Import Libraries

```
#Import all libraries
#Install EMIST library, import datasets of letters, Matplotlib
!pip3 install emnist
from emnist import list_datasets
from emnist import extract_training_samples
import matplotlib as mpl
import os
import numpy as np  #linear algebra
import pandas as pd
from pandas_profiling import ProfileReport
import matplotlib.pyplot as plt
%matplotlib inline
import torch
import torchvision
import tensorflow as tf
import torch
import torch.nn as nn
print(tf.__version__)

import torch
import cv2
import torchvision.transforms as transforms
from tensorflow.keras.layers import *
from tensorflow.keras.models import *
import tensorflow_datasets as tfds
# import numpy as np
# import torch.nn.functional as F
# from torchvision.datasets import EMNIST
# from torch.utils.data import DataLoader
# import torchvision.transforms as tt
# from torch.utils.data import random_split
# from torchvision.utils import make_grid
# from tensorflow.keras.datasets import mnist
# from tensorflow.keras.utils import to_categorical
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import Conv2D
# from tensorflow.keras.layers import MaxPooling2D
# from tensorflow.keras.layers import Dense
# from tensorflow.keras.layers import Flatten
# from tensorflow.keras.optimizers import SGD

from PIL import Image
!pip install pyyaml h5py
import os

# from tensorflow import keras
# !pip install extra_keras_datasets
# from extra_keras_datasets import emnist

!pip install wandb
import wandb
from wandb.keras import WandbCallback
wandb.login()
```

```
!pip install tqdm

from numpy import argmax
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model

!pip install tqdm
from tqdm import tqdm

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

matplotlib.rcParams['figure.facecolor'] = '#ffffff'
```

```
one-any.whl (7.3 kB)
 tqdm in /usr/local/lib/python3.7/dist-packages (from emnist) (4.63.0)
 requests in /usr/local/lib/python3.7/dist-packages (from emnist) (2.23.0)
 numpy in /usr/local/lib/python3.7/dist-packages (from emnist) (1.21.5)
 idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->emnist) (2.10)
 certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->emnist) (2021.10
 urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from reques
 chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->emnist) (3.0.4)
 emnist
0.0

 pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
 h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
 numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.21.5)
 cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)

2.py3-none-any.whl (1.7 MB)
████████| 1.7 MB 3.9 MB/s
.0
.4.0-py2.py3-none-any.whl (9.0 kB)
 python-dateutil>=2.6.1 in /usr/local/lib/python3.7/dist-packages (from wandb) (2.8.2)
```

```
#Download dataset and it is 536 MB
list_datasets()
```

```
Downloading emnist.zip: 43.9MB [00:01, 40.5MB/s]
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-2-aa4b3a38056e> in <module>()
      1 #Download dataset and it is 536 MB
----> 2 list_datasets()

                    4 frames
/usr/local/lib/python3.7/dist-packages/urllib3/response.py in stream(self, amt,
decode_content)
    493                 yield line
    494         else:
--> 495             while not is_fp_closed(self._fp):
    496                 data = self.read(amt=amt, decode_content=decode_content)
    497

KeyboardInterrupt:

SEARCH STACK OVERFLOW
████████| 63 kB 1.3 MB/s
```

```
images, labels = extract_training_samples('byclass')
images.shape
```

```
 urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from reques
```

GPU Running

```
packages: pathtools
```

```
# Get the GPU device name.
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
```

```
def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        """Number of batches"""
        return len(self.dl)
```

```
# Returns current working runtype

device = get_default_device()
device
```

DataLoader Preparation

```
dataset = EMNIST(root="data/", split="byclass", download=True, train=True,
                transform=tt.Compose([
                    lambda img: tt.functional.rotate(img, -90),
                    lambda img: tt.functional.hflip(img),
                    tt.ToTensor()
                ]))
random_seed = 50
torch.manual_seed(random_seed);

val_size = 50000
train_size = len(dataset) - val_size
```

```
train_ds, val_ds = random_split(dataset, [train_size, val_size])
len(train_ds), len(val_ds)
```

```
batch_size = 400

train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=4, pin_memory=True)
val_dl = DataLoader(val_ds, batch_size*2, num_workers=4, pin_memory=True)
```

```
# Lets see a batch of images

def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(12, 12))
        fig, ax = plt.subplots(figsize=(12, 12))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images, nrow=20).permute(1, 2, 0))
        break

show_batch(train_dl)
```

```
train_dl = DeviceDataLoader(train_dl, device)
val_dl = DeviceDataLoader(val_dl, device)
```

```
show_batch(train_dl_1)
```

