Computer Vision Project 5:
TinyYOLO - framework for Handwriting Recognition

Ian Sinclair
ENCE 3620-1

March 30, 2022
Dr. Mohammad H. Mahoor

Abstract

Considered is an implementation of the tinyYOLOv3 (and tiny-YOLOv4) framework using darkflow. YOLO (you only look once) is an advanced image recognition neural network that is designed to recognize classify and draw bounding boxes around multiple objects in an image simultaneously. Here, we re-train the network to detect handwritten digits on a page. And so this involved defining 36 classes (for each letter in the alphabet and digits $0-9$.) and building a synthesized data set from EMNIST where multiple letters/digits are maintained and labeled on single images. Lastly, the tinyYOLOv3 darkflow framework was used to train the model to detect both the location and label of digits on over 400 images. Unfortunately, training the model was unsuccessful on $tinyYOLOv3$. However, worked on the pre-build library for $tiny - YOLOv4$ made by Darkflow and Roboflow.

1 GPU - Hardware Requirements

Typically, YOLO frameworks including darkflow are run on dedicated NVIDIA GPU cards, this drastically increases training and query time. Here colab was used to remotely connect to a GPU, specifically, a Tesla P100-PCIE was used to train the model (Note, colab does not guarantee the quality of available GPU's and so training speed may vary).

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

2 Installing Darkflow And Tensor With Compatibility

Darkflow implementation of YOLO was retrained on the synthesized data set. For this, it is critical that all libraries packages are compatible with the version of darkflow/tinyYOLO.

```
!pip install tensorflow-gpu==1.15.0
!pip install imageio

# Download and build darkflow (the tensorflow implementation of YOLO)
import os
import pathlib

if "darkflow-master" in pathlib.Path.cwd().parts:
  while "darkflow-master" in pathlib.Path.cwd().parts:
    os.chdir('..')
elif not pathlib.Path("darkflow-master").exists():
  !git clone --depth 1 https://github.com/thtrieu/darkflow.git
  # Compile darkflow
  %cd darkflow
  !python setup.py build_ext --inplace
  # Change darkflow to darkflow-master to distinguish between folder names
  %cd ../
  !mv darkflow darkflow-master
  %cd darkflow-master


# Upload yolo.weights, pre-trained weights file (for YOLO v2) from an external Google drive
weights = 'yolo'
weights_file = weights + '.weights'
if not os.path.exists('weights_file'):
  !gdown --id 0B1tW_VtY7oniTnBYYWdqSHNGSUU
  !mkdir bin
  !mv yolo.weights bin


# Imports
%cd darkflow-master
%tensorflow_version 1.15.0rc2

# For importing/exporting files, working with arrays, etc
import time
import urllib
import numpy as np
import pandas as pd
import imageio

# For actual object detection
import tensorflow as tf
from darkflow.net.build import TFNet
threshold = 0.25

# For drawing onto and plotting images
import matplotlib.pyplot as plt
import cv2
%config InlineBackend.figure_format = 'svg'
```

From here a model pipeline can be constructed by,

```
params = {
    'model': 'cfg/yolo.cfg',
    'load': 'bin/yolo.weights',
    'threshold': threshold,
    'gpu': 1.0
}

# Run the model
tfnet = TFNet(params)
```

3 Testing Darkflow Detection

The above implementation of darkflow has a pre-computed deep neural network for multiclass object detection which also automatically downloads demo images to validate if darkflow was armed correctly, (without incompatibility errors). And so this default test was run to check the model.

```
img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
result = tfnet.return_predict(img)
for i in range(0,len(result)) :
  img = cv2.rectangle(img, (result[i]['topleft']['x'], result[i]['topleft']['y']), (result[i]['bottomright']['x'], result[i]['bottomright']['y']), (0, 255, 0), 7)
  img = cv2.putText(img, result[i]['label'], (result[i]['topleft']['x'], result[i]['topleft']['y']), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0), 2)
plt.imshow(img)
```

