

SD2 - Yunhua Zhao

American Dream

...

Martin Lahoumh
Brandon Tjandra
Miguel Luna
Jiazhou Zhang

Project Recap

The goal of this project is to create a web application that:

1. Takes an image (supplied by the user) that contains text in some language as input
2. Extracts the text
3. Translates it to a user's preferred language
4. Places the translated text on the appropriate parts of the image

Research Overview

For our project, we mainly referenced the two following articles:

- Article 1: Character Region Awareness for Text Detection (2019, Baek et al)

source: <https://arxiv.org/pdf/1904.01941>

- Article 2: What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis (2019, Baek et al)

source: <https://arxiv.org/pdf/1904.01906>

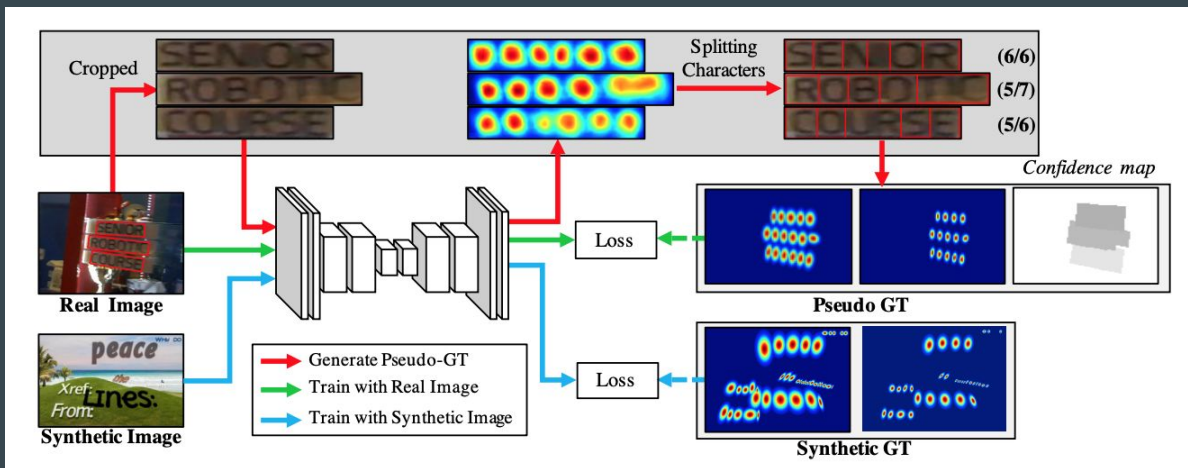
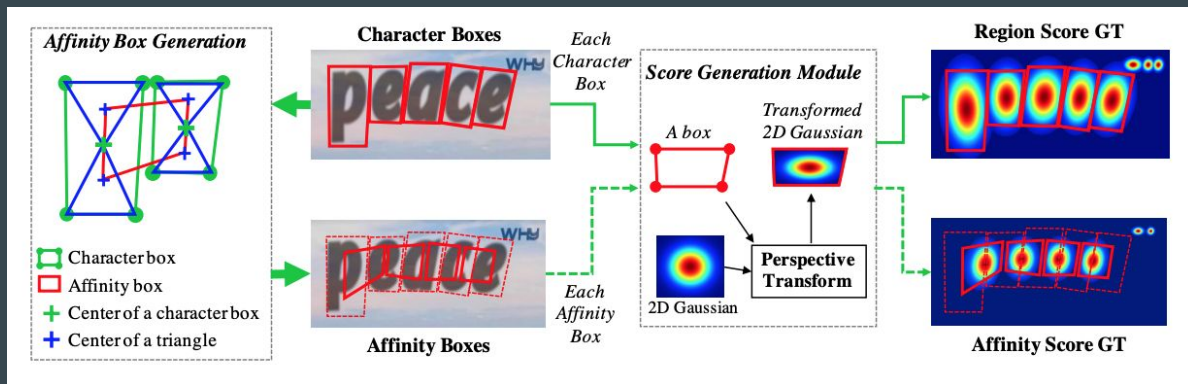
Summary: Character Region Awareness for Text Detection

- Proposal of CRAFT text detection model
- CNN that uses character-level text detection and character affinity to detect location of words in images.
- Trained with ICDARs and MSRATD500.

Takeaways :

- Text detection models for scene text struggle with accuracy due to word contexts and meanings.
- It is used in the OCR pipeline of EasyOCR, so it is the text detection model for this project.

CRAFT Score Generation Example



Summary: What Is Wrong With Scene Text Recognition Model Comparisons?

Training scene text recognition (STR) models are costly, making it difficult to obtain labeled data. The authors resort to using synthetic text datasets like MJSynth and SynthText.

This brings up the question of whether performance improvements are due to the model itself or a better set of training data.

Summary: What Is Wrong With Scene Text Recognition Model Comparisons?

Takeaways :

Creating and training our own model from scratch would be infeasible due to a lack of datasets suited for scene text detection.

This is why we believe that using a pre-trained OCR such as EasyOCR is necessary for the completion of this project given our time and knowledge limitations.

Previous Work

Over the course of last semester and this semester, we have researched ways to implement an OCR (optical character recognition) for this project.

We have also attempted **twice** to develop our own OCR model, with unsatisfactory results. We realized that doing this would be infeasible given our time, knowledge, and resources.

This section will briefly cover the work done on those models and our reason for dropping them.