MNIST

[MNIST recognition refernce]

```
# save the final model to file
# load train and test dataset
def load_dataset(data):
    # load dataset
    (trainX, trainY), (testX, testY) = data
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm

# define cnn model
def define_model():
    model = Sequential()
 #(2,2) size of pooling area for max pooling    #(3,3) convolution kernel size  #32 number of convo
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(2
    model.add(MaxPooling2D((2, 2)))
    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(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(62, activation='softmax'))
    # compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# run the test harness for evaluating a model
def run_test_harness(data):
    # load dataset

    trainX, trainY, testX, testY = load_dataset(data)
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # define model
    model = define_model()
    # fit model
    model.fit(trainX, trainY, epochs=10, batch_size=32, verbose=0)
    # save model
    model.save('/content/sample_data/final_model.h5')



# entry point, run the test harness
data=mnist.load_data()
run_test_harness(data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-61db1c596acd> in <module>()
      1 # entry point, run the test harness
----> 2 data=mnist.load_data()
      3 run_test_harness(data)

NameError: name 'mnist' is not defined
```

SEARCH STACK OVERFLOW

SAVE MODEL

```
# load and prepare the image
def load_image(filename):
    # load the image
    img = load_img(filename, color_mode='grayscale', target_size=(28, 28))
    # convert to array
    img = img_to_array(img)
    # reshape into a single sample with 1 channel
    img = img.reshape(1, 28, 28, 1)
    # prepare pixel data
    img = img.astype('float32')
    img = img / 255.0
    return img

# load an image and predict the class
def run_example():
    # load the image
    img = load_image('/content/sample_data/img.png')
    # load model
    model = load_model('/content/sample_data/final_model.h5')
    # predict the class
    predict_value = model.predict(img)
    digit = argmax(predict_value)
    print(digit)

# entry point, run the example



print('RESULT:')
run_example()
```

EMNIST DATASET RECOGNITION

[Reference](#)
[EMNIST Classification](#)
[EMNIST Classification NB](#)

[research Paper](#)
[CNN Theory](#)

[SAVE AND RESTORE MODEL FROM WANDB](#)

[github alphabet recognition](#)

TO FIND OUT WHAT IS TESTLOADER AND HOW TO GIVE FRAME BY FRAME IN VALIDATION DATA? or emnist classification predicts on one image

RESEARCH FOR AROUBA AND SAIMA

Points to note:

- We are using bymerge variant of EMNIST dataset. Here labels like j, o, i etc which look like J, O, I are merged.
- The EMNIST images provided here are inverted horizontally and rotated 90 anti-clockwise. For the ease of experimentation, we don't want to use it in this configuration. Thus we will rotate the image back by 90 deg anti-clockwise.
- We have a total of 814255 images. They are 28x28 pixels in resolution with only one channel.
- We have 697932 images as training data.
- We have 116323 images as testing data.
- We have 47 classes as shown:

```
# Gather EMNIST/bymerge dataset
train_ds, validation_ds = tfds.load(
    "emnist/bymerge",
    split=["train[:85%]", "train[85%:]"],
    as_supervised=True
)
```

