss: 0.0203 - accuracy: 0.9928 - val_loss: 0.0454 - val_accuracy: 0.9874
     Epoch 8/20
     148/148 [==============================] - 110s 747ms/step - loss: 0.0192 - accuracy: 0.9935 - val_loss: 0.0454 - val_accuracy: 0.9855
     Epoch 9/20
     148/148 [==============================] - 111s 749ms/step - loss: 0.0179 - accuracy: 0.9942 - val_loss: 0.0386 - val_accuracy: 0.9883
     Epoch 10/20
     148/148 [==============================] - 110s 746ms/step - loss: 0.0134 - accuracy: 0.9956 - val_loss: 0.0472 - val_accuracy: 0.9876
     Epoch 11/20
     148/148 [==============================] - 110s 742ms/step - loss: 0.0154 - accuracy: 0.9947 - val_loss: 0.0371 - val_accuracy: 0.9902
     Epoch 12/20
     148/148 [==============================] - 110s 747ms/step - loss: 0.0121 - accuracy: 0.9960 - val_loss: 0.0435 - val_accuracy: 0.9895
     Epoch 13/20
     148/148 [==============================] - 111s 749ms/step - loss: 0.0123 - accuracy: 0.9960 - val_loss: 0.0523 - val_accuracy: 0.9888
     Epoch 14/20
     148/148 [==============================] - 110s 746ms/step - loss: 0.0110 - accuracy: 0.9963 - val_loss: 0.0492 - val_accuracy: 0.9876