1. TensorFlow Model

Three convolutional layers:

- 1) 32 filters, size: 3x3, parameter: 320
 - a) Spatial dimension reduced to (16, 64, 32)
- 2) 64 filters, size 3x3, parameters: 18,496
 - a) (8, 32, 64)
- 3) 128 filters, size 3x3, parameters: 73,856
 - a) (4, 16, 128)

Layer is reshaped into (4,2048) for sequence processing

Passed to two bidirectional layers:

- 1) 256 units, parameters: 2,229,248
- 2) 256 units, parameters: 394,240

Output layer has 80 possible output classes.
Softmax for classification.

Preprocessing and Training

We tried using the following datasets:

- **ICDAR-2015**

- Training + validation Sets
- 500 images + text annotations stored as .txt file

- **EMNIST**

- Training + validation Sets
- 800,000 character instances
- Handwritten digit dataset

- 1) Load images and their corresponding labels
- 2) Normalize the datasets by converting them into NumPy arrays and dividing by 255 to scale pixel values in a range of [0,1]
- 3) Reshape images to (-1, 32, 128, 1)
- 4) Collect unique characters to create consistent mapping, each char is mapped onto an integer.

Images are input of the CNN. Encoded label sequences are the output of the RNN.

Model Training & Evaluation on ICDAR

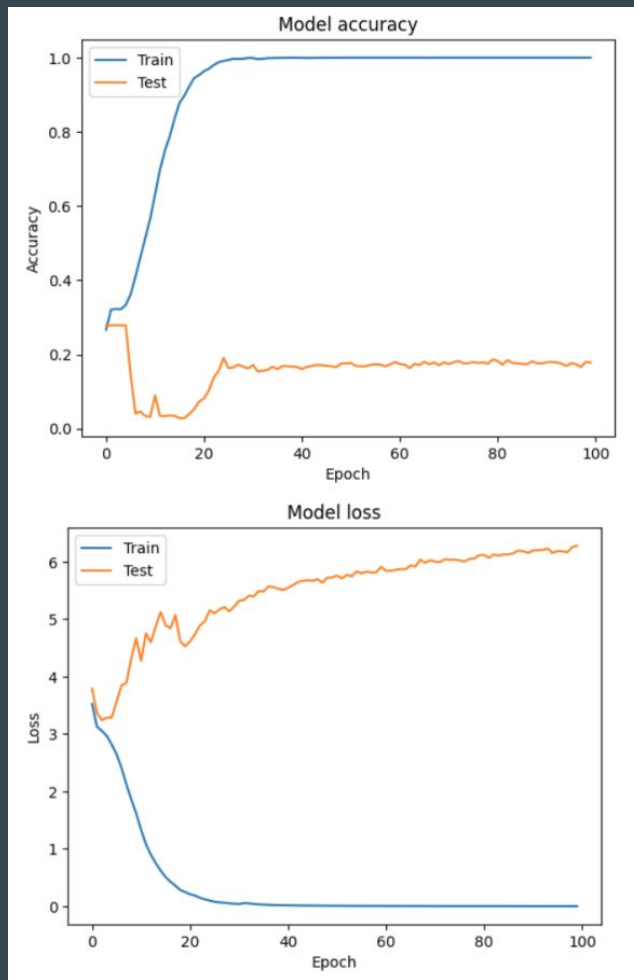
Optimizer: Adam optimizer with learning rate of 0.001

Loss Function: `sparse_categorical_crossentropy` due to multi-class classification

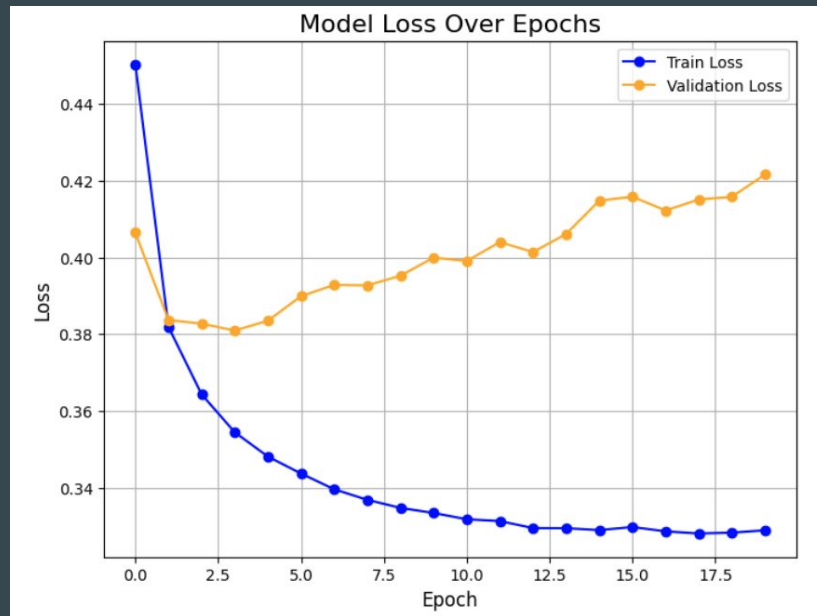
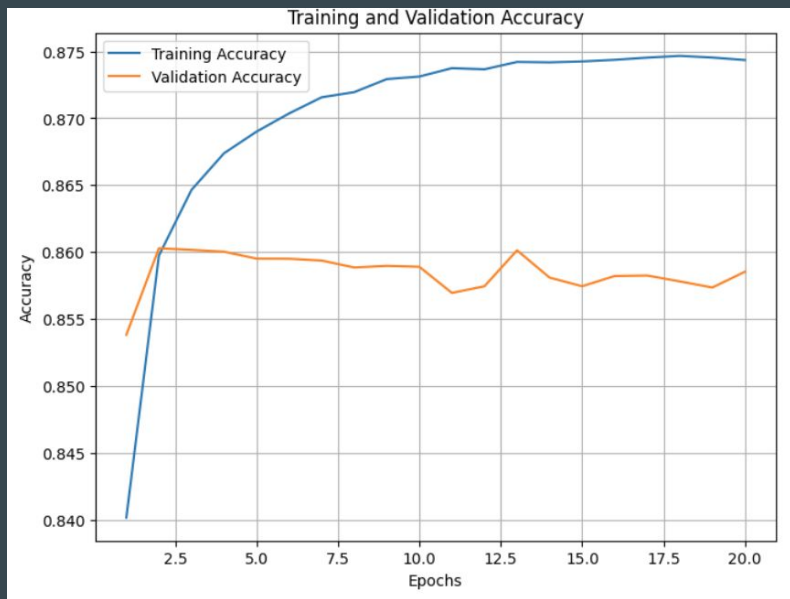
Metrics: Accuracy