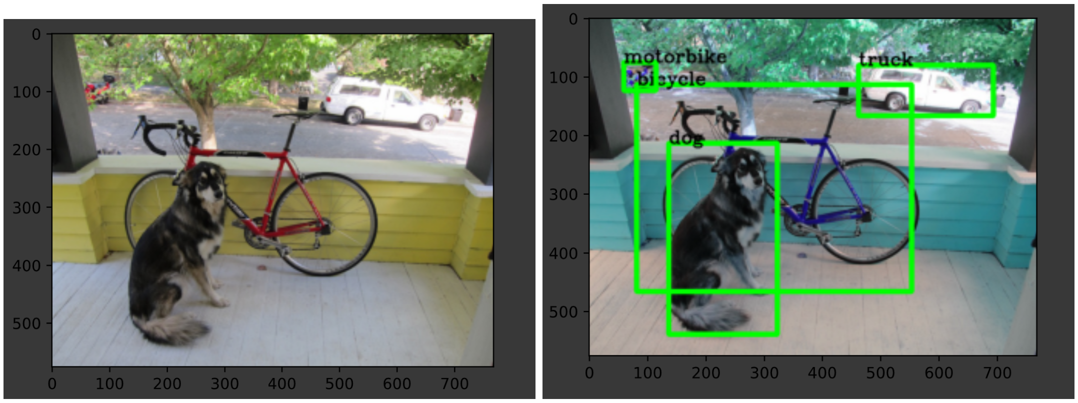


Figure 1: Darkflow - Master Demo of YOLOv3 Framework

4 Synthesizing Dataset EMNIST

EMNIST is a data set that contains 28 × 28 images of both digits and letters. It is an extension created by, Gregory Cohen et. al. of the original NMIST which only had digits from $0 - 9$.
The original (non-synthesized data set) is implemented by pip install,

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml

!pip install emnist
from emnist import list_datasets
```

Also the data set needs to be warmed up,

```
list_datasets()
```

Finally, the raw training and testing images and labels (annotations) can be parsed in.

```
from emnist import extract_training_samples
from emnist import extract_test_samples

test_1, test_labels_1 = extract_test_samples('letters')
train_1, train_labels_1 = extract_training_samples('letters')

train_labels_1 = list(np.asarray(train_labels_1) + 10)
test_labels_1 = list(np.asarray(test_labels_1) + 10)

test_2, test_labels_2 = extract_test_samples('digits')
train_2, train_labels_2 = extract_training_samples('digits')

Test_images, Test_labels = np.concatenate((test_1, test_2), axis=0),np.concatenate((test_labels_1,
     test_labels_2), axis=0)
Train_images, Train_labels = np.concatenate((train_1, train_2), axis=0),np.concatenate((train_labels_1,
      train_labels_2), axis=0)

print(Test_images.shape)
print(Test_labels.shape)

