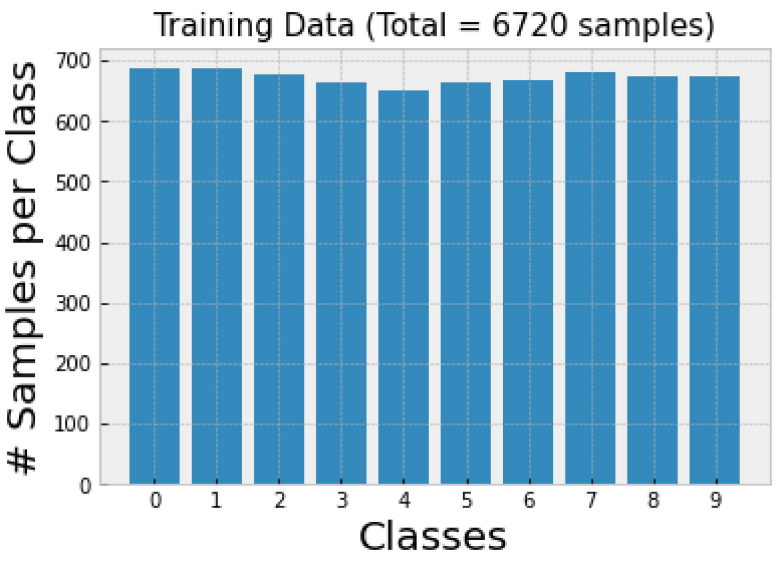
Training the data on CNN

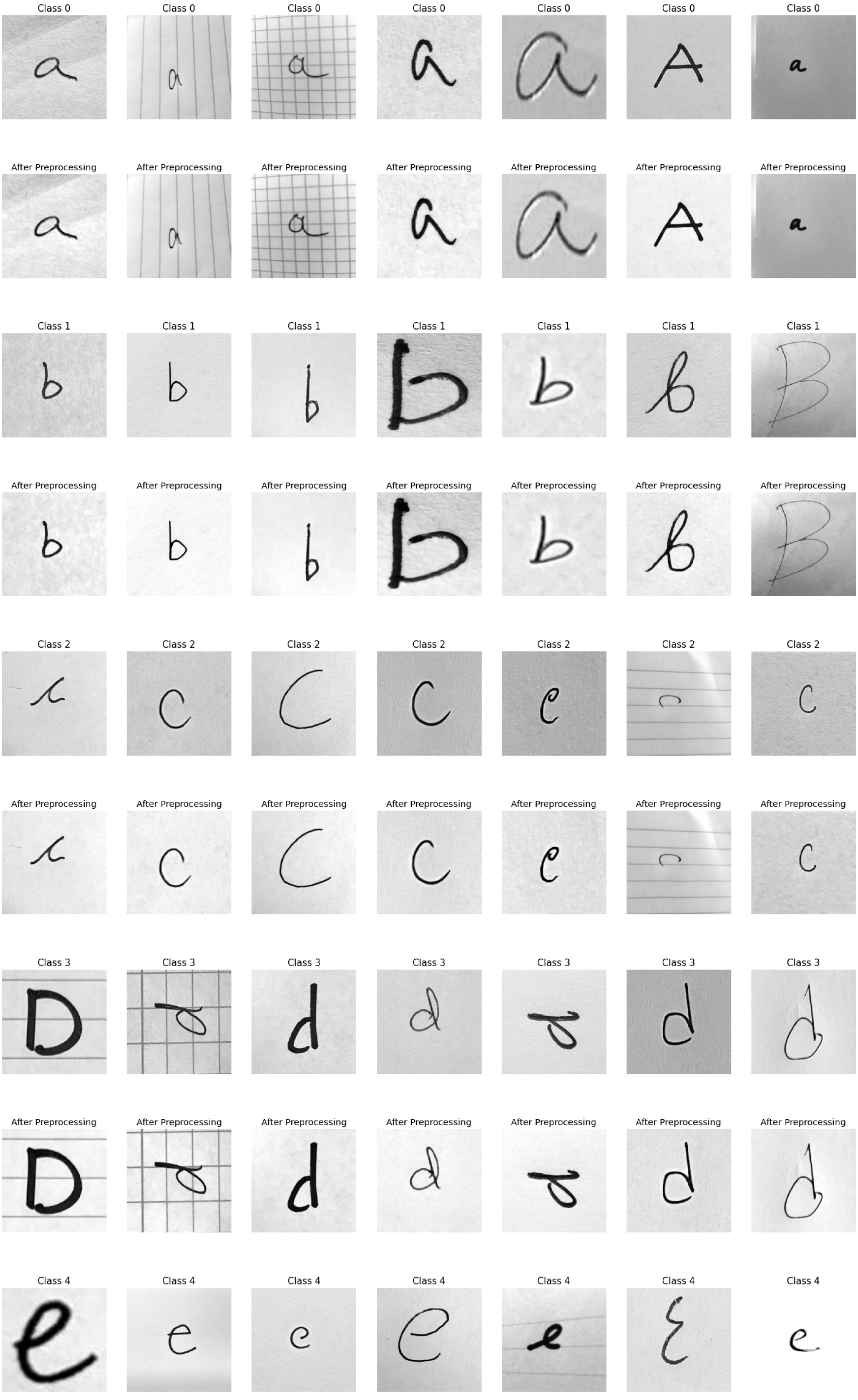
In [1]:
```
from sklearn.model_selection import KFold, StratifiedKFold
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, Dropout, RandomFlip, RandomRotation, RandomZoom, RandomContrast, RandomCrop,
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.model_selection import train_test_split, GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from tensorflow.keras.layers import BatchNormalization
from numpy import mean, std
from matplotlib import pyplot as plt
import tensorflow as tf
import numpy as np
import pandas as pd
import numpy.random as npr
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')

# Loading Data
data_train = np.load('data_train.npy')
labels_train = np.load('new_labels_train.npy')

print(data_train.shape, labels_train.shape)

# Counting number samples per class
vals, counts = np.unique(labels_train, return_counts=True)

plt.bar(vals, counts)
plt.xticks(range(10),range(10))
plt.xlabel('Classes',size=20)
plt.ylabel('# Samples per Class', size=20)
plt.title('Training Data (Total = '+str(data_train.shape[1])+' samples)',size=15);
```