     Epoch 15/20
     148/148 [==============================] - 110s 747ms/step - loss: 0.0086 - accuracy: 0.9972 - val_loss: 0.0557 - val_accuracy: 0.9879
     Epoch 16/20
     148/148 [==============================] - 110s 746ms/step - loss: 0.0097 - accuracy: 0.9967 - val_loss: 0.0527 - val_accuracy: 0.9857
     Epoch 17/20
     148/148 [==============================] - 111s 747ms/step - loss: 0.0071 - accuracy: 0.9975 - val_loss: 0.0457 - val_accuracy: 0.9898
     Epoch 18/20
     148/148 [==============================] - 111s 749ms/step - loss: 0.0093 - accuracy: 0.9971 - val_loss: 0.0433 - val_accuracy: 0.9890
     Epoch 19/20
     148/148 [==============================] - 110s 746ms/step - loss: 0.0085 - accuracy: 0.9973 - val_loss: 0.0578 - val_accuracy: 0.9879
     Epoch 20/20
     148/148 [==============================] - 110s 745ms/step - loss: 0.0079 - accuracy: 0.9973 - val_loss: 0.0466 - val_accuracy: 0.9886
```

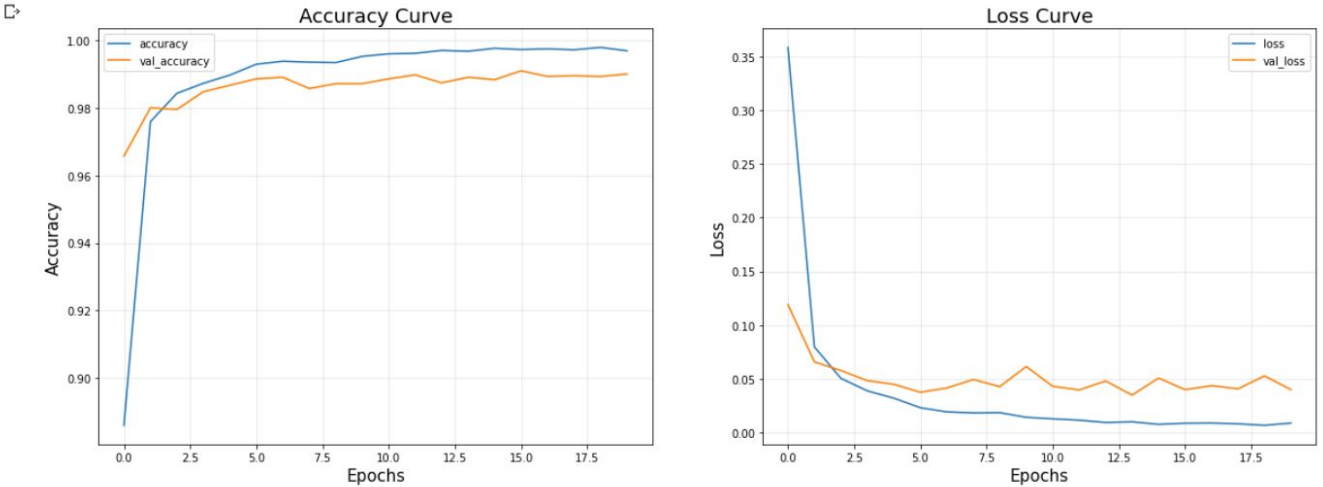
Dataset size 60,000 samples이어서 30분걸림ㅋ.ㅋ

- 한 번의 epoch는 인공 신경망에서 전체 데이터 셋에 대해 forward pass/backward pass 과정을 거친 것을 말함. 즉, 전체 데이터 셋에 대해 한 번 학습을 완료한 상태