print(Train_images.shape)
print(Train_labels.shape)
```

Raw images in either set are typically bounded around the letter,
Importantly, these images do not contain information about their bounding boxes and also only have a single image per image. And so are initially poor training candidates for the YOLO framework. As a result, we construct a new more compatible data set from the raw EMNIST images.
Here, a larger (200 × 200) black image is created, then randomly selected and randomly sized EMNIST data images are pasted/ overlayed on the blank image to make a larger image with multiple hand written digits.
Critically, by implementing this data set in this way, the original labels from EMNIST are no long compatible (because they do not contain spacial information about where each letter is located in the larger image.) And so

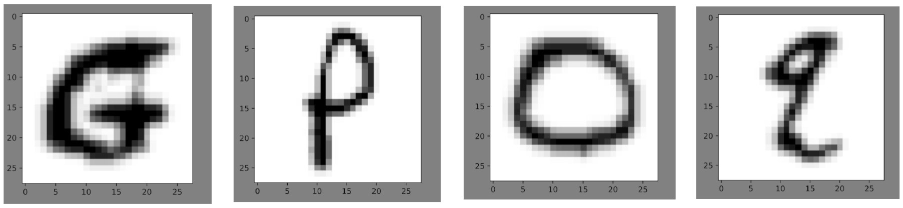


Figure 2: EMNIST Data Image Examples

new annotations/labels are parse where each image contains a non trivial set of the original labels describing both the locations of the bounding boxes around each digit and their label, $1 - 36$.

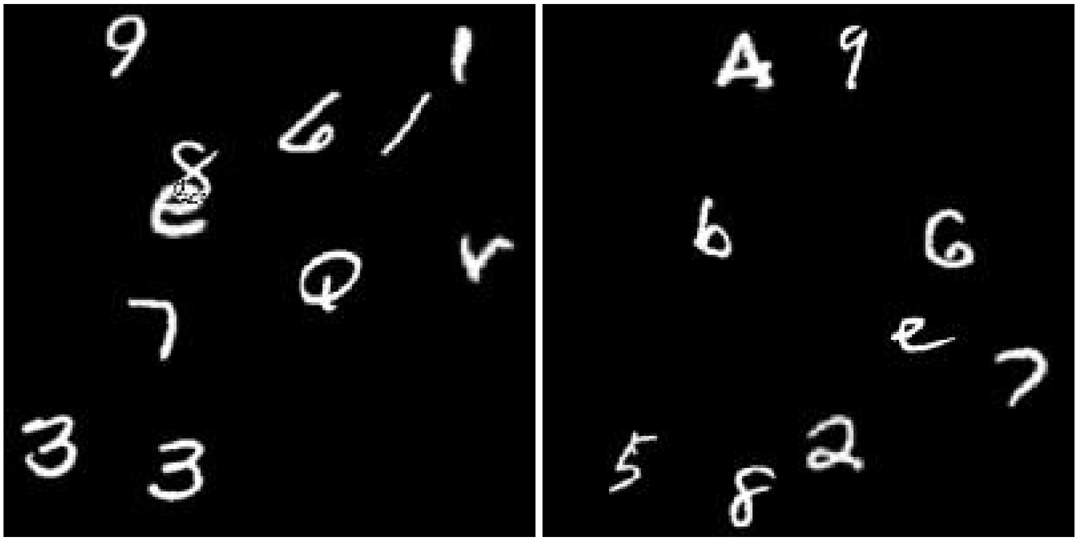


Figure 3: Synthesized Dataset Examples

Additionally, annotations needed to be in a compatible format for Darkflow automatic parsing. Which is .xml web based format. And so annotations where automatically generated to a corresponding file to each image. Each annotation contained information about each digit on the larger image, including bounding box dimensions and digit label.

```
import random
from PIL import Image
from xml.etree import ElementTree as ET

dataset_size = 1
for set_size in range(0,dataset_size) :

  #Blank Image
  temp_img = np.zeros([200,200],dtype=np.uint8)