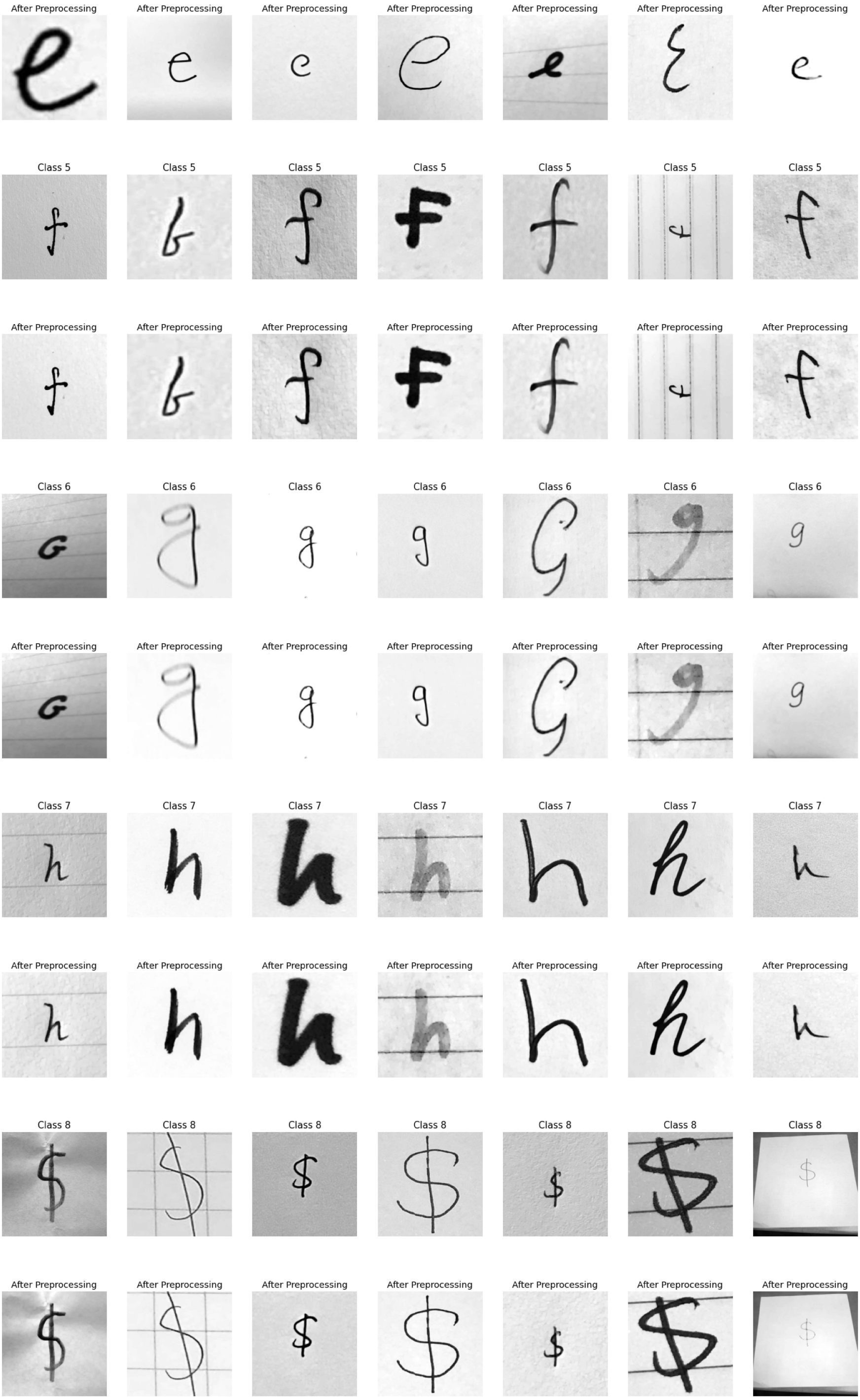
```
(90000, 6720) (6720,)
```

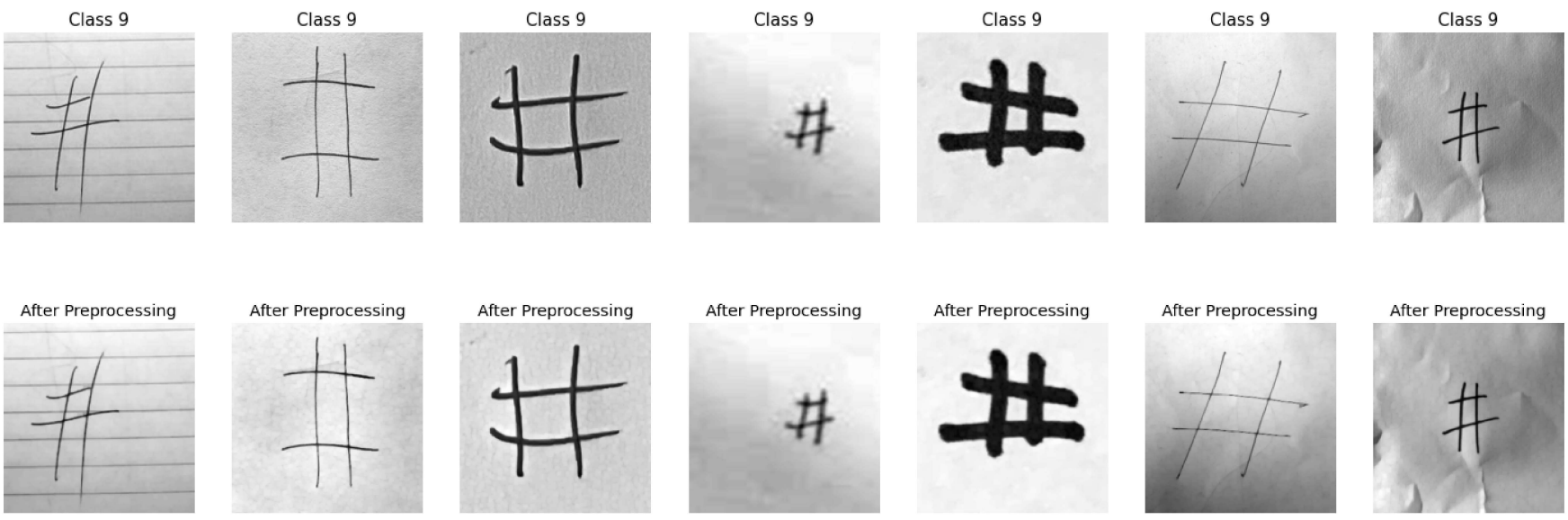


Visualising some images after and before preprocessing

In [2]:
```
import cv2
kernel = np.ones((5,5),np.uint8)
for i in range(0,10):
    rnd_sample = npr.permutation(np.where(labels_train==i)[0])
    fig1=plt.figure(figsize=(25,20))
    for j in range(7):
        fig1.add_subplot(5,7,j+1)
        plt.imshow(data_train[:,rnd_sample[j]].reshape((300,300)),cmap='gray')
        plt.axis('off');plt.title('Class '+str(int(labels_train[rnd_sample[j]])),size=15)
    plt.show()
    print('\n\n')
    j = 0
    fig=plt.figure(figsize=(25,20))
    dt = []
    k=0
    for j in range(7):
        fig.add_subplot(5,7,j+1)
        dt.append(cv2.medianBlur((data_train[:,rnd_sample[j]].reshape((300,300))), 3))
        dt[j] = cv2.morphologyEx(dt[j], cv2.MORPH_OPEN, kernel)
        plt.imshow(dt[j],cmap='gray')
        k = k + 1
        plt.axis('off');plt.title('After Preprocessing')
    plt.show()
    print('\n\n')
```


Class 0
Class 0
Class 0
Class 0
Class 0
Class 0
Class 0
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
Class 1
Class 1
Class 1
Class 1
Class 1
Class 1
Class 1
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
Class 2
Class 2
Class 2
Class 2
Class 2
Class 2
Class 2
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
Class 3
Class 3
Class 3
Class 3
Class 3
Class 3
Class 3
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
After Preprocessing
Class 4
Class 4
Class 4
Class 4
Class 4
Class 4
Class 4