Batch Size: 32

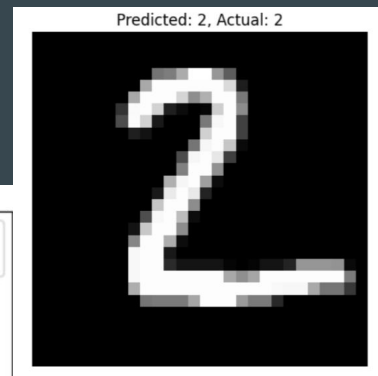
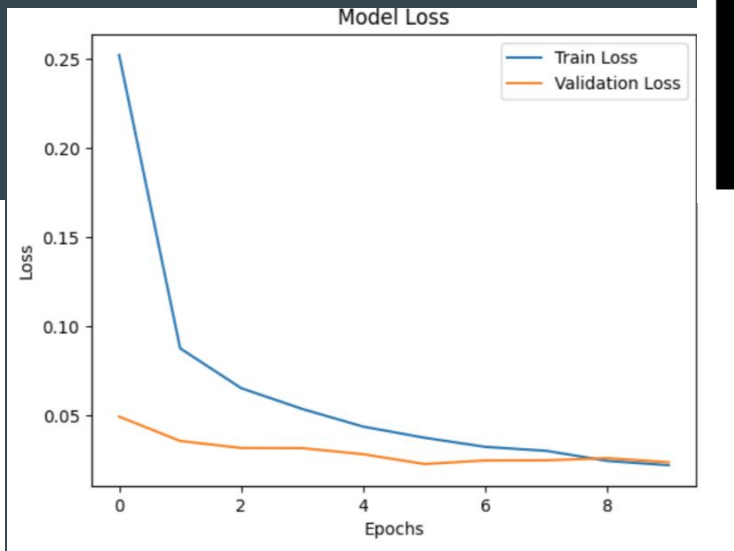
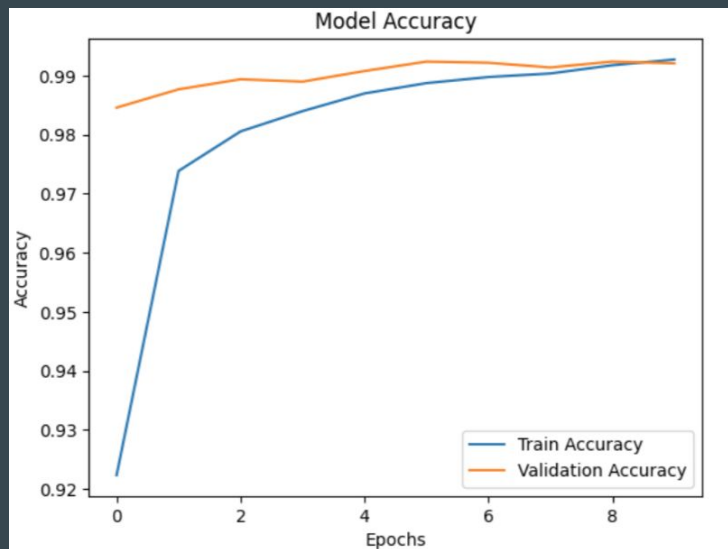
Epochs: 100



Model Training & Evaluation On EMNIST



Model Training & Evaluation On MNIST



Model Training & Evaluation Results

	precision	recall	f1-score	support
0	0.681956	0.815680	0.742848	5778.000000
1	0.690640	0.885940	0.776194	6330.000000
2	0.951537	0.970182	0.960769	5869.000000
accuracy	0.857062	0.857062	0.857062	0.857062
macro avg	0.758172	0.721127	0.720694	116323.000000
weighted avg	0.844566	0.857062	0.843660	116323.000000

Model Analysis

The other major problem with this implementation is the **lack of a good dataset** to train our recognition model.

While many of the OCR models we found in our research use the ICDAR datasets, they are not suitable for the recognition models alone. If we were to use ICDAR, we would need to isolate the images containing only text.