  index = set_size
  annot = "<annotation>\n\
    <folder>VOC2007</folder>\n\
    <filename>" + str(index) + ".jpg</filename>\n\
    <source>\n\
      <database>EMNIST</database>\n\
      <annotation>EMNIST</annotation>\n\
      <image>EMNIST</image>\n\
      </source>\n\
        <owner>\n\
        <name>Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik</name>\n\
        </owner>\n\
          <size>\n\
            <width>200</width>\n\
            <height>200</height>\n\
            <depth>1</depth>\n\
```

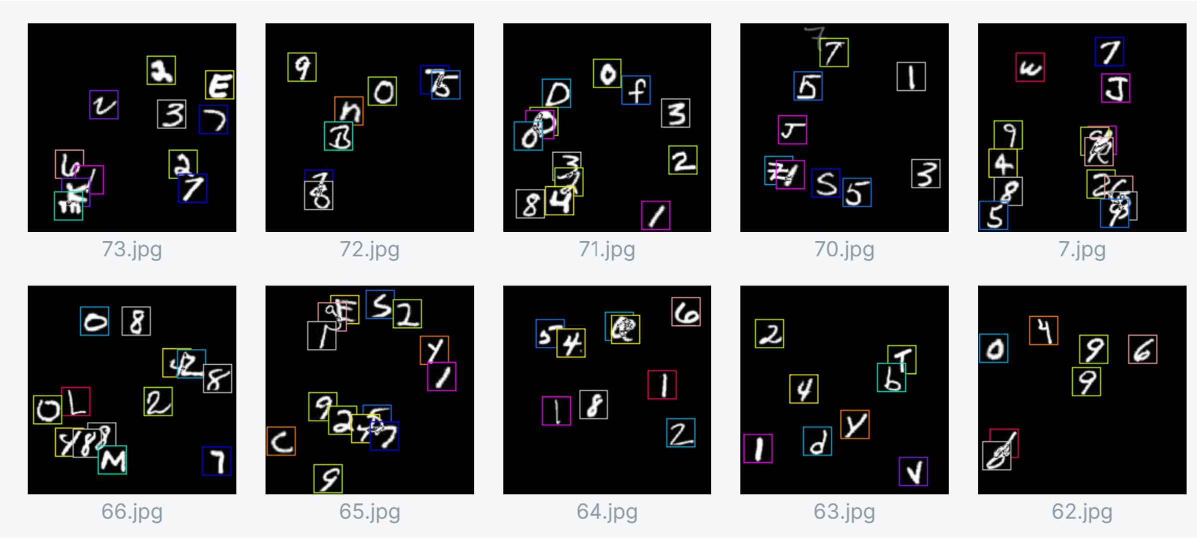


Figure 4: Synthesized Data set with bounding boxes

```
              </size>\n\
              <segmented>0</segmented>"
    stack = random.randint(7, 15)
    for i in range(0 , stack) :
      j = random.randint(0 , len(Train_images) )
      anchor = ( random.randint(0,200 - len(Train_images[j]-1)), random.randint(0,200-len(Train_images[j])-1))
      temp_img[anchor[0]:anchor[0]+len(Train_images[j]) ,anchor[1]:anchor[1]+len(Train_images[j])] += Train_images[j]
      annot += "<object>\n\
                <name>" + str(Train_labels[j]) + "</name>\n\
                <pose>Left</pose>\n\
                <truncated>0</truncated>\n\
                <difficult>0</difficult>\n\
                  <bndbox>\n\
                    <xmin>" + str(anchor[0]) + "</xmin>\n\
                    <ymin>" + str(anchor[1]) + "</ymin>\n\
                    <xmax>" + str(anchor[0] + len(Train_images[j])) + "</xmax>\n\
                    <ymax>" + str(anchor[1] + len(Train_images[j])) + "</ymax>\n\
                  </bndbox>\n\
                </object>\n"
    annot += "</annotation>"

    tree = ET.XML(annot)
    with open("/content/darkflow-master/test/training/annotations/" + str(index) + '.xml', "wb") as f :
      f.write(ET.tostring(tree))

    im = Image.fromarray(temp_img)
    im.save("/content/darkflow-master/test/training/images/"  + str(index) + ".jpg")
    #temp_img = temp_img.paste(img2, (0,0))
```

The dataset was generated with a subset of 40,000 randomly selected EMNIST images to create 200 larger images containing multiple digit objects. And including bounding boxes results in,

5 Training - Original Darkflow $tiny - YOLOv3$

After the data set was initialized, Darkflow's YOLO implementation can be re-trained for the new data, with the designated number of classes.
First, ensure the colab runtime is still connected to a GPU,

```
physical_devices = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

Then, train the model using the file locations that store the data-set

```
with tf.device('/device:GPU:0') :
  !python flow --model cfg/tiny-yolo-voc-3c.cfg --load bin/yolo.weights --train --annotation test/
      training/annotations --dataset test/training/images
```

Unfortunately, this method was incompatible with colabs GPU and so the training time run on CPU exceeded my patience.