After Preprocessing
Class 5
Class 6
Class 7
Class 8



In [3]:

```
# import argparse
# import skimage as sk
# from skimage import transform
# from skimage import util
import cv2
train_data_length = data_train.shape[1]
# Initial function for loading training set and preprocess dataset
def load_dataset():
    # load dataset
    data_rgb = []
    kernel = np.ones((4,4),np.uint8)
    for i in range(train_data_length):
        data_rgb.append(data_train[:,i].reshape(300,300))
        data_rgb[i] = cv2.medianBlur(data_rgb[i], 3)
        data_rgb[i] = cv2.morphologyEx(data_rgb[i], cv2.MORPH_OPEN, kernel)
        data_rgb[i] = cv2.resize(data_rgb[i], (50,50), interpolation=cv2.INTER_AREA)
    data_rgb = np.array(data_rgb)
    print(data_rgb.shape)

    # Using 80-20 split
    trainX, testX, trainY, testY = train_test_split(data_rgb, labels_train, test_size = 0.2, stratify=labels_train)

    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 50,50, 1))
    testX = testX.reshape((testX.shape[0], 50,50, 1))

    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    print(trainX.shape, trainY.shape, testX.shape, testY.shape)
    return trainX, trainY, testX, testY
```

In [4]:

```
# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')

    # he_uniformize to range 0-1
    train_norm = (train_norm) / 255.0
    test_norm = (test_norm) / 255.0

    # return he_uniformized images
    return train_norm, test_norm
```

In [5]:

```
# define cnn model
def define_model(learn_rate = '0.007'):
    model = Sequential()
    model.add(RandomContrast((0,0.3),input_shape=(50,50,1)))
    model.add(RandomRotation(0.1, fill_mode='reflect', interpolation='bilinear'))
    model.add(Conv2D(32, (5, 5), activation='relu', kernel_initializer='he_uniform', input_shape=(50,50, 1)))
    model.add(Conv2D(32, (5, 5), activation='relu', kernel_initializer='he_uniform'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.5))
    model.add(Conv2D(32, (3,3),activation='relu', kernel_initializer='he_uniform'))
    model.add(Conv2D(64, (3,3),activation='relu', kernel_initializer='he_uniform'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.5))
    model.add(Conv2D(64,(3,3), activation='relu', kernel_initializer='he_uniform'))
    model.add(Conv2D(64,(3,3), activation='relu', kernel_initializer='he_uniform'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.4))
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.4))
    model.add(Dense(10, activation='softmax'))
```

```
    # compile model
    opt = Adam(learning_rate = learn_rate , epsilon=1e-5)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

In [6]:

```
# Function for Finding best parameters using GRID CV
def training(dataX, dataY, tes_X, tes_y):
    parameters = {'epochs':[800], 'batch_size':[32, 128], 'learn_rate':[0.001, 0.0007]}
    model = KerasClassifier(build_fn = define_model, epochs = 800, batch_size = 32)
    grid = GridSearchCV(estimator = model, param_grid = parameters, n_jobs = 1, cv = 3, verbose = 2, return_train_score =
    grid_result = grid.fit(dataX, dataY)

    # summarize results
    print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
    means = grid_result.cv_results_['mean_test_score']
    stds = grid_result.cv_results_['std_test_score']
    params = grid_result.cv_results_['params']
    for mean, stdev, param in zip(means, stds, params):
        print("%f (%f) with: %r" % (mean, stdev, param))
    return grid_result.best_estimator_
```

Code for GRID Search and Best estimator in comments below, you don't have to run these 2 cells below as we know our best parameters now.

In [13]:

```
# def load_and_train():
#     # load dataset
#     trainX, trainY, testX, testY = load_dataset()

#     # prepare pixel data
#     trainX, testX = prep_pixels(trainX, testX)

#     # evaluate model
#     best_estimator = training(trainX, trainY, testX, testY)
#     return best_estimator
# best_estimator = load_and_train()
```

In [10]:

```
# print(best_estimator.get_params())
```

```
{'epochs': 800, 'batch_size': 32, 'learn_rate': 0.0007, 'build_fn': <function define_model at 0x2b6665dff5e0>}
```

We can see from above what the best parameters are, so we will train our model now on these parameters.

In [7]:

```
def final_training(dataX, dataY, testX, testY):
    model = define_model(learn_rate = 0.0007)
    print(model.summary())
    trainX, trainY, testX, testY = dataX, dataY, testX, testY
    # fit model
    history = model.fit(trainX, trainY, epochs = 800, batch_size = 32, validation_data = (testX, testY), verbose=1)
    print("Model Fitted")
    return model, history
```

In [8]:

```
# plot diagnostic learning curves
def summarize_diagnostics(histories):
    plt.figure(figsize=(12,7))
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Training and validation Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('epoch')
    plt.legend(['Accuracy', 'Val Accuracy'], loc='lower right')
    plt.show()
    plt.figure(figsize=(12,7))
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Training and validation Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epochs')
    plt.legend(['Loss', 'Val Loss'], loc='upper right')
    plt.show()
```

This below cell is where we call 3 functions to split and preprocess the dataset, Normalize the images and finally train on 80% data.

In [9]:

```
trainX, trainY, testX, testY = load_dataset()

# prepare pixel data
trainX, testX = prep_pixels(trainX, testX)

# training the model
model, history = final_training(trainX, trainY, testX, testY)
```

```
(6720, 50, 50)
```

```
(5376, 50, 50, 1) (5376, 10) (1344, 50, 50, 1) (1344, 10)
2022-04-20 13:56:29.748779: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-04-20 13:56:30.252276: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 79125 MB memory:  -> device: 0, name: NVIDIA A100-SXM4-80GB, pci bus id: 0000:47:00.0, compute capability: 8.0
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
random_contrast (RandomContr (None, 50, 50, 1)         0
_________________________________________________________________
random_rotation (RandomRotat (None, 50, 50, 1)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 46, 46, 32)        832
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 42, 42, 32)        25632
_________________________________________________________________
batch_normalization (BatchNo (None, 42, 42, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 21, 21, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 21, 21, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 19, 19, 32)        9248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 17, 17, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 17, 17, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 6, 6, 64)          36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 64)          256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 256)               65792
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 228,682
Trainable params: 228,362
Non-trainable params: 320
_________________________________________________________________
None
2022-04-20 13:56:31.266568: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/800
2022-04-20 13:56:33.352678: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8201
2022-04-20 13:56:34.264488: W tensorflow/stream_executor/gpu/asm_compiler.cc:77] Couldn't get ptxas version string: Internal: Running ptxas --version returned 32512
2022-04-20 13:56:34.369035: W tensorflow/stream_executor/gpu/redzone_allocator.cc:314] Internal: ptxas exited with non-zero error code 32512, output:
Relying on driver to perform ptx compilation.
Modify $PATH to customize ptxas location.
This message will be only logged once.
2022-04-20 13:56:35.782709: I tensorflow/stream_executor/cuda/cuda_blas.cc:1760] TensorFloat-32 will be used for the matrix multiplication. This will only be logged once.
168/168 [==============================] - 6s 6ms/step - loss: 3.4276 - accuracy: 0.1025 - val_loss: 2.3224 - val_accuracy: 0.1079
Epoch 2/800
168/168 [==============================] - 1s 5ms/step - loss: 2.4370 - accuracy: 0.1256 - val_loss: 2.2919 - val_accuracy: 0.1287
Epoch 3/800
168/168 [==============================] - 1s 4ms/step - loss: 2.3497 - accuracy: 0.1198 - val_loss: 2.2682 - val_accuracy: 0.1659
Epoch 4/800
168/168 [==============================] - 1s 5ms/step - loss: 2.2975 - accuracy: 0.1295 - val_loss: 2.2488 - val_accuracy: 0.1696
Epoch 5/800
168/168 [==============================] - 1s 5ms/step - loss: 2.2669 - accuracy: 0.1479 - val_loss: 2.1815 - val_accuracy: 0.1927
Epoch 6/800
168/168 [==============================] - 1s 5ms/step - loss: 2.2229 - accuracy: 0.1788 - val_loss: 2.1117 - val_accuracy: 0.2277
Epoch 7/800
```