While (E)MNIST is more suitable, it only contains numbers and not characters, and it is handwritten text, not printed text.

Model Analysis

One major problem with this implementation is the lack of a detection model to go with it.

The OCR pipeline requires a text detection model (for detecting text location in the image) and a text recognition model (for transcribing image text).

We could not find a way to implement our own text detection model from scratch, so we focused on the recognition model.

As such, when attempting to find text from an image in the dataset, it searches the **whole image** instead of a centralized location, so it won't work most of the time.

2. TensorFlow + CTC

(source: [Text Recognition With TensorFlow and CTC network](#))

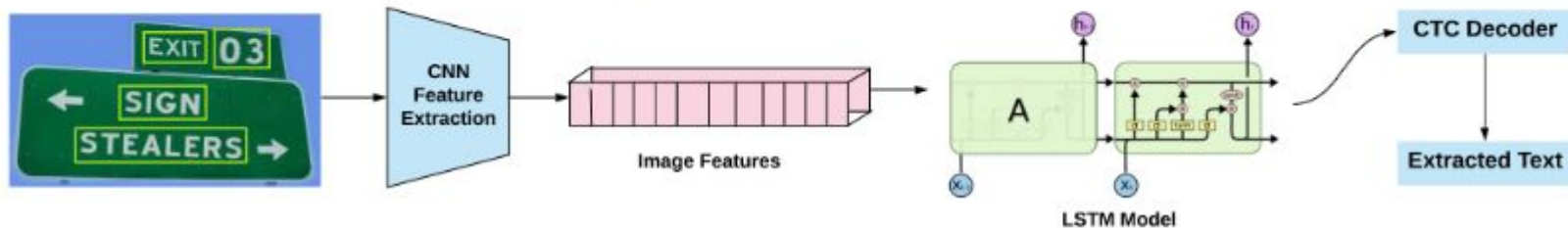
In this attempt, we referenced a TensorFlow model with a CTC loss function.

This model has two advantages compared to the last:

1. CTC networks allow for word extraction without requiring **individual character segmentation**
2. This model makes use of a synthetic dataset called **MJSynth**, which is more suited for text recognition models
 - 9 million images covering 90k English words

Text Recognition Pipeline:

Text Recognition usign CNN+LSTM network with CTC loss



Input image
(None, 32, 128, 3)

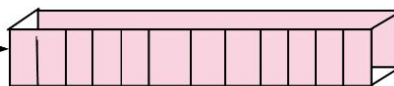
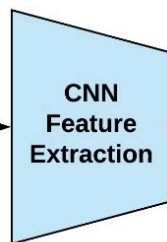
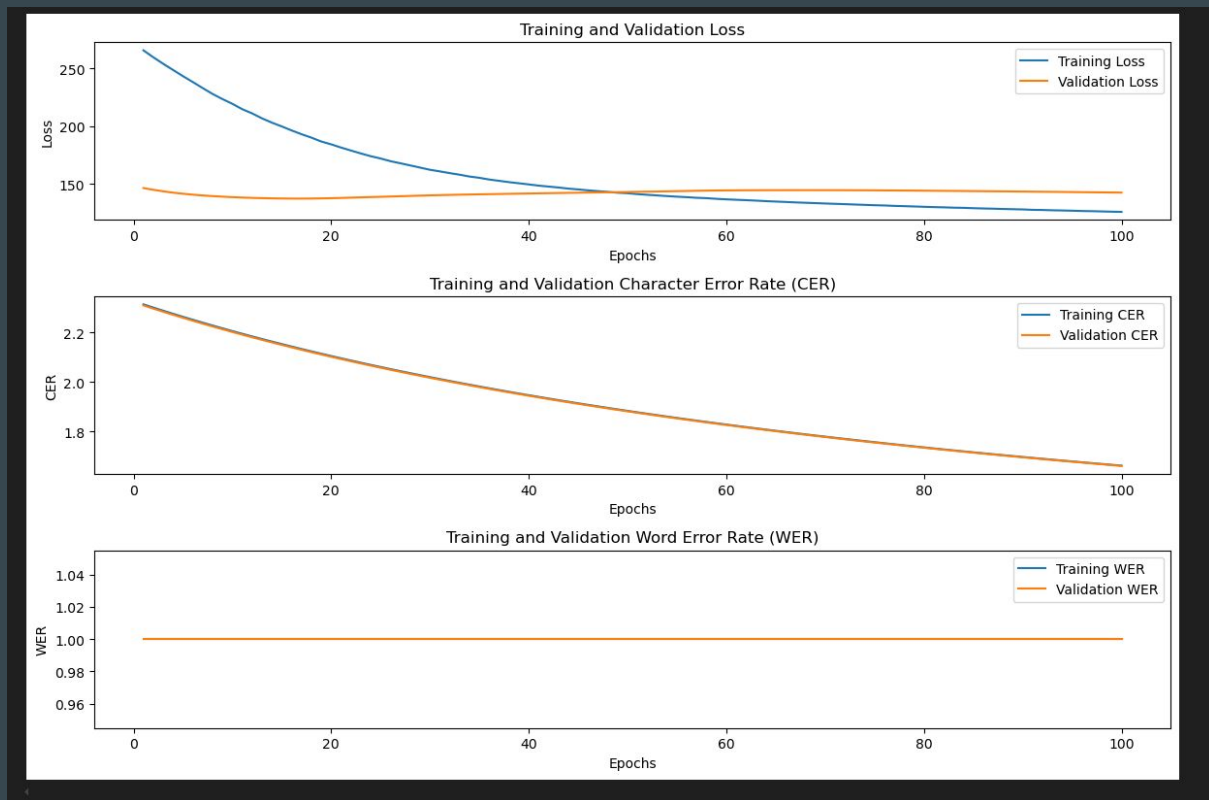


Image Features

Feature map
(None, 8, 32, 64)

Model Training



Model Analysis (cont.)