6 Training Roboflow Darkflow $tiny - YOLOv4$

An alternative training model was considered by uploading the synthesized dataset to Roboflow and the using the pre-built colab notebook by Darkflow that is directly compatible with Roboflow. This is mainly to test the validity of the data set and I did not write any of the code in the pre-built library. If anything was changed it was just the model parameters.
However, the code in the pre-build notebook follows the same logic as the above implementation.
Critically, this also involves switch darkflow versions to YOLOv4-tiny which is significantly faster than YOLOv3-tiny.
The model was trained for 30 minutes and these were some of the test results. **Note: that labels are numerical,** $0 - 36$**, and so letters are labeled by increasing alphabet from** 10**. So** a **is** 11**, and** z **is** 36.

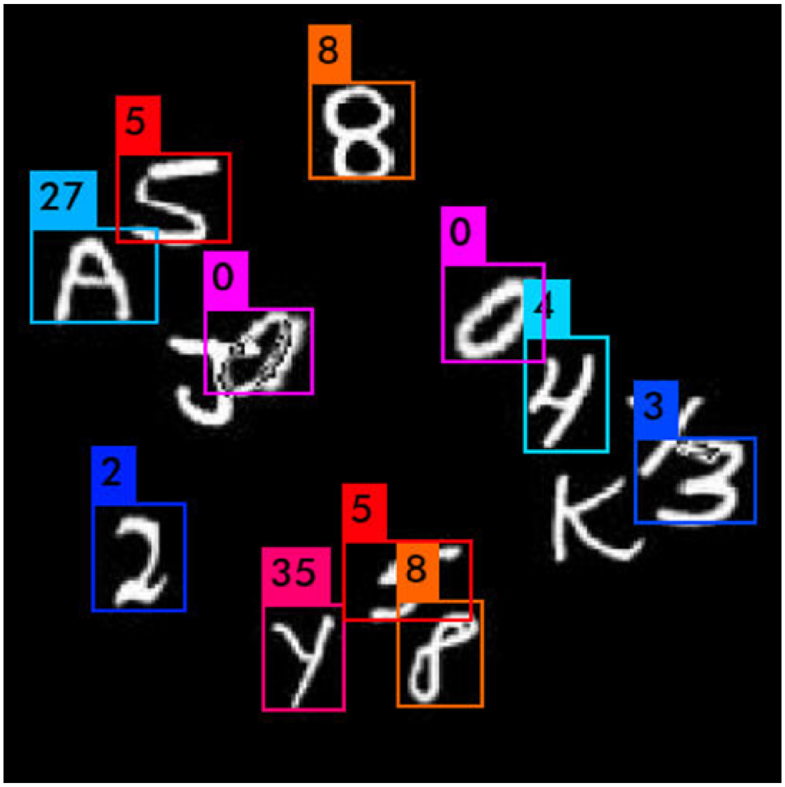
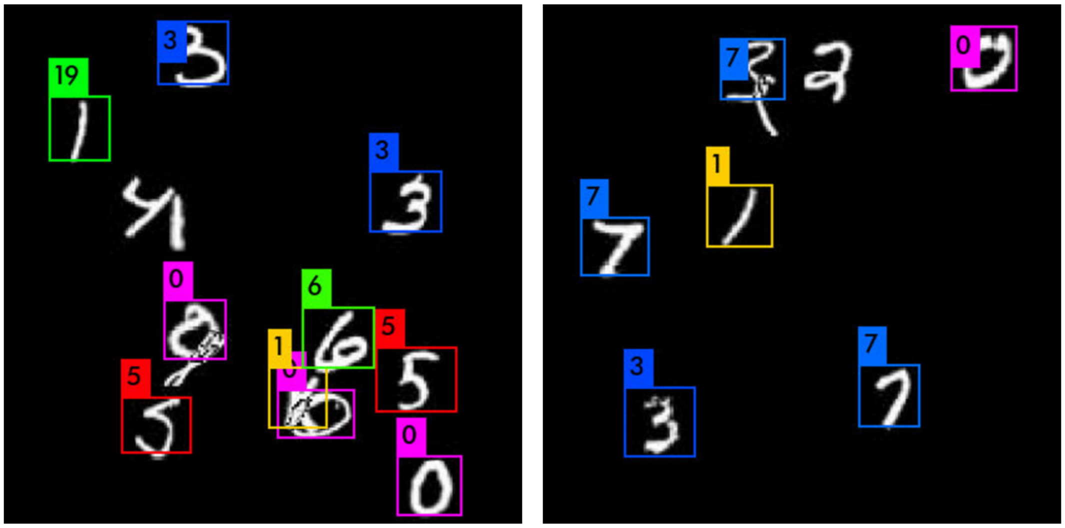