```
168/168 [==============================] - 1s 4ms/step - loss: 2.1738 - accuracy: 0.1970 - val_loss: 2.0937 - val_accurac
y: 0.2455
Epoch 8/800
168/168 [==============================] - 1s 5ms/step - loss: 2.1528 - accuracy: 0.2052 - val_loss: 2.0337 - val_accurac
y: 0.2448
Epoch 9/800
168/168 [==============================] - 1s 5ms/step - loss: 2.1084 - accuracy: 0.2108 - val_loss: 1.9826 - val_accurac
y: 0.2805
Epoch 10/800
168/168 [==============================] - 1s 4ms/step - loss: 2.0672 - accuracy: 0.2193 - val_loss: 1.8753 - val_accurac
y: 0.3103
Epoch 11/800
168/168 [==============================] - 1s 4ms/step - loss: 2.0255 - accuracy: 0.2461 - val_loss: 1.9378 - val_accurac
y: 0.2969
Epoch 12/800
168/168 [==============================] - 1s 4ms/step - loss: 1.9790 - accuracy: 0.2675 - val_loss: 1.8436 - val_accurac
y: 0.3356
Epoch 13/800
168/168 [==============================] - 1s 5ms/step - loss: 1.9527 - accuracy: 0.2673 - val_loss: 1.7890 - val_accurac
y: 0.3497
Epoch 14/800
168/168 [==============================] - 1s 5ms/step - loss: 1.9237 - accuracy: 0.2811 - val_loss: 1.6658 - val_accurac
y: 0.3981
Epoch 15/800
168/168 [==============================] - 1s 5ms/step - loss: 1.8854 - accuracy: 0.3136 - val_loss: 1.6574 - val_accurac
y: 0.4204
Epoch 16/800
168/168 [==============================] - 1s 4ms/step - loss: 1.8559 - accuracy: 0.3203 - val_loss: 1.5604 - val_accurac
y: 0.4494
Epoch 17/800
168/168 [==============================] - 1s 5ms/step - loss: 1.8001 - accuracy: 0.3432 - val_loss: 1.5853 - val_accurac
y: 0.4397
Epoch 18/800
168/168 [==============================] - 1s 5ms/step - loss: 1.7185 - accuracy: 0.3823 - val_loss: 1.3914 - val_accurac
y: 0.5685
Epoch 19/800
168/168 [==============================] - 1s 5ms/step - loss: 1.6457 - accuracy: 0.4131 - val_loss: 1.4740 - val_accurac
y: 0.4635
Epoch 20/800
168/168 [==============================] - 1s 4ms/step - loss: 1.5861 - accuracy: 0.4306 - val_loss: 1.4350 - val_accurac
y: 0.4874
Epoch 21/800
168/168 [==============================] - 1s 4ms/step - loss: 1.5362 - accuracy: 0.4554 - val_loss: 1.3044 - val_accurac
y: 0.5655
Epoch 22/800
168/168 [==============================] - 1s 5ms/step - loss: 1.4792 - accuracy: 0.4829 - val_loss: 1.4089 - val_accurac
y: 0.5238
Epoch 23/800
168/168 [==============================] - 1s 5ms/step - loss: 1.4204 - accuracy: 0.5123 - val_loss: 1.1664 - val_accurac
y: 0.5781
Epoch 24/800
168/168 [==============================] - 1s 4ms/step - loss: 1.3674 - accuracy: 0.5268 - val_loss: 1.0499 - val_accurac
y: 0.6548
Epoch 25/800
168/168 [==============================] - 1s 5ms/step - loss: 1.3174 - accuracy: 0.5463 - val_loss: 0.9300 - val_accurac
y: 0.6882
Epoch 26/800
168/168 [==============================] - 1s 5ms/step - loss: 1.2657 - accuracy: 0.5605 - val_loss: 0.9048 - val_accurac
y: 0.6897
Epoch 27/800
168/168 [==============================] - 1s 5ms/step - loss: 1.2483 - accuracy: 0.5711 - val_loss: 0.9503 - val_accurac
y: 0.6763
Epoch 28/800
168/168 [==============================] - 1s 5ms/step - loss: 1.1901 - accuracy: 0.5952 - val_loss: 0.8530 - val_accurac
y: 0.7150
Epoch 29/800
168/168 [==============================] - 1s 6ms/step - loss: 1.1623 - accuracy: 0.6044 - val_loss: 0.8687 - val_accurac
y: 0.7173
Epoch 30/800
168/168 [==============================] - 1s 5ms/step - loss: 1.1273 - accuracy: 0.6138 - val_loss: 0.8598 - val_accurac
y: 0.7121
Epoch 31/800
168/168 [==============================] - 1s 5ms/step - loss: 1.0953 - accuracy: 0.6300 - val_loss: 0.8026 - val_accurac
y: 0.7277
Epoch 32/800
168/168 [==============================] - 1s 5ms/step - loss: 1.0795 - accuracy: 0.6453 - val_loss: 0.7402 - val_accurac
y: 0.7560
Epoch 33/800
168/168 [==============================] - 1s 4ms/step - loss: 1.0098 - accuracy: 0.6628 - val_loss: 0.7267 - val_accurac
y: 0.7485
Epoch 34/800
168/168 [==============================] - 1s 4ms/step - loss: 1.0077 - accuracy: 0.6656 - val_loss: 0.7047 - val_accurac
y: 0.7805
Epoch 35/800
168/168 [==============================] - 1s 5ms/step - loss: 0.9852 - accuracy: 0.6782 - val_loss: 0.7622 - val_accurac
y: 0.7411
Epoch 36/800
168/168 [==============================] - 2s 11ms/step - loss: 0.9664 - accuracy: 0.6775 - val_loss: 0.6947 - val_accura
cy: 0.7641
Epoch 37/800
168/168 [==============================] - 1s 4ms/step - loss: 0.9167 - accuracy: 0.6936 - val_loss: 0.6373 - val_accurac
y: 0.7932
Epoch 38/800
168/168 [==============================] - 1s 4ms/step - loss: 0.9023 - accuracy: 0.7070 - val_loss: 0.6056 - val_accurac
y: 0.8065
```