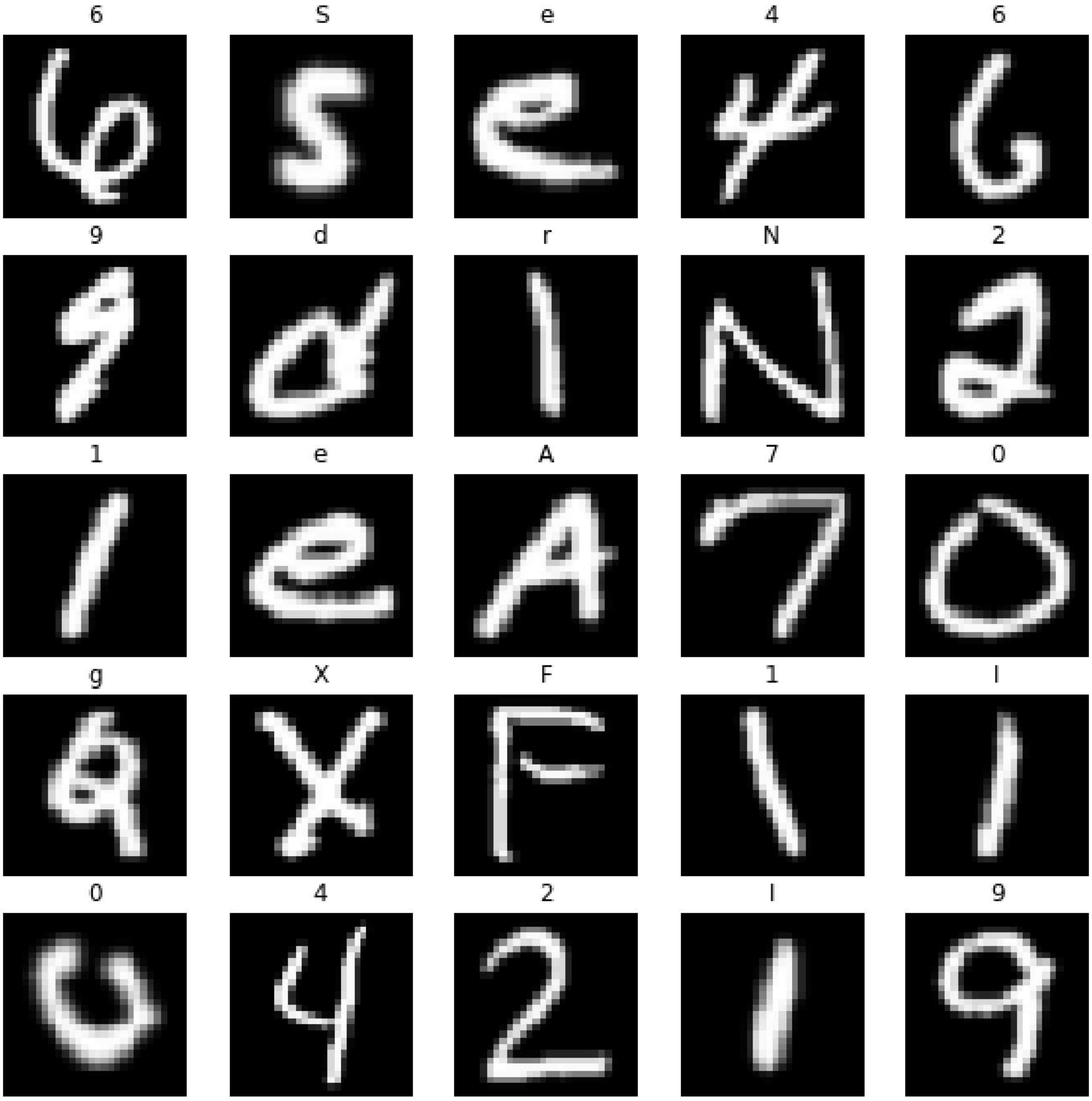
```
  #47 classes

LABELS = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
          'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
          'a', 'b', 'd', 'e', 'f', 'g', 'h', 'n', 'q', 'r', 't']
len(LABELS)
```

```
Downloading and preparing dataset emnist/bymerge/3.0.0 (download: 535.73 MiB, generated: Unkno
Dl Completed...: 100%       1/1 [00:17<00:00, 12.60s/ url]

Dl Size...: 100%       535/535 [00:17<00:00, 51.09 MiB/s]

Extraction completed...: 100%       1/1 [00:17<00:00, 17.55s/ file]




Extraction completed...: 100%       4/4 [00:12<00:00, 3.57s/ file]

Shuffling and writing examples to /root/tensorflow_datasets/emnist/bymerge/3.0.0.incompleteGZV
100%                                          697931/697932 [00:03<00:00, 288506.06 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/emnist/bymerge/3.0.0.incompleteGZV
100%                                          116322/116323 [00:00<00:00, 230190.69 examples/s]
Dataset emnist downloaded and prepared to /root/tensorflow_datasets/emnist/bymerge/3.0.0. Subs
47
```

```
AUTO = tf.data.experimental.AUTOTUNE
BATCH_SIZE = 256

## We are transposing to rotate the image by 90 deg clockwise making the images human friendly.
def transpose_and_flatten(image, label=None):
  image = tf.image.convert_image_dtype(image, dtype=tf.float32) # scale image pixels to [0,1]
  image = tf.transpose(image, [1,0,2]) # transpose to get human friendly image, since rotation
  image = tf.reshape(image, shape=(784,)) # permutation invariant or flatten

  label = tf.one_hot(label, depth=len(LABELS)) # one hot encode label

  return image, label

trainloader = (
    train_ds
    .shuffle(1024)
    .map(transpose_and_flatten, num_parallel_calls=AUTO)
    .batch(BATCH_SIZE)
    .prefetch(AUTO)
)

testloader = (
    validation_ds
    .map(transpose_and_flatten, num_parallel_calls=AUTO)
    .batch(BATCH_SIZE)
    .prefetch(AUTO)
)


testloader
```

```
<PrefetchDataset element_spec=(TensorSpec(shape=(None, 784), dtype=tf.float32, name=None), Ten
```

```
imgs, labels = next(iter(trainloader))

plt.figure(figsize=(10, 10))
for n in range(25):
    ax = plt.subplot(5, 5, n+1)
    plt.imshow(tf.reshape(imgs[n], shape=(28,28)), cmap='gray')
    plt.title(LABELS[np.argmax(labels[n])])
    plt.axis('off')
```



```
type(imgs[0])
```

```
tensorflow.python.framework.ops.EagerTensor