여기서 epochs = 20는 전체 데이터를 20번 사용해서 학습을 거치는 것

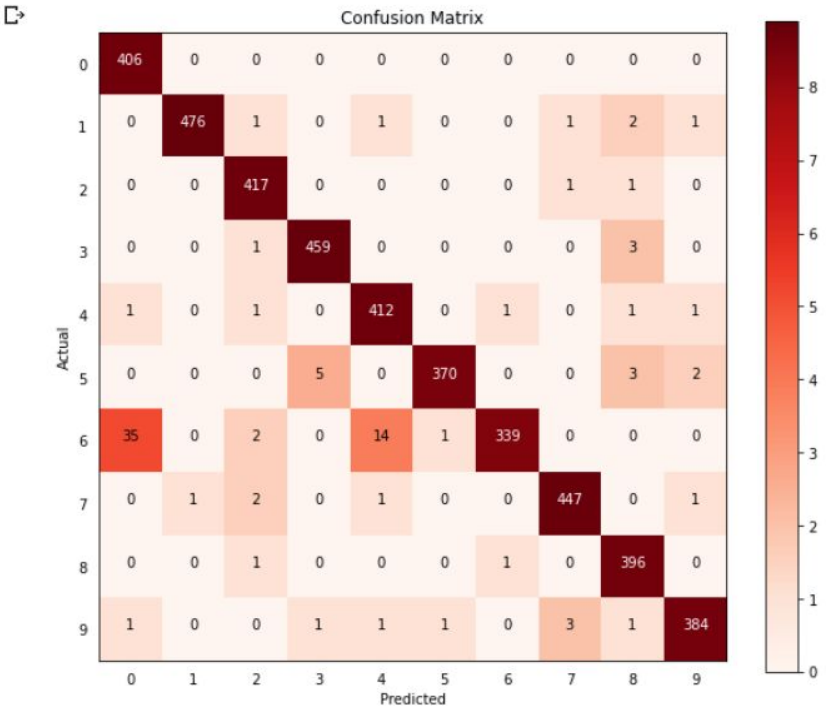
Building Model: Training Performance

각 epoch 마다 training, validation set의 정확도, 손실 뽑아내기



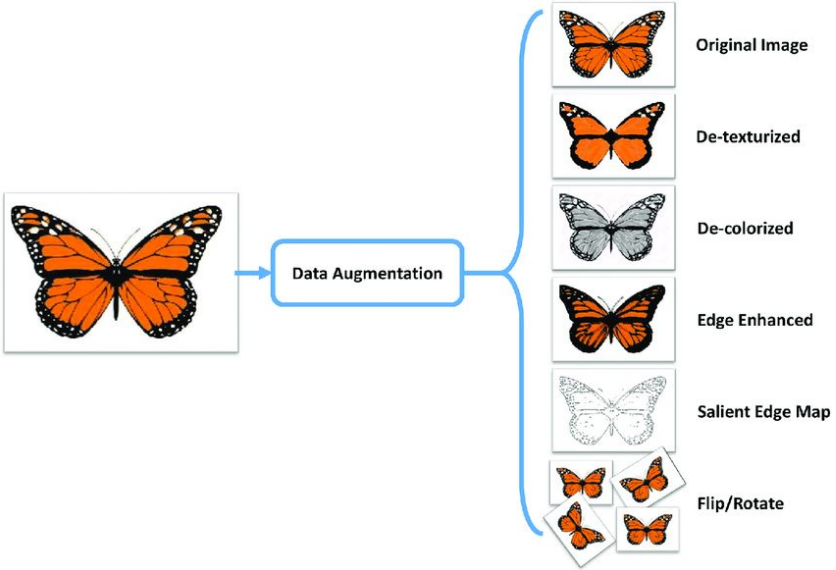
Accuracy↑, Loss↓: **Model** > Validation set

Building Model: Confusion Matrix



- **To visualize of the performance of an algorithm**
- Row: Instances in a predicted class
- Column: Instances in an actual class
- Makes it easy to see if the system is confusing two classes

Image Augmentation



- 머신러닝의 과적합 문제를 해결하기 위함
- 한정된 수량의 데이터셋(이미지)들을 회전, 색변화 등을 주어 더 많은 양의 데이터셋을 머신러닝이 학습할 수 있게함

Image Augmentation in Digit Recognizer

```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=False,
        vertical_flip=False)

datagen.fit(train_x)
```

- **rotation_range**
- **zoom_range**
- **width_shift_range**
- **height_shift_range**

Vertical, Horizontal Flip도 대표적인 **Image Augmentation**의 방법이지만 **digit**을 다루는 주제 특성상 사용할 수 없었음. **(EX: 9**를 뒤집으면 **6**으로 인식 가능**)**

Learning Rate

```
lrr = ReduceLROnPlateau(monitor='val_accuracy',patience=2,verbose=1,factor=0.5, min_lr=0.00001)
```

- Machine Learning에서 train 되는 양의 비율(학습률)