In addition to the issues faced in the previous model, the main challenge with using this model was how large the dataset was (**10 GB data**), which was both time and resource intensive even after splitting it up.

Without a text detection model, we are also still not able to perform adequate OCR inferences on images with scene text.

Previous Work Conclusion

If we are to continue making an OCR, we would need:

- Our own detection model
- Training datasets better suited for both models
- Fine-tuning either model further to improve accuracy

Since we could not find resources regarding the individual components of an OCR pipeline, this would require time for us to develop ourselves. Improving metrics would also require more time and resources to look into, making this overall unfeasible to continue.

As such, this is why we believe the only way to proceed is by using a pre-trained OCR model.

Model Overview: EasyOCR

(source: <https://github.com/JaidedAI/EasyOCR>)

- We've established that is a pretrained model, which solves our problems
- As the name suggests, it is easy to use. Getting bounding boxes and image text only takes a few lines
- EasyOCR supports 80 input languages. This allows us to use different input languages in our translation application without needing several datasets for those languages in training

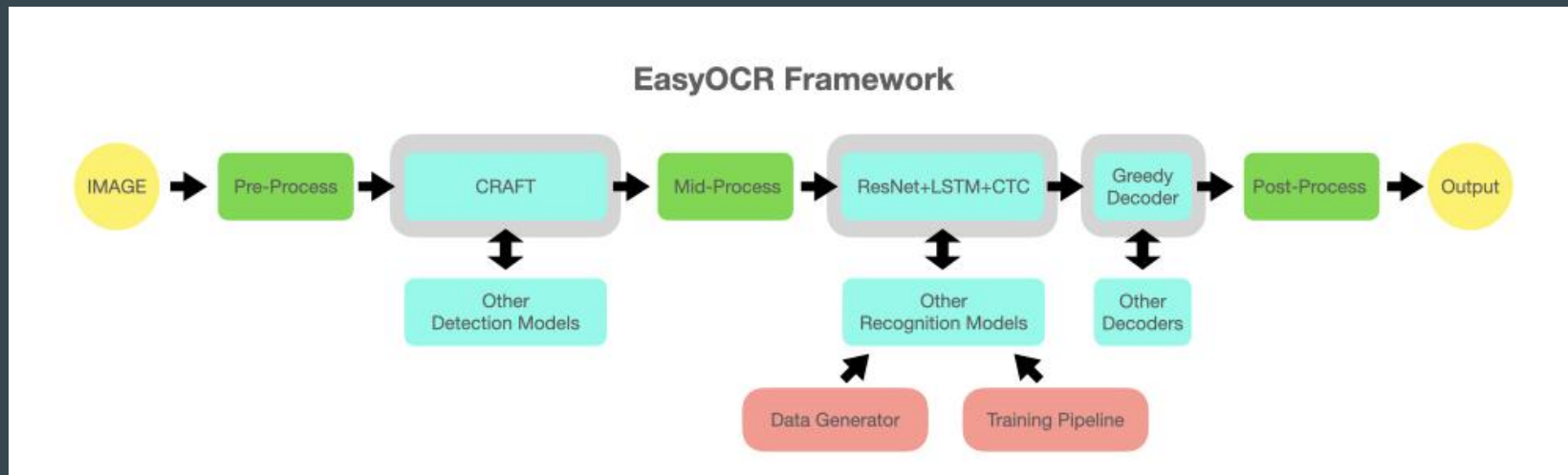
```
import easyocr
reader = easyocr.Reader(['ch_sim', 'en'])
reader.readtext('chinese.jpg')
```

Output will be in list format, each item represents bounding box, text and confident level, respectively.

```
[[[189, 75], [469, 75], [469, 165], [189, 165]], '愚园路', 0.3754989504814148],
 [[86, 80], [134, 80], [134, 128], [86, 128]], '西', 0.40452659130096436],
 [[517, 81], [565, 81], [565, 123], [517, 123]], '东', 0.9989598989486694],
 [[78, 126], [136, 126], [136, 156], [78, 156]], '315', 0.8125889301300049],
 [[514, 126], [574, 126], [574, 156], [514, 156]], '309', 0.4971577227115631],
 [[226, 170], [414, 170], [414, 220], [226, 220]], 'Yuyuan Rd.', 0.8261902332305908],
 [[79, 173], [125, 173], [125, 213], [79, 213]], 'W', 0.9848111271858215],
 [[529, 173], [569, 173], [569, 213], [529, 213]], 'E', 0.8405593633651733]]
```

EasyOCR Framework

(source: <https://github.com/JaidedAI/EasyOCR>)



Model Evaluation Overview

We downloaded the images + text annotations of the following datasets to evaluate EasyOCR ourselves:

- **TextOCR**

- Training + validation sets
- Over 800,000 text annotations on around 25,000 images
- Images from OpenImages training set (provided by website)
- Annotations stored on JSON

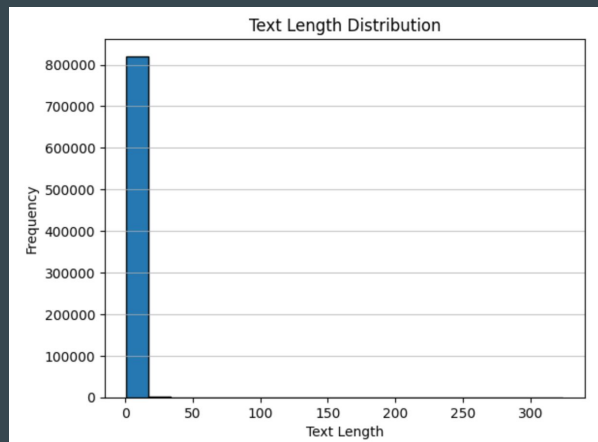
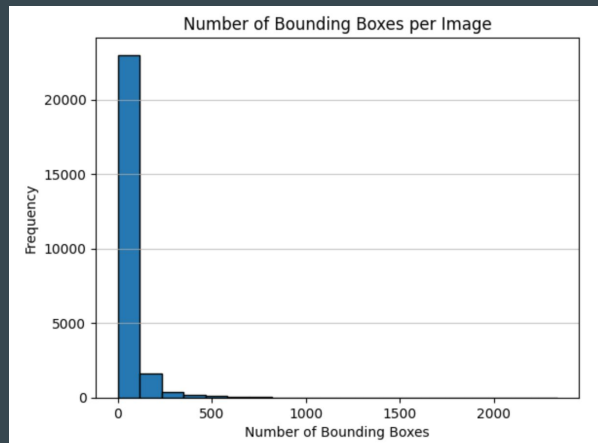
- **ICDAR-2015**

- Training + validation Sets
- 500 images + text annotations stored as .txt file
- Images taken by people from different places in the world

Data Exploration: TextOCR (legible text)

Total images: 24902
Total bounding boxes: 1228142
Average number of bounding boxes per image: 48.29
Median number of bounding boxes per image: 21.00
Standard deviation of nbounding boxes per image: 90.93
Minimum bounding boxes in an image: 1
Maximum bounding boxes in an image: 2337

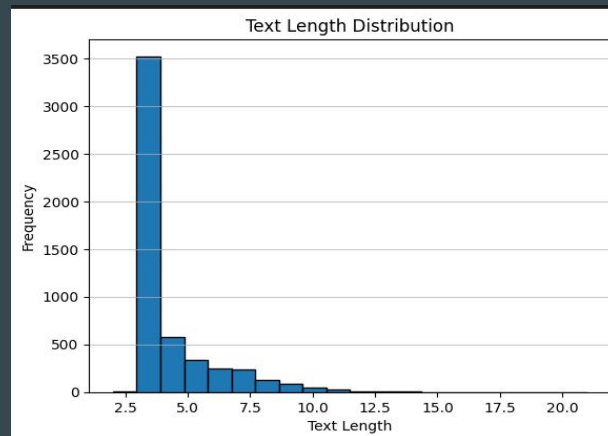
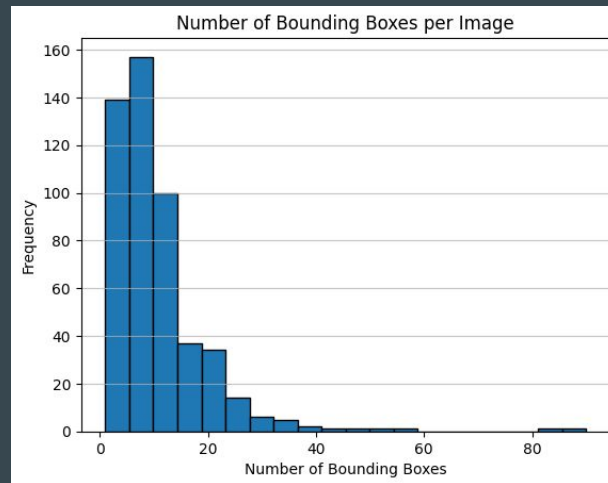
Median text length: 4.00
Standard deviation of text length: 2.97
Minimum text length: 1
Maximum text length: 324
Average text length: 4.54



Data Exploration: ICDAR-2015

Total number of images: 500
Total bounding boxes: 5230
Average number of bounding boxes per image: 10.46
Average text length: 3.93
Median number of bounding boxes per image: 8.00
Standard deviation of nbounding boxes per image: 9.14
Minimum bounding boxes in an image: 1
Maximum bounding boxes in an image: 90

