In [1]:

```
# Text Detection
# Forrester Welch
# The goal of this project to recognize where an instance of text appears i

# Imports for convolutional neural networks, data management, and image prc
import tensorflow as tf
import pandas as pd
import json
import os
import csv
import numpy as np
from PIL import Image
from sklearn.model_selection import train_test_split
import tensorflow.keras.layers as layer
```

In [2]:

```
# Load the data from the cocotext json file
# Download the cocotext annotations json here: https://bgshih.github.io/coc
# Download cocotext.v2.zip [12 MB] and unzip for cocotext.v2.json, then ren
data = pd.read_json('cocotext.json')
```

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 236291 entries, 45346 to 390310
Data columns (total 5 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   cats       0 non-null       float64
 1   anns       201126 non-null  object
 2   imgs       53686 non-null   object
 3   imgToAnns  53686 non-null   object
 4   info       0 non-null       float64
dtypes: float64(2), object(3)
memory usage: 10.8+ MB
```

In [4]:

```
data.columns
```

```
Index(['cats', 'anns', 'imgs', 'imgToAnns', 'info'], dtype='object')
```

In [5]:
```
#example of element in data['anns']
data['anns'].iloc[1000]
```

```
{'area': 67.21,
 'bbox': [262.6, 218.4, 9.8, 8.1],
 'class': 'machine printed',
 'id': 102540,
 'image_id': 353906,
 'language': 'english',
 'legibility': 'illegible',
 'mask': [263.5, 219.3, 262.6, 225.9, 272.4, 226.5, 272.0, 218.4],
 'utf8_string': ''}
```

In [6]:
```
# cycle through annotations
# only add elements if machine printed, english, and legible
# create dataset of image ids (we will later convert id to image filename)
# create bbox dataset
annotations = data['anns']
image = []
bbox = []

# This step may take a couple minutes
for i in range(len(data['anns'])):
    current = annotations.iloc[i]
    if(pd.isna(current)):
        continue
    if(current['class'] == 'machine printed' and current['language'] == 'en
      and current['legibility'] == 'legible' and current['image_id'] not in
            image.append(annotations.iloc[i]['image_id'])
            bbox.append(annotations.iloc[i]['bbox'])
```

In [7]:
```
# example element of data['imgs']
data['imgs'].iloc[1000]['file_name']
```

```
'COCO_train2014_000000102540.jpg'
```

In [8]:

```
# To change the image_id value to the filename of the image, I need a hashm
# The dict object in python is supposed to operate like a hash map, but I c
# to make it work for our purposes. I found this implementation of a hash t
# below. This HashTable implementation made it simple and easy to convert i
# https://www.geeksforgeeks.org/hash-map-in-python/

class HashTable:

    # Create empty bucket list of given size
    def __init__(self, size):
        self.size = size
        self.hash_table = self.create_buckets()

    def create_buckets(self):
        return [[] for _ in range(self.size)]

    # Insert values into hash map
    def set_val(self, key, val):

        # Get the index from the key
        # using hash function
        hashed_key = hash(key) % self.size

        # Get the bucket corresponding to index
        bucket = self.hash_table[hashed_key]

        found_key = False
        for index, record in enumerate(bucket):
            record_key, record_val = record

            # check if the bucket has same key as
            # the key to be inserted
            if record_key == key:
                found_key = True
                break

        # If the bucket has same key as the key to be inserted,
        # Update the key value
        # Otherwise append the new key-value pair to the bucket
        if found_key:
```

```
                bucket[index] = (key, val)
            else:
                bucket.append((key, val))

    # Return searched value with specific key
    def get_val(self, key):

        # Get the index from the key using
        # hash function
        hashed_key = hash(key) % self.size

        # Get the bucket corresponding to index
        bucket = self.hash_table[hashed_key]

        found_key = False
        for index, record in enumerate(bucket):
            record_key, record_val = record

            # check if the bucket has same key as
            # the key being searched
            if record_key == key:
                found_key = True
                break

        # If the bucket has same key as the key being searched,
        # Return the value found
        # Otherwise indicate there was no record found
        if found_key:
            return record_val
        else:
            return "No record found"

    # Remove a value with specific key
    def delete_val(self, key):

        # Get the index from the key using
        # hash function
        hashed_key = hash(key) % self.size

        # Get the bucket corresponding to index
```

```
        bucket = self.hash_table[hashed_key]

        found_key = False
        for index, record in enumerate(bucket):
            record_key, record_val = record

            # check if the bucket has same key as
            # the key to be deleted
            if record_key == key:
                found_key = True
                break
        if found_key:
            bucket.pop(index)
        return

    # To print the items of hash map
    def __str__(self):
        return "".join(str(item) for item in self.hash_table)


hash_table = HashTable(50)

# insert some values
hash_table.set_val('gfg@example.com', 'some value')
print(hash_table)
print()

hash_table.set_val('portal@example.com', 'some other value')
print(hash_table)
print()

# search/access a record with key
print(hash_table.get_val('portal@example.com'))
print()

# delete or remove a value
hash_table.delete_val('portal@example.com')
print(hash_table)
```

```
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][
```

```
lue')]