- If too high? -> 오류를 감지하지 못하고 다 train 해버림
- If too low? -> Machine Learning 과정에 필요한 시간 증가 및 수많은 오류 발견으로 Machine Learning이 멈출 수 있음

Parameters:

- **monitor**: 모니터링할 수량 (val_accuracy)
- **patience**: Metric이 얼마 동안 변화가 없을 때 learning rate를 감소시킬지 결정 (2로 지정)
- **factor**: learning rate 감소 비율 상수 (`0.5`로 지정)
- **min_lr**: leaning rate 하한값 (`0.00001`로 지정)

Results after Image Augmentation

Errors in validation set: 185

Error Persentage : 2.2023809523809526

Accuracy : 97.79761904761905

Validation set Shape : 8400

Before Image Augmentation

Errors in validation set: 57

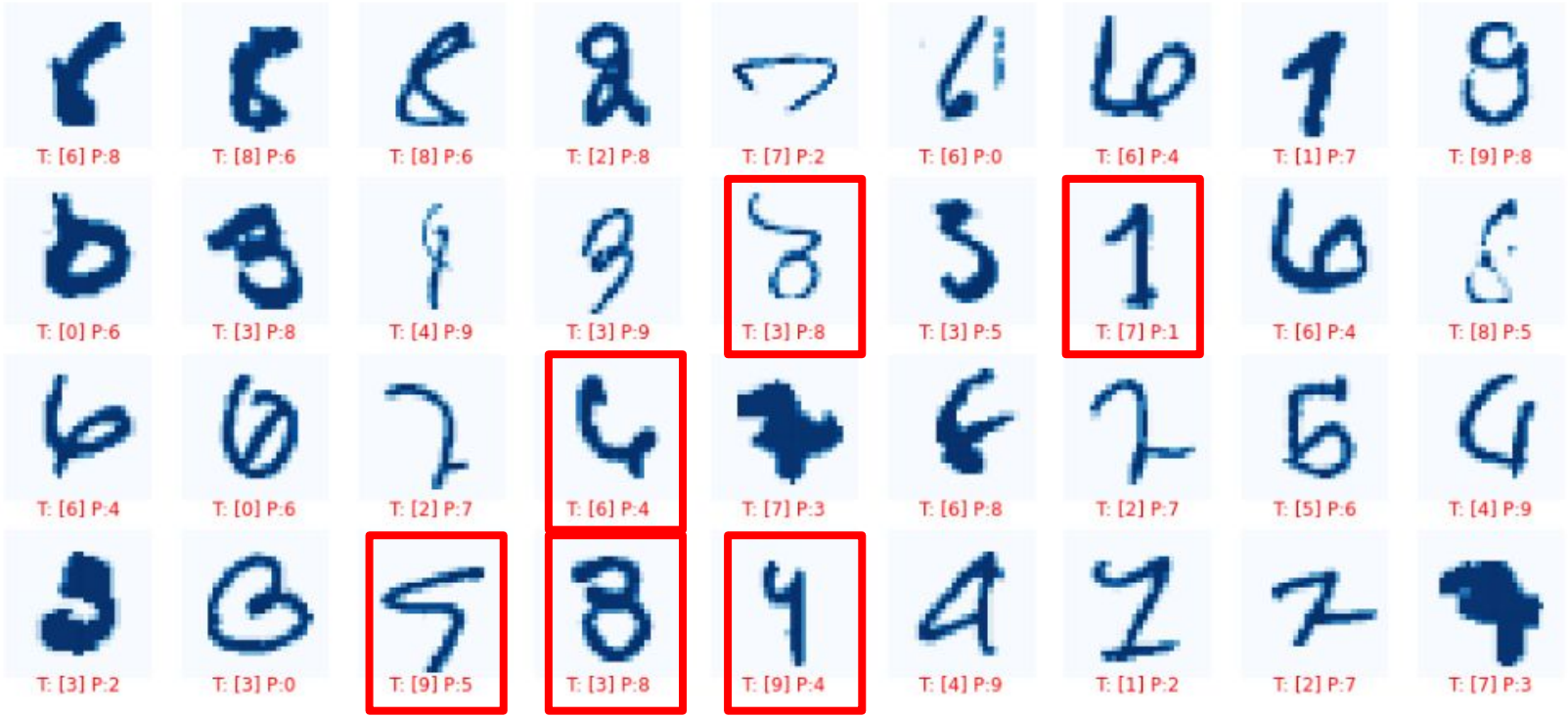
Error Persentage : 0.6785714285714286

Accuracy : 99.32142857142857

Validation set Shape : 8400

After Image Augmentation

Errors


T: [6] P:8
T: [8] P:6
T: [8] P:6
T: [2] P:8
T: [7] P:2
T: [6] P:0
T: [6] P:4
T: [1] P:7
T: [9] P:8
T: [0] P:6
T: [3] P:8
T: [4] P:9
T: [3] P:9
T: [3] P:8
T: [3] P:5
T: [7] P:1
T: [6] P:4
T: [8] P:5
T: [6] P:4
T: [0] P:6
T: [2] P:7
T: [6] P:4
T: [7] P:3
T: [6] P:8
T: [2] P:7
T: [5] P:6
T: [4] P:9
T: [3] P:2
T: [3] P:0
T: [9] P:5
T: [3] P:8
T: [9] P:4
T: [4] P:9
T: [1] P:2
T: [2] P:7
T: [7] P:3