Project Timeline

Start of Project

- Project referenced from Nvidia code
    - CNN model copied exactly from code
- Initial image size = 256
- Started as a base start

Issues & Solutions # 1

1. No idea how to properly format data
    - Unsure what the size should be, or what should be in our data.
    - **As a result, our data had to be formatted 70% Training, 20% Validation, and 10% Testing**
    - **Dataset should be a mix of images (Ex: Waldo/no Waldo)**
2. Running the CNN model would end up using all the RAM & CPU
    - Most likely had to do with incorrect data sizes.

Middle of Project (Part 1)

- Updated Dataset from issues from initial project
    - Correctly classified
        - xTrain, xValid $\rightarrow$ images
        - yTrain, yValid $\rightarrow$ folder name
    - Used binary as the classifier.
        - Sigmoid activation for model remained the same.
- Result interpretation
    - Decided that if result is greater than 0.5, that means Waldo was in the image.
    - Otherwise, he was not in the image

```
result = make_predictions('Hey-Waldo-master/pepe.jpeg', width, height)
if result > 0.5:
    print("Waldo is present in the image!")
else:
    print("Waldo is not present in the image.")


result = make_predictions('Hey-Waldo-master/256/waldo/2_0_1.jpg',
width, height)
if result > 0.5:
    print("Waldo is present in the image!")
else:
    print("Waldo is not present in the image.")
```

Issues and Solutions #2

1. Incorrect classification
    - It would not be accurate at all. Waldo was not being classified. The CNN model that we planned on using was a classification model.
2. Incorrect Results
    - **SOLUTION**: There were too many Not-Waldo images in the training set. In order to fix this, we have to *OVERSAMPLE* the Waldo images.

Middle of Project (Part 2)

- Improvements were made on the CNN Model

    - Replaced with a Regression-based model

```
# Old CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(width,
height, 3)))
model.add(MaxPool2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPool2D(2, 2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPool2D(2, 2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
model.fit(xTrain, yTrain , validation_data=(xValid, yValid),
        epochs=10, batch_size=4)

# New CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(width,
height, 3)))
model.add(MaxPool2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPool2D(2, 2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPool2D(2, 2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(2, activation='linear'))  # Two output neurons for the
x and y coordinates

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(xTrain, yTrain , validation_data=(xValid, yValid),
        epochs=10, batch_size=4)
```

- Since CNN was changed, so was the Testing

- Instead of determining if waldo is in an image by the output value, the new output will be the coordinate where it predicts where waldo is located.

```
# Old Testing
result = make_predictions('Hey-Waldo-master/pepe.jpeg', width, height)
if result > 0.5:
    print("Waldo is present in the image!")
else:
    print("Waldo is not present in the image.")


result = make_predictions('Hey-Waldo-master/256/waldo/2_0_1.jpg',
width, height)
if result > 0.5:
    print("Waldo is present in the image!")
else:
    print("Waldo is not present in the image.")

# New updated Testing
for items in range(len(testingSet)):
getImg = testingSet[items][0]
print(f"===\n i = {items} \n===")
print(f"Image: {getImg} and Number: {testingSet[items][1]}")
imgResults = predictions(getImg, width, height, model)
print(f"Results: {imgResults}")

print("\n\n=== === ===\nNOW TESTING WITH WALDO")
imgWPATH = os.path.join('Hey-Waldo-master', '64', 'waldo',
'19_0_7.jpg')
imgWALDORes = predictions(imgWPATH, width, height, model)
print(f"Waldo Results: {imgWALDORes}")
```

- Since we are now using the coordinates provided by the dataset file, we must appoint the correct coordinates to the correct image as the "labels" (yTrain). This will help the CNN model learn which coordinates and shapes identify waldo within an image.

```
def coord_Data_Appointing(dataset, coord_file, w, binary, hasFile):
    # Opens the coordinates file
    if(hasFile == True):
        with open(coord_file, 'r') as f:
            coords = json.load(f)
    imgData_pairs = []
    imgDirect = os.path.dirname(dataset[0][0])
    # Retrieves the image name and the coordinates
    # Appends them together and preps them for coord appointing
    for img in dataset:
        imgPath = img[0]
        imgName = os.path.basename(imgPath)
        if(imgName != '.DS_Store'):
            pattern = r'\d+'
            digits = re.findall(pattern, imgName)
            if len(digits[0]) == 1:
                digits[0] = '0' + digits[0]
            imgData_pairs.append((imgName, digits))

    imgData_pairs = sorted(imgData_pairs, key=lambda x: x[1])

    # Coordinates are assigned to the image
    for img in imgData_pairs:
        imgName = img[0]
        # Get coords
        xPos = -1
        yPos = -1
        if hasFile == True:
            # issue?
            for item in coords[str(int(w))][str(int(img[1][0]))]:
                if(item["x"] == (img[1][1]) and item["y"] == (img[1]
[2])):
                    xPos = item["x_px"] / 64
                    yPos = item["y_px"] / 64
                    break

        xyCoords = [xPos, yPos]
        dataIndx = dataset.index((imgDirect + '/' + imgName, binary))

        temp = list(dataset[dataIndx])
        temp.append(xyCoords)
        dataset[dataIndx] = tuple(temp)
```

- Our data was not being normalized beforehand.

```
    def dataAdjusting(imgDataset, width, height):
        imgs = []
        coords = []
        for imgPath in imgDataset:
            if(os.path.basename(imgPath[0]) != '.DS_Store'):
                img = cv2.imread(imgPath[0])
                img = cv2.resize(img, (width, height))
                img = img / 255.0
                imgs.append(img)

                normCoords = [(imgPath[2][0])/width, (imgPath[2]
[1])/height]
                coords.append(normCoords)

        imgs = np.array(imgs)
        #coords = (np.array(coords))/(width * height)
        coords = (np.array(coords))

        return imgs, coords
```