```
Epoch 39/800
168/168 [==============================] - 1s 5ms/step - loss: 0.8859 - accuracy: 0.7096 - val_loss: 0.6947 - val_accurac
y: 0.7790
Epoch 40/800
168/168 [==============================] - 1s 5ms/step - loss: 0.8911 - accuracy: 0.7063 - val_loss: 0.7163 - val_accurac
y: 0.7626
Epoch 41/800
168/168 [==============================] - 1s 5ms/step - loss: 0.8608 - accuracy: 0.7160 - val_loss: 0.6201 - val_accurac
y: 0.7865
Epoch 42/800
168/168 [==============================] - 1s 4ms/step - loss: 0.8586 - accuracy: 0.7184 - val_loss: 0.7056 - val_accurac
y: 0.7612
Epoch 43/800
168/168 [==============================] - 1s 5ms/step - loss: 0.8082 - accuracy: 0.7307 - val_loss: 0.6105 - val_accurac
y: 0.7902
Epoch 44/800
168/168 [==============================] - 1s 5ms/step - loss: 0.7918 - accuracy: 0.7394 - val_loss: 0.6486 - val_accurac
y: 0.7783
Epoch 45/800
168/168 [==============================] - 1s 5ms/step - loss: 0.7865 - accuracy: 0.7372 - val_loss: 0.5711 - val_accurac
y: 0.8222
Epoch 46/800
168/168 [==============================] - 1s 4ms/step - loss: 0.7584 - accuracy: 0.7493 - val_loss: 0.4940 - val_accurac
y: 0.8341
Epoch 47/800
168/168 [==============================] - 1s 4ms/step - loss: 0.7589 - accuracy: 0.7500 - val_loss: 0.7121 - val_accurac
y: 0.7746
Epoch 48/800
168/168 [==============================] - 1s 5ms/step - loss: 0.7588 - accuracy: 0.7467 - val_loss: 0.5285 - val_accurac
y: 0.8326
Epoch 49/800
168/168 [==============================] - 1s 4ms/step - loss: 0.7295 - accuracy: 0.7599 - val_loss: 0.6696 - val_accurac
y: 0.7954
Epoch 50/800
168/168 [==============================] - 1s 4ms/step - loss: 0.7224 - accuracy: 0.7636 - val_loss: 0.5189 - val_accurac
y: 0.8348
Epoch 51/800
168/168 [==============================] - 1s 4ms/step - loss: 0.7113 - accuracy: 0.7706 - val_loss: 0.5149 - val_accurac
y: 0.8318
Epoch 52/800
168/168 [==============================] - 1s 5ms/step - loss: 0.7040 - accuracy: 0.7822 - val_loss: 0.4644 - val_accurac
y: 0.8467
Epoch 53/800
168/168 [==============================] - 1s 4ms/step - loss: 0.6747 - accuracy: 0.7760 - val_loss: 0.4953 - val_accurac
y: 0.8296
Epoch 54/800
168/168 [==============================] - 1s 4ms/step - loss: 0.6708 - accuracy: 0.7859 - val_loss: 0.4516 - val_accurac
y: 0.8519
Epoch 55/800
168/168 [==============================] - 1s 5ms/step - loss: 0.6672 - accuracy: 0.7835 - val_loss: 0.5157 - val_accurac
y: 0.8356
Epoch 56/800
168/168 [==============================] - 1s 4ms/step - loss: 0.6797 - accuracy: 0.7738 - val_loss: 0.5063 - val_accurac
y: 0.8259
Epoch 57/800
168/168 [==============================] - 1s 4ms/step - loss: 0.6529 - accuracy: 0.7831 - val_loss: 0.4404 - val_accurac
y: 0.8564
Epoch 58/800
168/168 [==============================] - 1s 4ms/step - loss: 0.6453 - accuracy: 0.7898 - val_loss: 0.6866 - val_accurac
y: 0.7723
Epoch 59/800
168/168 [==============================] - 1s 5ms/step - loss: 0.6697 - accuracy: 0.7844 - val_loss: 0.7575 - val_accurac
y: 0.7812
Epoch 60/800
168/168 [==============================] - 1s 4ms/step - loss: 0.6371 - accuracy: 0.7967 - val_loss: 0.5183 - val_accurac
y: 0.8318
Epoch 61/800
168/168 [==============================] - 1s 4ms/step - loss: 0.6327 - accuracy: 0.7930 - val_loss: 0.5575 - val_accurac
y: 0.8155
Epoch 62/800
168/168 [==============================] - 1s 5ms/step - loss: 0.6313 - accuracy: 0.8012 - val_loss: 0.6615 - val_accurac
y: 0.7999
Epoch 63/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5993 - accuracy: 0.8052 - val_loss: 0.4122 - val_accurac
y: 0.8676
Epoch 64/800
168/168 [==============================] - 1s 5ms/step - loss: 0.6065 - accuracy: 0.8092 - val_loss: 0.4945 - val_accurac
y: 0.8348
Epoch 65/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5875 - accuracy: 0.8088 - val_loss: 0.4826 - val_accurac
y: 0.8527
Epoch 66/800
168/168 [==============================] - 1s 4ms/step - loss: 0.5786 - accuracy: 0.8106 - val_loss: 0.4357 - val_accurac
y: 0.8430
Epoch 67/800
168/168 [==============================] - 1s 4ms/step - loss: 0.6017 - accuracy: 0.8067 - val_loss: 0.4859 - val_accurac
y: 0.8504
Epoch 68/800
168/168 [==============================] - 2s 11ms/step - loss: 0.5824 - accuracy: 0.8203 - val_loss: 0.4263 - val_accura
cy: 0.8668
Epoch 69/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5884 - accuracy: 0.8166 - val_loss: 0.4160 - val_accurac
y: 0.8757
Epoch 70/800
168/168 [==============================] - 1s 4ms/step - loss: 0.5780 - accuracy: 0.8140 - val_loss: 0.5490 - val_accurac
```

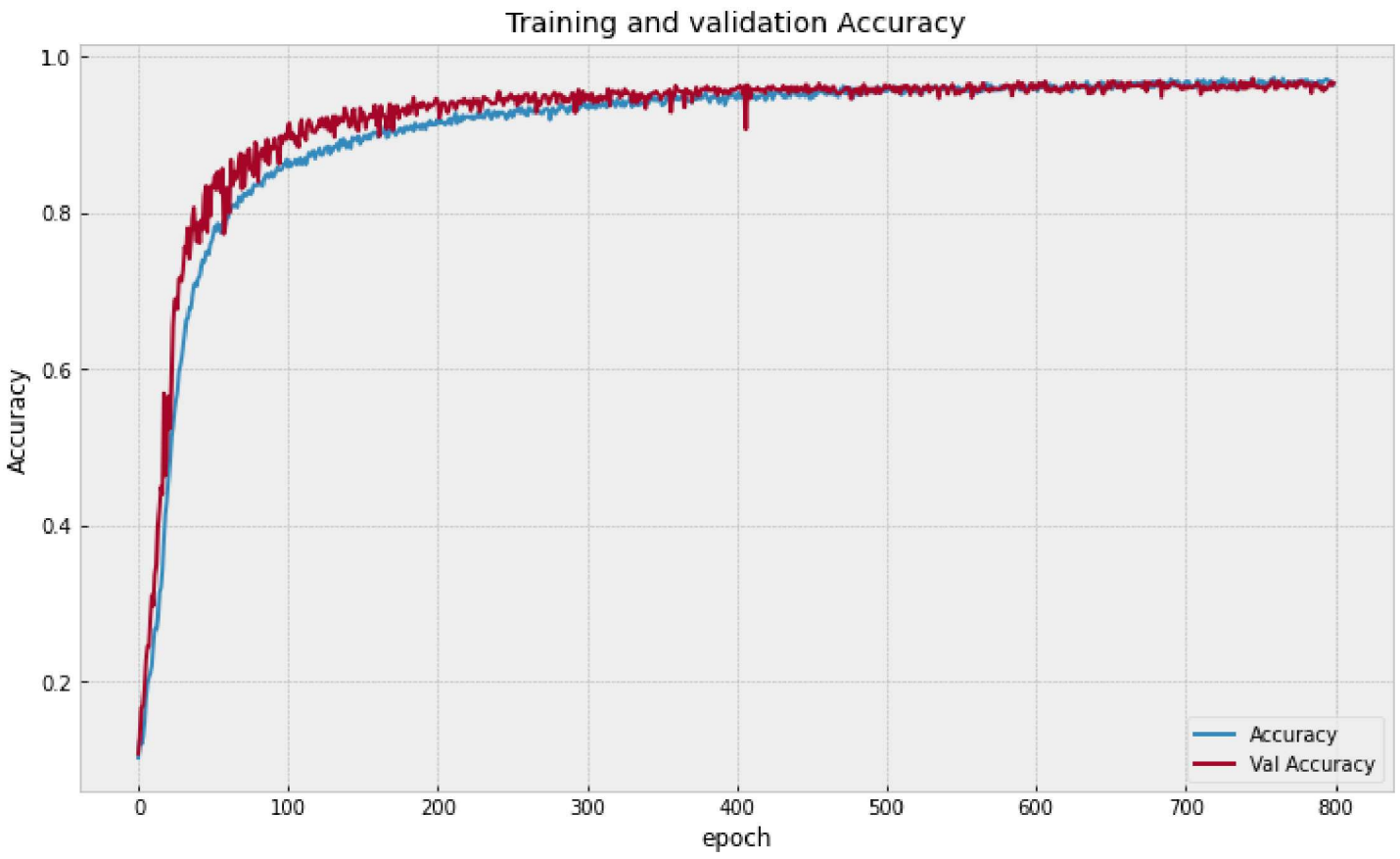
```
y: 0.8304
Epoch 71/800
168/168 [==============================] - 1s 4ms/step - loss: 0.5634 - accuracy: 0.8235 - val_loss: 0.3835 - val_accurac
y: 0.8735
Epoch 72/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5479 - accuracy: 0.8250 - val_loss: 0.4911 - val_accurac
y: 0.8333
Epoch 73/800
168/168 [==============================] - 1s 4ms/step - loss: 0.5637 - accuracy: 0.8212 - val_loss: 0.4285 - val_accurac
y: 0.8579
Epoch 74/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5567 - accuracy: 0.8251 - val_loss: 0.3736 - val_accurac
y: 0.8780
Epoch 75/800
168/168 [==============================] - 1s 4ms/step - loss: 0.5213 - accuracy: 0.8279 - val_loss: 0.3639 - val_accurac
y: 0.8817
Epoch 76/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5514 - accuracy: 0.8235 - val_loss: 0.4410 - val_accurac
y: 0.8542
Epoch 77/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5351 - accuracy: 0.8274 - val_loss: 0.4318 - val_accurac
y: 0.8571
Epoch 78/800
168/168 [==============================] - 1s 4ms/step - loss: 0.5232 - accuracy: 0.8358 - val_loss: 0.5192 - val_accurac
y: 0.8467
Epoch 79/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5207 - accuracy: 0.8350 - val_loss: 0.3710 - val_accurac
y: 0.8899
Epoch 80/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5231 - accuracy: 0.8346 - val_loss: 0.3568 - val_accurac
y: 0.8862
Epoch 81/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5043 - accuracy: 0.8348 - val_loss: 0.5231 - val_accurac
y: 0.8393
Epoch 82/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5217 - accuracy: 0.8363 - val_loss: 0.4040 - val_accurac
y: 0.8757
Epoch 83/800
168/168 [==============================] - 1s 4ms/step - loss: 0.5099 - accuracy: 0.8384 - val_loss: 0.4169 - val_accurac
y: 0.8713
Epoch 84/800
168/168 [==============================] - 1s 5ms/step - loss: 0.5017 - accuracy: 0.8337 - val_loss: 0.3528 - val_accurac
y: 0.8824