Figure 5: Test Output: tinyTOLOv4 model prediction



7 Appendix

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)

from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')

Imports: Darkflow (YOLO) and Tensorflow

from sklearn import svm
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml
import numpy as np



from google.colab import drive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
from google.colab import drive
drive.mount('/content/drive')

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Install required libraries
#!pip install tensorflow-gpu==1.15.0rc2
!pip install tensorflow-gpu==1.15.0
!pip install imageio
```

```

# Download and build darkflow (the tensorflow implementation of YOLO)
import os
import pathlib

if "darkflow-master" in pathlib.Path.cwd().parts:
  while "darkflow-master" in pathlib.Path.cwd().parts:
    os.chdir('..')
elif not pathlib.Path("darkflow-master").exists():
  !git clone --depth 1 https://github.com/thtrieu/darkflow.git
  # Compile darkflow
  %cd darkflow
  !python setup.py build_ext --inplace
  # Change darkflow to darkflow-master to distinguish between folder names
  %cd ../
  !mv darkflow darkflow-master
  %cd darkflow-master


# Upload yolo.weights, pre-trained weights file (for YOLO v2) from an external Google drive
weights = 'yolo'
weights_file = weights + '.weights'
if not os.path.exists('weights_file'):
  !gdown --id 0B1tW_VtY7oniTnBYYWdqSHNGSUU
  !mkdir bin
  !mv yolo.weights bin


# Imports
%cd darkflow-master
%tensorflow_version 1.15.0rc2

# For importing/exporting files, working with arrays, etc
import time
import urllib
import numpy as np
import pandas as pd
import imageio

# For actual object detection
import tensorflow as tf
from darkflow.net.build import TFNet
threshold = 0.25

# For drawing onto and plotting images
import matplotlib.pyplot as plt
import cv2
%config InlineBackend.figure_format = 'svg'

Define a Model:

params = {
    'model': 'cfg/yolo.cfg',
    'load': 'bin/yolo.weights',
    'threshold': threshold,
    'gpu': 0.0
}

# Run the model
tfnet = TFNet(params)

Test a image and make a bounding box

import matplotlib.pyplot as plt
import matplotlib.image as mp
import PIL as pl
import cv2

image_path = '/content/darkflow-master/sample_img/sample_dog.jpg'
image =  mp.imread(image_path)
plt.imshow(image)

Query Image
```

```

img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
result = tfnet.return_predict(img)
img.shape


Extract Data

tl = (result[1]['topleft']['x'], result[1]['topleft']['y'])
br = (result[1]['bottomright']['x'], result[1]['bottomright']['y'])
label = result[1]['label']

print(label)

Display Box

for i in range(0,len(result)) :
  img = cv2.rectangle(img, (result[i]['topleft']['x'], result[i]['topleft']['y']), (result[i]['bottomright']['x'], result[i]['bottomright']['y']), (0, 255, 0), 7)
  img = cv2.putText(img, result[i]['label'], (result[i]['topleft']['x'], result[i]['topleft']['y']), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0), 2)
plt.imshow(img)