```

```
def DenseModel():
  inputs = Input(shape=(784,))
  x = Dense(256, activation='relu')(inputs)
  x = Dense(128, activation='relu')(x)
  outputs = Dense(len(LABELS), activation='softmax')(x)  #Index recieved from LABELS list

  return Model(inputs=inputs, outputs=outputs)
```

```
early_stopper = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=5, verbose=0, mode='auto', restore_best_weights=True
)


# initialize wandb run
# do not change entity
wandb.init(entity='iit-bhu', project='emnist')

# hyperparameters
config = wandb.config
config.epochs = 70
config.learning_rate = 0.001

# model
tf.keras.backend.clear_session()
model = DenseModel()

# optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=config.learning_rate)

# compile
model.compile(optimizer, 'categorical_crossentropy', metrics=['acc'])

# train
model.fit(trainloader,
          epochs=config.epochs,
          validation_data=testloader,
          callbacks=[WandbCallback(),
                     early_stopper])
```

```
wandb: Currently logged in as: insi29 (use `wandb login --relogin` to force relogin)
Tracking run with wandb version 0.12.11
Run data is saved locally in /content/wandb/run-20220304_152316-1n4moujo
Syncing run fluent-wave-243 to Weights & Biases (docs)
Epoch 1/70
2318/2318 [==============================] - 157s 67ms/step - loss: 0.6043 - acc: 0.8129 - val
Epoch 2/70
2318/2318 [==============================] - 160s 69ms/step - loss: 0.3935 - acc: 0.8655 - val
Epoch 3/70
2318/2318 [==============================] - 157s 68ms/step - loss: 0.3550 - acc: 0.8757 - val
Epoch 4/70
2318/2318 [==============================] - 156s 67ms/step - loss: 0.3326 - acc: 0.8820 - val
Epoch 5/70
2318/2318 [==============================] - 156s 67ms/step - loss: 0.3167 - acc: 0.8864 - val
Epoch 6/70
2318/2318 [==============================] - 166s 72ms/step - loss: 0.3043 - acc: 0.8898 - val
Epoch 7/70
2318/2318 [==============================] - 160s 69ms/step - loss: 0.2947 - acc: 0.8927 - val
Epoch 8/70
2318/2318 [==============================] - 174s 75ms/step - loss: 0.2865 - acc: 0.8948 - val
Epoch 9/70
2318/2318 [==============================] - 168s 73ms/step - loss: 0.2789 - acc: 0.8969 - val
Epoch 10/70
2318/2318 [==============================] - 182s 79ms/step - loss: 0.2730 - acc: 0.8984 - val
<keras.callbacks.History at 0x7fd2f0efba10>
```