Epoch 85/800
168/168 [==============================] - 1s 5ms/step - loss: 0.4909 - accuracy: 0.8438 - val_loss: 0.3804 - val_accurac
y: 0.8757
Epoch 86/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4719 - accuracy: 0.8464 - val_loss: 0.4284 - val_accurac
y: 0.8661
Epoch 87/800
168/168 [==============================] - 1s 5ms/step - loss: 0.4867 - accuracy: 0.8493 - val_loss: 0.3290 - val_accurac
y: 0.8936
Epoch 88/800
168/168 [==============================] - 1s 5ms/step - loss: 0.4935 - accuracy: 0.8451 - val_loss: 0.4223 - val_accurac
y: 0.8616
Epoch 89/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4612 - accuracy: 0.8475 - val_loss: 0.3810 - val_accurac
y: 0.8780
Epoch 90/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4637 - accuracy: 0.8516 - val_loss: 0.3933 - val_accurac
y: 0.8854
Epoch 91/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4775 - accuracy: 0.8458 - val_loss: 0.3839 - val_accurac
y: 0.8817
Epoch 92/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4611 - accuracy: 0.8480 - val_loss: 0.3437 - val_accurac
y: 0.8951
Epoch 93/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4518 - accuracy: 0.8527 - val_loss: 0.3944 - val_accurac
y: 0.8839
Epoch 94/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4442 - accuracy: 0.8581 - val_loss: 0.3973 - val_accurac
y: 0.8795
Epoch 95/800
168/168 [==============================] - 1s 5ms/step - loss: 0.4690 - accuracy: 0.8540 - val_loss: 0.4223 - val_accurac
y: 0.8624
Epoch 96/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4367 - accuracy: 0.8544 - val_loss: 0.3291 - val_accurac
y: 0.8981
Epoch 97/800
168/168 [==============================] - 1s 5ms/step - loss: 0.4546 - accuracy: 0.8560 - val_loss: 0.3516 - val_accurac
y: 0.8891
Epoch 98/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4303 - accuracy: 0.8624 - val_loss: 0.3255 - val_accurac
y: 0.8951
Epoch 99/800
168/168 [==============================] - 1s 5ms/step - loss: 0.4501 - accuracy: 0.8612 - val_loss: 0.3379 - val_accurac
y: 0.9003
Epoch 100/800
168/168 [==============================] - 1s 5ms/step - loss: 0.4159 - accuracy: 0.8679 - val_loss: 0.3347 - val_accurac
y: 0.8936
Epoch 101/800
168/168 [==============================] - 1s 4ms/step - loss: 0.4465 - accuracy: 0.8583 - val_loss: 0.3053 - val_accurac
y: 0.9122
Epoch 102/800
```

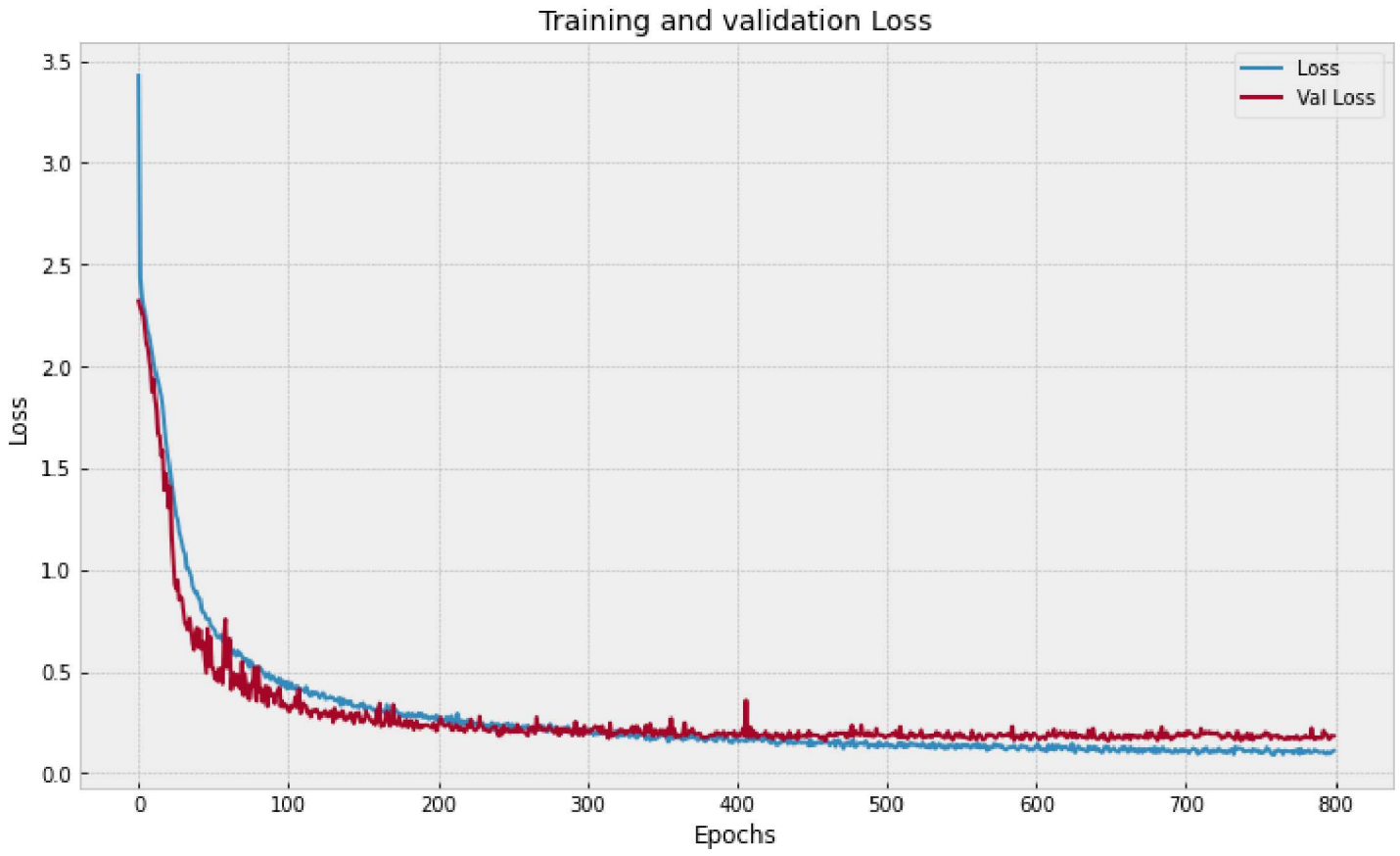
```
y: 0.9568
Epoch 736/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1236 - accuracy: 0.9643 - val_loss: 0.2033 - val_accurac
y: 0.9568
Epoch 737/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1249 - accuracy: 0.9617 - val_loss: 0.1741 - val_accurac
y: 0.9621
Epoch 738/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1220 - accuracy: 0.9587 - val_loss: 0.1600 - val_accurac
y: 0.9680
Epoch 739/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1199 - accuracy: 0.9652 - val_loss: 0.1839 - val_accurac
y: 0.9606
Epoch 740/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1074 - accuracy: 0.9656 - val_loss: 0.1776 - val_accurac
y: 0.9583
Epoch 741/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1099 - accuracy: 0.9667 - val_loss: 0.1719 - val_accurac
y: 0.9658
Epoch 742/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1129 - accuracy: 0.9630 - val_loss: 0.1990 - val_accurac
y: 0.9591
Epoch 743/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1141 - accuracy: 0.9619 - val_loss: 0.1714 - val_accurac
y: 0.9650
Epoch 744/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1115 - accuracy: 0.9632 - val_loss: 0.1672 - val_accurac
y: 0.9650
Epoch 745/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1037 - accuracy: 0.9708 - val_loss: 0.1957 - val_accurac
y: 0.9613
Epoch 746/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1216 - accuracy: 0.9632 - val_loss: 0.1642 - val_accurac
y: 0.9717
Epoch 747/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1145 - accuracy: 0.9637 - val_loss: 0.1797 - val_accurac
y: 0.9635
Epoch 748/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1119 - accuracy: 0.9626 - val_loss: 0.1813 - val_accurac
y: 0.9606
Epoch 749/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1162 - accuracy: 0.9628 - val_loss: 0.1847 - val_accurac
y: 0.9628
Epoch 750/800
168/168 [==============================] - 2s 11ms/step - loss: 0.1015 - accuracy: 0.9691 - val_loss: 0.1769 - val_accura
cy: 0.9598
Epoch 751/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1078 - accuracy: 0.9658 - val_loss: 0.1775 - val_accurac
y: 0.9606
Epoch 752/800
168/168 [==============================] - 1s 5ms/step - loss: 0.0996 - accuracy: 0.9686 - val_loss: 0.1903 - val_accurac
y: 0.9606
Epoch 753/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1070 - accuracy: 0.9647 - val_loss: 0.1845 - val_accurac
y: 0.9613
Epoch 754/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1082 - accuracy: 0.9667 - val_loss: 0.2023 - val_accurac
y: 0.9554
Epoch 755/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1285 - accuracy: 0.9602 - val_loss: 0.2007 - val_accurac
y: 0.9591
Epoch 756/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1184 - accuracy: 0.9639 - val_loss: 0.1842 - val_accurac
y: 0.9628
Epoch 757/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1174 - accuracy: 0.9652 - val_loss: 0.1937 - val_accurac
y: 0.9613
Epoch 758/800
168/168 [==============================] - 1s 5ms/step - loss: 0.0981 - accuracy: 0.9699 - val_loss: 0.1672 - val_accurac
y: 0.9665
Epoch 759/800
168/168 [==============================] - 1s 4ms/step - loss: 0.0919 - accuracy: 0.9701 - val_loss: 0.1839 - val_accurac
y: 0.9688
Epoch 760/800
168/168 [==============================] - 1s 4ms/step - loss: 0.0911 - accuracy: 0.9725 - val_loss: 0.1850 - val_accurac
y: 0.9665
Epoch 761/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1127 - accuracy: 0.9645 - val_loss: 0.1840 - val_accurac
y: 0.9583
Epoch 762/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1140 - accuracy: 0.9671 - val_loss: 0.1894 - val_accurac
y: 0.9591
Epoch 763/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1173 - accuracy: 0.9643 - val_loss: 0.1713 - val_accurac
y: 0.9680
Epoch 764/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1101 - accuracy: 0.9647 - val_loss: 0.1814 - val_accurac
y: 0.9621
Epoch 765/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1099 - accuracy: 0.9669 - val_loss: 0.1781 - val_accurac
y: 0.9643
Epoch 766/800
168/168 [==============================] - 1s 5ms/step - loss: 0.0930 - accuracy: 0.9715 - val_loss: 0.1735 - val_accurac
y: 0.9628
Epoch 767/800
```