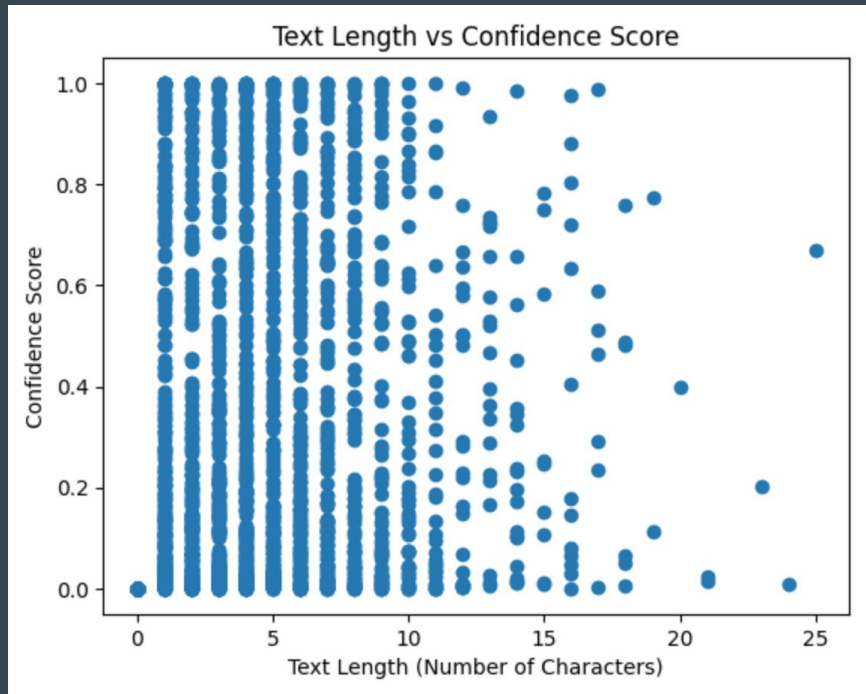
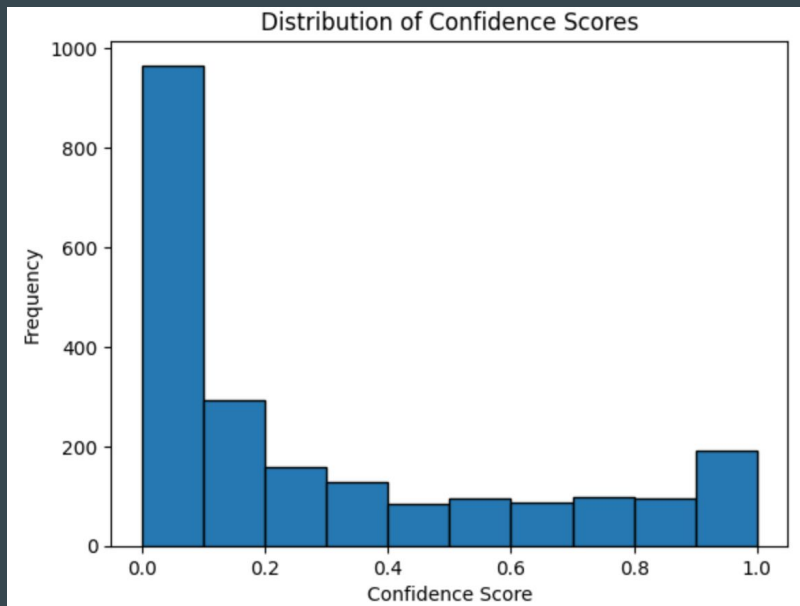
Median text length: 3.00
Standard deviation of text length: 1.77
Minimum text length: 2
Maximum text length: 21



Model Inference Overview

The model metrics shown in this slide were obtained by running the ICDAR-2015 testing dataset (500 images) through the model.

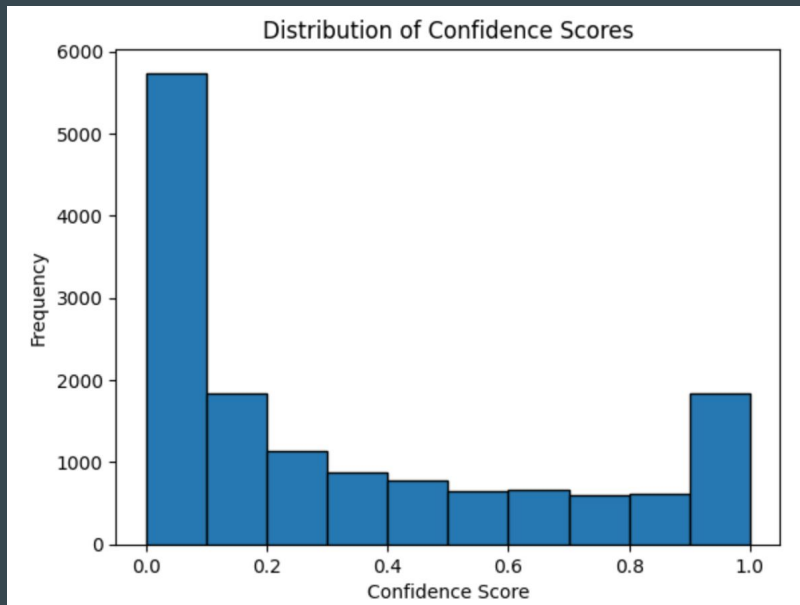
- Number of bounding boxes created: **2,196**
- Average confidence score: **0.29**



Model Inference Overview

The model metrics shown in this slide were obtained by running part of the train + validation segment of the TextOCR dataset (1,000 testing images) through the model.