SAVE AND RESTORE FROM WANDB

```
api = wandb.Api()

run = api.run("iit-bhu/emnist/1n4moujo")

model.save(os.path.join(wandb.run.dir, "model.h5"))

# Save a model file manually from the current directory:
wandb.save('model.h5')
```

```
['/content/wandb/run-20220304_152316-1n4moujo/files/model.h5']
```

```
# restore the model file "model.h5" from a specific run by user "lavanyashukla"
# in project "save_and_restore" from run "10pr4joa"
best_model = wandb.restore('model.h5', run_path="iit-bhu/emnist/1n4moujo")

# use the "name" attribute of the returned object
# if your framework expects a filename, e.g. as in Keras
model.load_weights(best_model.name)
```

SAVE AND LOAD MODEL

```
model.save('/content/sample_data/final_model.h5')
model = load_model('/content/sample_data/final_model.h5')
```

SAVE AND LOAD MODEL FROM DRIVE

```
!pip install pyyaml h5py
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (3.13)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h
Mounted at /content/gdrive
```

```
model.save('/content/gdrive/MyDrive/FYP@10Pearls/final.h5')
```

```
model_best=load_model('/content/gdrive/MyDrive/FYP@10Pearls/final.h5')
```

Prediciton of Test Data

```
y_test = []    #Labels already known/expected
y_preds = []   #Labels predicted # o/p index
data_p=[]
for imgs, labels in tqdm(testloader):
      y_test.extend(np.argmax(labels, axis=1))
      y_pred = model_best.predict(imgs)
      y_preds.extend(np.argmax(y_pred, axis=1))
for i in (y_preds):
  data_p.append(LABELS[i])
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (4.63.0)
100%|██████████| 409/409 [01:03<00:00,  6.41it/s]
```

Convert Image to Tensor

```
import cv2
test=[]
def infer_prec(img, img_size):
    img = tf.expand_dims(img, -1)          # from 28 x 28 to 28 x 28 x 1
    img = tf.divide(img, 255)              # normalize
    img = tf.image.resize(img,             # resize acc to the input
             [img_size, img_size])
    img = tf.reshape(img,                  # reshape to add batch dimension
            [784])
    return img

img = cv2.imread('/content/sample_data/sample_image.png', 0)   # read image as gray scale
print(img.shape)   # (720, 1280)

img = infer_prec(img, 28)  # call preprocess function
print(img.shape)
img=tf.image.convert_image_dtype(
    img, dtype=tf.float32, name=None
)
test.append(img)
test=np.array(test)
```

```
(1480, 1490)
(784,)
```

```
plt.imshow(tf.reshape(img, shape=(28,28)), cmap='gray')
```

```
    <matplotlib.image.AxesImage at 0x7fd2ef57c190>
```

```
type(test)

    numpy.ndarray
```

```
tested=tf.convert_to_tensor(test, dtype=tf.float32)
```

```
tested2=tf.convert_to_tensor(img_l, dtype=tf.float32)
```

```
type(tested)

    tensorflow.python.framework.ops.EagerTensor
```

```
y_pred = model_best.predict(tested)
#y_pred  # probabilities
```

```
# get predicted label
pred=tf.argmax(y_pred, axis=-1).numpy() # array([8], dtype=int64)
```

```
pred

    array([7])
```

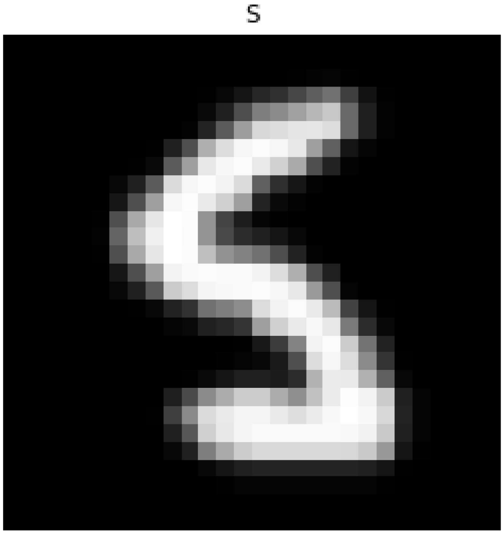
```
LABELS[pred[0]]

    '7'
```

```
df=pd.DataFrame({'Expected Label':y_test,'Predicted Label':y_preds,'LABELS':data_p})
df.head(10)
```

	Expected Label	Predicted Label	LABELS
0	28	28	S
1	6	6	6
2	1	1	1
3	25	33	X
4	2	2	2
5	1	1	1
6	0	0	0
7	9	9	9
8	12	12	C
9	5	5	5

```
plt.figure(figsize=(10, 10))
ax = plt.subplot(2, 2, 1)
plt.imshow(tf.reshape(imgs2[0], shape=(28,28)), cmap='gray')
plt.title(LABELS[y_preds[0]])
plt.axis('off')
```

```
(-0.5, 27.5, 27.5, -0.5)
```



```
DF=pd.DataFrame({'Labels':LABELS})
DF
```