```
168/168 [==============================] - 1s 5ms/step - loss: 0.1128 - accuracy: 0.9652 - val_loss: 0.1678 - val_accurac
y: 0.9628
Epoch 768/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1131 - accuracy: 0.9660 - val_loss: 0.1695 - val_accurac
y: 0.9606
Epoch 769/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1081 - accuracy: 0.9669 - val_loss: 0.1804 - val_accurac
y: 0.9591
Epoch 770/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1099 - accuracy: 0.9661 - val_loss: 0.1694 - val_accurac
y: 0.9673
Epoch 771/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1069 - accuracy: 0.9650 - val_loss: 0.1771 - val_accurac
y: 0.9643
Epoch 772/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1106 - accuracy: 0.9641 - val_loss: 0.1729 - val_accurac
y: 0.9628
Epoch 773/800
168/168 [==============================] - 1s 5ms/step - loss: 0.0979 - accuracy: 0.9680 - val_loss: 0.1780 - val_accurac
y: 0.9621
Epoch 774/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1145 - accuracy: 0.9645 - val_loss: 0.1693 - val_accurac
y: 0.9665
Epoch 775/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1068 - accuracy: 0.9665 - val_loss: 0.1857 - val_accurac
y: 0.9598
Epoch 776/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1205 - accuracy: 0.9643 - val_loss: 0.1787 - val_accurac
y: 0.9621
Epoch 777/800
168/168 [==============================] - 1s 5ms/step - loss: 0.0953 - accuracy: 0.9691 - val_loss: 0.1652 - val_accurac
y: 0.9680
Epoch 778/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1229 - accuracy: 0.9652 - val_loss: 0.1703 - val_accurac
y: 0.9665
Epoch 779/800
168/168 [==============================] - 1s 5ms/step - loss: 0.0991 - accuracy: 0.9695 - val_loss: 0.1778 - val_accurac
y: 0.9680
Epoch 780/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1097 - accuracy: 0.9684 - val_loss: 0.1746 - val_accurac
y: 0.9650
Epoch 781/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1034 - accuracy: 0.9673 - val_loss: 0.1868 - val_accurac
y: 0.9613
Epoch 782/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1080 - accuracy: 0.9699 - val_loss: 0.1757 - val_accurac
y: 0.9643
Epoch 783/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1167 - accuracy: 0.9656 - val_loss: 0.1784 - val_accurac
y: 0.9650
Epoch 784/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1016 - accuracy: 0.9663 - val_loss: 0.1741 - val_accurac
y: 0.9643
Epoch 785/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1031 - accuracy: 0.9669 - val_loss: 0.2221 - val_accurac
y: 0.9524
Epoch 786/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1051 - accuracy: 0.9665 - val_loss: 0.1816 - val_accurac
y: 0.9635
Epoch 787/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1177 - accuracy: 0.9656 - val_loss: 0.1666 - val_accurac
y: 0.9650
Epoch 788/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1124 - accuracy: 0.9654 - val_loss: 0.1779 - val_accurac
y: 0.9613
Epoch 789/800
168/168 [==============================] - 2s 11ms/step - loss: 0.1123 - accuracy: 0.9650 - val_loss: 0.1721 - val_accura
cy: 0.9650
Epoch 790/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1036 - accuracy: 0.9684 - val_loss: 0.1779 - val_accurac
y: 0.9613
Epoch 791/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1049 - accuracy: 0.9680 - val_loss: 0.1733 - val_accurac
y: 0.9591
Epoch 792/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1062 - accuracy: 0.9656 - val_loss: 0.1732 - val_accurac
y: 0.9621
Epoch 793/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1087 - accuracy: 0.9682 - val_loss: 0.2100 - val_accurac
y: 0.9554
Epoch 794/800
168/168 [==============================] - 1s 5ms/step - loss: 0.0984 - accuracy: 0.9693 - val_loss: 0.1946 - val_accurac
y: 0.9591
Epoch 795/800
168/168 [==============================] - 1s 4ms/step - loss: 0.0962 - accuracy: 0.9708 - val_loss: 0.1935 - val_accurac
y: 0.9583
Epoch 796/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1076 - accuracy: 0.9680 - val_loss: 0.1726 - val_accurac
y: 0.9621
Epoch 797/800
168/168 [==============================] - 1s 4ms/step - loss: 0.0982 - accuracy: 0.9688 - val_loss: 0.1669 - val_accurac
y: 0.9643
Epoch 798/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1003 - accuracy: 0.9680 - val_loss: 0.1867 - val_accurac
y: 0.9665
```