[('portal@example.com', 'some other value')][][][][][][][][][][][][][][][][][][][][][][][][][][
[][][][][][][][][('gfg@example.com', 'some value')]

some other value

[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][][
lue')]
```

In [9]:

```python
# Intialize HashTable with key-value pair of image_id <-> file_name

# Our hashtable that will store key-value pairs of id-filename
image_tree = HashTable(10000)

# Cycle through imgs to gather data
for i in range(len(data['imgs'])):
    if(pd.isna(data['imgs'].iloc[i])):
        continue
    current = data['imgs'].iloc[i]
    image_tree.set_val(current['id'], current['file_name'])
```

In [10]:

```python
# Convert the image vector to contain file names instead of id numbers
for i in range(len(image)):
    image[i] = image_tree.get_val(image[i])
```

In [11]:

```
# This block of code can be skipped in the future. It is now commented out
# The purpose of this block is to put the image file names into a text file
# The reason for this has to do with how the images for this project were c
# The coco-text.json annotations were released as an addendum to the origin
# Every instance of text in that datset was recorded in coco-text.json. How
# has an instance of text. It is not posssible to download just the text im
# 2014 COCO image dataset can be downloaded. To save storage space, I moved
# of the folder so I could delete the unnecessary images all at once. For r
# command to move a list of files is as follows:
# for i in $(cat all_text_image_names.txt); do mv "$i" /temp_dest/; done
# The set of necessary images can be found in my github at:


#unique_filenames = list(set(image))
#import numpy as np
#name_file = open("all_text_image_names.txt", "w")
#np.savetxt(name_file, unique_filenames, fmt="%s")

#name_file.close()
```

In [12]:

```
# Convert the list of image file_names to a list of 2d-arrays of pixels
# Converts the bbox into a scale of [0,1]
# bbox is originally annotated: x,y,width,height
    # We convert to xmin, ymin, xmax, ymax on scale of 0-1
# This step may take five minutes
for i in range(len(image)):
    image_name = "train2014/" + image[i]
    width, height = Image.open(image_name).size
    xmax = (bbox[i][0] + bbox[i][2]) / width
    ymax = (bbox[i][1] + bbox[i][3]) / height
    xmin = bbox[i][0] / width
    ymin = bbox[i][1] / height
    bbox[i] = [xmin, ymin, xmax, ymax]
    # The images are resized to (100,100) because the kernel could not hand
    # With limitless computational resources, a full size 600x600 image may
    # Interestingly, when images were resized to 128x128 or 164x164, they h
    # than the 100x100 option.
    file = tf.keras.preprocessing.image.load_img(image_name, target_size=(1
    image[i] = tf.keras.preprocessing.image.img_to_array(file)
```

In [ ]:

In [13]:

```
# The image pixel values are rescaled from [0-1]
image = np.array(image, dtype="float32") / 255
bbox = np.array(bbox, dtype="float32")
```

In [14]:

```
# Split the train and test data with split size .2 and random seed = 400
image_train, image_test, bbox_train, bbox_test = train_test_split(image, bb
```

In [15]:

```
# use imgs to create key value map of id to image name
# get list of all image names from imgs
# cycle through anns and keep every image that contains an annotation of te
```

In [16]:

```
# Create a keras Model
# Our model has 3 convolutional layers with 32,32, and 64 filters
# The final layer of our model returns four neurons, each representing a cc
# This architecture differs from traditional neural networks doing classifi
# Instead of recognizing what an object is, we aim to find where an object
# This is done using regression to calculate the best fitting bounding box.
# The outline for how to make a regression layer the final layer of the mod
# following link: https://medium.com/analytics-vidhya/object-localization-w
# We trained the model using a different number of filters on each layer, a
# of layers. The results of these experiments are noted in the final writeu
# more filters lead to overfitting which lowered accuracy on the validation
def get_model():
    inputs = tf.keras.Input(shape=(100,100,1))
    x = layer.Conv2D(32, (3,3), activation='relu')(inputs)
    x = layer.MaxPooling2D((3,3))(x)
    x = layer.Conv2D(32, (3,3), activation='relu')(x)
    x = layer.MaxPooling2D((3,3))(x)
    x = layer.Conv2D(64, (3,3), activation='relu')(x)
    x = layer.GlobalAveragePooling2D()(x)


    reg_head = layer.Dense(128, activation='relu')(x)
    reg_head = layer.Dense(64, activation='relu')(x)
    reg_head = layer.Dense(32, activation='relu')(reg_head)
    # Notice the name of the layer.
    reg_head = layer.Dense(4, activation='sigmoid', name='bbox')(reg_head)
    return tf.keras.Model(inputs=[inputs], outputs=[reg_head])
```

In [ ]:

In [17]:

```
# Initalize the model
model = get_model()
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100, 100, 1)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 98, 98, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          18496
_________________________________________________________________
global_average_pooling2d (Gl (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
bbox (Dense)                 (None, 4)                 132
=================================================================
Total params: 34,436
Trainable params: 34,436
Non-trainable params: 0
_________________________________________________________________
```

In [18]:

```
batch_size = 128
# We experimented with more epochs, but this led to overfitting the data an
# on the validation set. Around 30 epochs, the loss function comes close tc
epochs = 30

losses = "mean_squared_error"

model.compile(loss=losses, optimizer="adam", metrics=["accuracy"])
```

In [19]:

```
# Train the model.
# This step may take 20-30 minutes. It may be easier to change epochs to 10
model.fit(image_train, bbox_train, batch_size=batch_size, epochs=epochs, va
```

```
Epoch 1/30
83/83 [==============================] - 60s 704ms/step - loss: 0.0702 - accuracy: 0.5224 - val_
5308
Epoch 2/30
83/83 [==============================] - 53s 637ms/step - loss: 0.0698 - accuracy: 0.5219 - val_
5308
Epoch 3/30
83/83 [==============================] - 54s 645ms/step - loss: 0.0689 - accuracy: 0.5336 - val_
5308
Epoch 4/30
83/83 [==============================] - 51s 619ms/step - loss: 0.0684 - accuracy: 0.5411 - val_
5282
Epoch 5/30
83/83 [==============================] - 50s 598ms/step - loss: 0.0686 - accuracy: 0.5541 - val_
5529
Epoch 6/30
83/83 [==============================] - 47s 569ms/step - loss: 0.0672 - accuracy: 0.5580 - val_
5586
Epoch 7/30
83/83 [==============================] - 47s 563ms/step - loss: 0.0672 - accuracy: 0.5653 - val_
5533
Epoch 8/30
83/83 [==============================] - 46s 560ms/step - loss: 0.0667 - accuracy: 0.5757 - val_
5621
Epoch 9/30
83/83 [==============================] - 47s 570ms/step - loss: 0.0670 - accuracy: 0.5787 - val_
5617
Epoch 10/30
83/83 [==============================] - 46s 557ms/step - loss: 0.0680 - accuracy: 0.5683 - val_
5510
Epoch 11/30
83/83 [==============================] - 53s 636ms/step - loss: 0.0657 - accuracy: 0.5688 - val_
5724
Epoch 12/30
83/83 [==============================] - 53s 635ms/step - loss: 0.0658 - accuracy: 0.5764 - val_
5434
Epoch 13/30
83/83 [==============================] - 48s 574ms/step - loss: 0.0670 - accuracy: 0.5822 - val_
5583
Epoch 14/30
83/83 [==============================] - 48s 578ms/step - loss: 0.0657 - accuracy: 0.5879 - val_
5678
Epoch 15/30
83/83 [==============================] - 47s 568ms/step - loss: 0.0657 - accuracy: 0.5810 - val_
5838
Epoch 16/30
83/83 [==============================] - 47s 570ms/step - loss: 0.0669 - accuracy: 0.5814 - val_
5655
Epoch 17/30
83/83 [==============================] - 47s 568ms/step - loss: 0.0652 - accuracy: 0.5807 - val_
5701
```

```
Epoch 18/30
83/83 [==============================] - 47s 566ms/step - loss: 0.0658 - accuracy: 0.5811 - val_
5625
Epoch 19/30
83/83 [==============================] - 48s 574ms/step - loss: 0.0657 - accuracy: 0.5806 - val_
5613
Epoch 20/30
83/83 [==============================] - 48s 573ms/step - loss: 0.0651 - accuracy: 0.5950 - val_
5621
Epoch 21/30
83/83 [==============================] - 47s 566ms/step - loss: 0.0654 - accuracy: 0.5759 - val_
5312
Epoch 22/30
83/83 [==============================] - 50s 600ms/step - loss: 0.0646 - accuracy: 0.5895 - val_
5472
Epoch 23/30
83/83 [==============================] - 51s 621ms/step - loss: 0.0652 - accuracy: 0.5891 - val_
5560
Epoch 24/30
83/83 [==============================] - 51s 612ms/step - loss: 0.0647 - accuracy: 0.5844 - val_
5640
Epoch 25/30
83/83 [==============================] - 52s 631ms/step - loss: 0.0638 - accuracy: 0.5923 - val_
5567
Epoch 26/30
83/83 [==============================] - 52s 624ms/step - loss: 0.0642 - accuracy: 0.5916 - val_
5758
Epoch 27/30
83/83 [==============================] - 54s 653ms/step - loss: 0.0632 - accuracy: 0.6055 - val_
5541
Epoch 28/30
83/83 [==============================] - 52s 632ms/step - loss: 0.0645 - accuracy: 0.5841 - val_
5407
Epoch 29/30
83/83 [==============================] - 51s 608ms/step - loss: 0.0628 - accuracy: 0.6033 - val_
5762
Epoch 30/30
83/83 [==============================] - 64s 775ms/step - loss: 0.0623 - accuracy: 0.6064 - val_
5605
```

```
<tensorflow.python.keras.callbacks.History at 0x7fb29ef1a7b8>
```

In [20]:

```python
# Measure accuracy against validation set.
model.evaluate(image_test, bbox_test)
```

```
103/103 [==============================] - 6s 60ms/step - loss: 0.0653 - accuracy: 0.5689
```

```
[0.06526488810777664, 0.5688604712486267]
```