	Labels
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	G
17	H
18	I
19	J
20	K
21	L
22	M
23	N
24	O
25	P
26	Q
27	R

```
LABELS[28]
```

```
'S'
```

30 U

```
y_preds[0]
```

```
28
```

33 X

```
y_test[0]
```

```
28
```

Updates in Code

```
38      a
```

```
def cv2_imshow(img):
    plt.imshow(img,cmap='gray')
def predict_image(img):
  image = cv2.imread(img)
  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  blurred = cv2.GaussianBlur(gray, (5, 5), 0)
  edged = cv2.Canny(blurred, 30, 150)
  cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
  cnts = imutils.grab_contours(cnts)
  cnts = sort_contours(cnts, method="left-to-right")[0]
  chars = []
#   print(cnts)

  #cv2_imshow(edged)

  for c in cnts:
    # compute the bounding box of the contour
    (x, y, w, h) = cv2.boundingRect(c)
    # filter out bounding boxes, ensuring they are neither too small
    # nor too large
    if (w >= 5 and w <= 150) and (h >= 15 and h <= 120):
      # extract the character and threshold it to make the character
      # appear as *white* (foreground) on a *black* background, then
      # grab the width and height of the thresholded image
      roi = gray[y:y + h, x:x + w]
      thresh = cv2.threshold(roi, 0, 255,
        cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
      (tH, tW) = thresh.shape
      # if the width is greater than the height, resize along the
      # width dimension
      if tW > tH:
        thresh = imutils.resize(thresh, width=32)
      # otherwise, resize along the height
      else:
        thresh = imutils.resize(thresh, height=32)
      (tH, tW) = thresh.shape
      dX = int(max(0, 32 - tW) / 2.0)
      dY = int(max(0, 32 - tH) / 2.0)
      # pad the image and force 32x32 dimensions
      padded = cv2.copyMakeBorder(thresh, top=dY, bottom=dY,
        left=dX, right=dX, borderType=cv2.BORDER_CONSTANT,
        value=(0, 0, 0))
      padded = cv2.resize(padded, (28, 28))
      padded = padded.astype("float32")
      padded = np.expand_dims(padded, axis=-1)    #originally -1
      chars.append((padded, (x, y, w, h)))
#       cv2_imshow(padded)

    #print("Padded: -")
    #cv2_imshow(padded)
  """
  padded = cv2.copyMakeBorder(blurred, top=dY, bottom=dY,
            left=dX, right=dX, borderType=cv2.BORDER_CONSTANT,
            value=(0, 0, 0))
  padded = cv2.resize(padded, (32, 32))
  """
#  final_image=np.expand_dims(padded,axis=2)
  print("characters ",len(chars))
  boxes = [b[1] for b in chars]
  #cv2_imshow(chars)
  chars = np.array([c[0] for c in chars], dtype="float32")
  # OCR the characters using our handwriting recognition model
#   for c in chars:
  preds = model.predict(chars)
  # define the list of label names
  labelNames = "0123456789"
  labelNames += "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
  labelNames = [l for l in labelNames]
  output=""
  fig=plt.figure()
  for c in range(len(chars)):
    print(c)
   # ax=fig.add_subplot(1,len(chars),c+1)
  #  ax.imshow(chars[c],cmap='gray')  ####################
  #fig.show()

  for (pred, (x, y, w, h)) in zip(preds, boxes):
    i = np.argmax(pred)
  #  print("i is ",i)
   # print("label length ",len(labelNames))
    #print("prediction is ",len(pred))
    prob = pred[i]
    label = labelNames[i]
    output+=label

  return output
predict_image('/content/drive/MyDrive/Kaggle/aro.jpg')
```

NANO NETS

TRAINED DATA WITH MODEL

NANO NET MODEL

```
import requests
import json
url = 'https://app.nanonets.com/api/v2/OCR/Model/866d66dc-69d6-433c-82c8-6123cb2db3b6/LabelFile/'

data = {'file': open('/content/sample_data/opencv_frame_0.png', 'rb')}

response = requests.post(url, auth=requests.auth.HTTPBasicAuth('NlE3KM3daTBYb6TrHNUOJdHLMe1tSNLB',

res=json.loads(response.text)


res["result"][0]["prediction"][0]["ocr_text"]
```