```
Epoch 799/800
168/168 [==============================] - 1s 4ms/step - loss: 0.1096 - accuracy: 0.9637 - val_loss: 0.1824 - val_accurac
y: 0.9628
Epoch 800/800
168/168 [==============================] - 1s 5ms/step - loss: 0.1129 - accuracy: 0.9639 - val_loss: 0.1847 - val_accurac
y: 0.9658
Model Fitted
```

In [10]:
```
summarize_diagnostics(history)
```





We can see that we are getting a flat line around 600-800 epochs with accuracy greater than 95%, and both Validation and Training accuracy and loss are close, so there is no underfitting and overfitting, but val_loss is increasing after some 700 epochs, so we should use epochs between 600 and 700. So, this model is good, so we can train our model with these parameters on complete data_train set.

Below are the final functions used to Load, Preprocess amd Train our full data.

In [12]:
```
def final_training_complete(dataX, dataY):
    model = define_model(learn_rate = 0.0007)
    trainX, trainY = dataX, dataY

    # fit model
    history = model.fit(trainX, trainY, epochs = 650, batch_size = 32, verbose=0)
    print("Model Fitted")
    return model, history
```

In [13]:
```
def load_full_dataset():
    # Load dataset
    data_rgb = []
    kernel = np.ones((4,4),np.uint8)
    for i in range(train_data_length):
        data_rgb.append(data_train[:,i].reshape(300,300))
        data_rgb[i] = cv2.medianBlur(data_rgb[i], 3)
        data_rgb[i] = cv2.morphologyEx(data_rgb[i], cv2.MORPH_OPEN, kernel)
        data_rgb[i] = cv2.resize(data_rgb[i], (50,50), interpolation=cv2.INTER_AREA)
    data_rgb = np.array(data_rgb)
    print(data_rgb.shape)
```

```
    # reshape dataset to have a single channel
    data_rgb = data_rgb.reshape((data_rgb.shape[0], 50,50, 1))

    # one hot encode target values
    labels = to_categorical(labels_train)
    print(data_rgb.shape, labels.shape)
    return data_rgb, labels
```

In [14]:
```
def prep_final_pixels(data):
    # convert from integers to floats
    data_norm = data.astype('float32')

    # he_uniformize to range 0-1
    data_norm = (data_norm) / 255.0

    # return he_uniformized images
    return data_norm
```

In this cell all final functions are called to train our dataset on all images.

In [15]:
```
data, target = load_full_dataset()

# prepare pixel data
data = prep_final_pixels(data)

# training the model
model, history = final_training_complete(data, target)
```

```
(6720, 50, 50)
(6720, 50, 50, 1) (6720, 10)
Model Fitted
```

In [16]:
```
plt.figure(figsize=(12,7))
plt.plot(history.history['accuracy'])
plt.plot(history.history['loss'])
plt.title('Training accuracy and Loss')
plt.ylabel('Accuracy and Loss')
plt.xlabel('epoch')
plt.legend(['Accuracy', 'Loss'], loc='center right')
plt.show()
```

In [17]:
```
# model.save("Final_CNN")
```

```
2022-04-20 14:16:39.478446: W tensorflow/python/util/util.cc:348] Sets are not currently considered sequences, but this m
ay change in the future, so consider avoiding using them.
INFO:tensorflow:Assets written to: Final_CNN/assets
```

Finally, we saved our model as "Final_CNN" and will use this trained model in our Easy_Test and Hard_Test files.