# EMINST Data

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml

!pip install emnist

from emnist import list_datasets

list_datasets()

from emnist import extract_training_samples
from emnist import extract_test_samples

test_1, test_labels_1 = extract_test_samples('letters')
train_1, train_labels_1 = extract_training_samples('letters')

train_labels_1 = list(np.asarray(train_labels_1) + 10)
test_labels_1 = list(np.asarray(test_labels_1) + 10)

test_2, test_labels_2 = extract_test_samples('digits')
train_2, train_labels_2 = extract_training_samples('digits')

Test_images, Test_labels = np.concatenate((test_1, test_2), axis=0),np.concatenate((test_labels_1, test_labels_2), axis=0)
Train_images, Train_labels = np.concatenate((train_1, train_2), axis=0),np.concatenate((train_labels_1, train_labels_2), axis=0)

print(Test_images.shape)
print(Test_labels.shape)


print(Train_images.shape)
print(Train_labels.shape)

Train_images.shape

train_1.shape

train_2.shape

Train_images[55].shape

#Let's print first 10 letters from EMINST training samples
for i in range(7):
    first_ten_letters = Train_images[i].reshape((28, 28))
    plt.imshow(first_ten_letters, cmap = 'binary')
    #plt.axis("off")
    plt.show()
```

```
# Gonna Synthisize my out data set with boxes...

plt.imshow(Train_images[5])

print(Train_labels[:6])

print(len(Train_images[50]))

print(str(10))

import random
from PIL import Image
from xml.etree import ElementTree as ET

dataset_size = 80
for set_size in range(0,dataset_size) :

  #Blank Image
  temp_img = np.zeros([200,200],dtype=np.uint8)

  index = set_size
  annot = "<annotation>\n\
    <folder>VOC2007</folder>\n\
    <filename>" + str(index) + ".jpg</filename>\n\
    <source>\n\
      <database>EMNIST</database>\n\
      <annotation>EMNIST</annotation>\n\
      <image>EMNIST</image>\n\
      </source>\n\
        <owner>\n\
        <name>Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik</name>\n\
        </owner>\n\
          <size>\n\
            <width>200</width>\n\
            <height>200</height>\n\
            <depth>1</depth>\n\
            </size>\n\
            <segmented>0</segmented>"
  stack = random.randint(7, 15)
  for i in range(0 , stack) :
    j = random.randint(0 , len(Train_images) )
    anchor = ( random.randint(0,200 - len(Train_images[j]-1)), random.randint(0,200-len(Train_images[j])-1))
    temp_img[anchor[0]:anchor[0]+len(Train_images[j]),anchor[1]:anchor[1]+len(Train_images[j])] += Train_images[j]
    annot += "<object>\n\
              <name>" + str(Train_labels[j]) + "</name>\n\
              <pose>Left</pose>\n\
              <truncated>0</truncated>\n\
              <difficult>0</difficult>\n\
                <bndbox>\n\
                  <xmin>" + str(anchor[1]) + "</xmin>\n\
                  <ymin>" + str(anchor[0]) + "</ymin>\n\
                  <xmax>" + str(anchor[1] + len(Train_images[j])) + "</xmax>\n\
                  <ymax>" + str(anchor[0] + len(Train_images[j])) + "</ymax>\n\
                </bndbox>\n\
              </object>\n"
  annot += "</annotation>"

  tree = ET.XML(annot)
  with open("/content/darkflow-master/test/training/annotations/" + str(index) + '.xml', "wb") as f :
    f.write(ET.tostring(tree))

  im = Image.fromarray(temp_img)
  im.save("/content/darkflow-master/test/training/images/" + str(index) + ".jpg")
  #temp_img = temp_img.paste(img2, (0,0))

# Training

physical_devices = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

tf.test.gpu_device_name()

tf.config.list_physical_devices('GPU')

with tf.device('/device:GPU:0') :
  !python flow --model cfg/tiny-yolo-voc-3c.cfg --load bin/yolo.weights --train --annotation test/training/annotations --dataset test/training/images

!zip -r /content/darkflow-master/test/training.zip /content/darkflow-master/test/training

from google.colab import files
files.download('/content/darkflow-master/test/training.zip')



files.download("/content/darkflow-master/test/training")
```