- Number of bounding boxes created: **14,730**
- Average confidence score: **0.26**



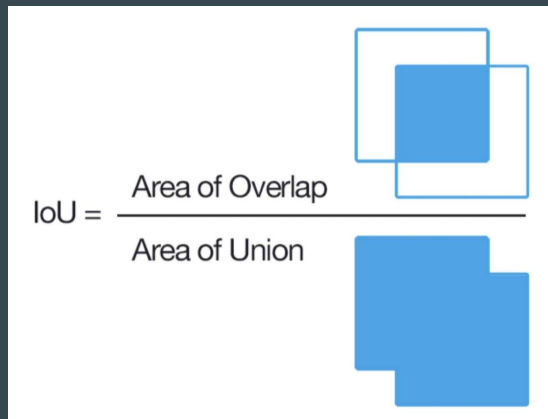
Model Evaluation

(source: [Measuring Labelling Quality with IOU and F1 Score](#), Isaac Tan)

To calculate precision, recall, and F1, we used Intersection over Union (IoU) scores as our loss function, comparing the predicted bounding boxes to the closest ground truth boxes (provided in the dataset). When the boxes overlap, they produce a score between 0 and 1: the closer to 1, the better the overlap and accuracy.

We used the following definitions:

- TP: $\text{IoU} > 0.5$
- TN: neither predicted box nor truth box exist (not needed or calculated)
- FP: $\text{IoU} < 0.5$
- FN: predicted box and closest truth box do not intersect ($\text{IoU} = 0$)



Model Evaluation (ICDAR-2015)

Precision is the ratio of correctly drawn annotations to the total number of drawn annotations.

Recall is the ratio of correctly drawn annotation to the total number of ground truth annotation.

F1 score gives us a harmonic mean between precision and recall.

```
Average score: 0.37  
Median score: 0.38  
Standard deviation: 0.31
```

```
Maximum score: 0.96  
Minimum score: 0
```

```
Precision: 0.55750  
Recall: 0.59834  
F1 score: 0.57720
```

```
Total IoU scores calculated: 2196  
Incorrect bounding boxes (IoU = 0): 595
```

Model Evaluation (TextOCR)

We used a larger subset for TextOCR, which itself is only a small fraction of the actual dataset. We can see that with more data, it performs much worse overall.

Recall and F1 score take a very big hit.

There are a lot less matches when it comes to making bounding box predictions.

```
Average score: 0.26  
Median score: 0.00  
Standard deviation: 0.34  
  
Maximum score: 0.98  
Minimum score: 0
```

```
Precision: 0.55190  
Recall: 0.35432  
F1 score: 0.21578
```

```
Total IoU scores calculated: 14730  
Incorrect bounding boxes (IoU = 0): 7398
```

Model Observations

The very low accuracy is likely due to the high difficulty of the datasets.

- The model does not perform well with image text that is transformed (rotated, warped, blurred, etc.) or uses an uncommon font
- However, many existing translation apps aren't typically expected to work with such images, so we don't find it to be a major problem

EasyOCR does not have full support for detecting handwritten text as of now (which may explain the font issues), which may present problems when using the model in our application.

Translation

(source: <https://pypi.org/project/googletrans/>)

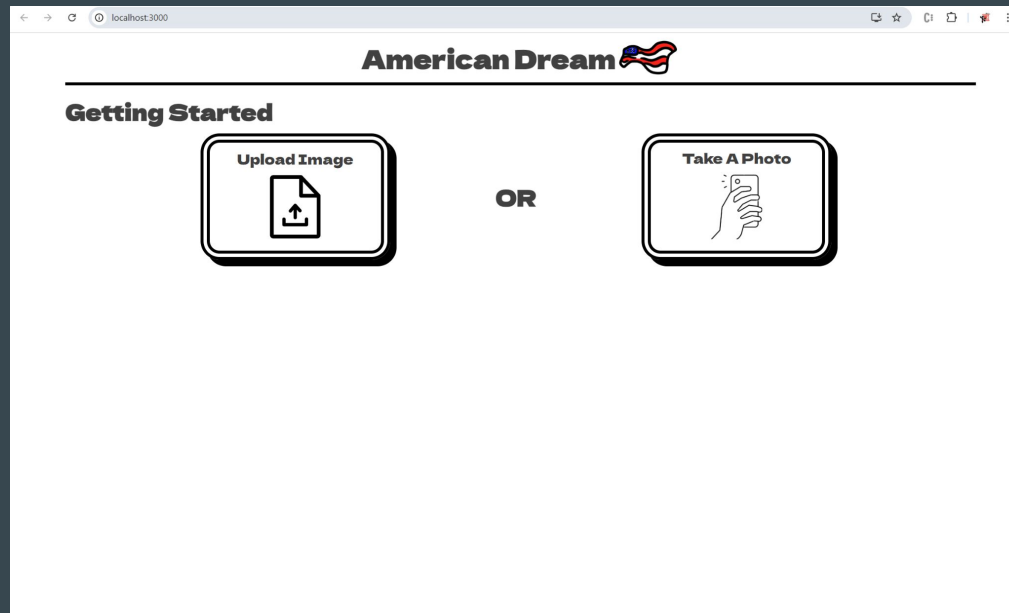
We are using googletrans, which is not a Google API, but rather an unofficial Python wrapper for the Google Translate website. This allows us to use Google Translate services easily with our model without having to pay for a Google API key. It also has the ability to detect text and provide a confidence score.

```
>>> from googletrans import Translator
>>> translator = Translator()
>>> translator.translate('안녕하세요.')
# <Translated src=ko dest=en text=Good evening. pronunciation=Good evening.>
>>> translator.translate('안녕하세요.', dest='ja')
# <Translated src=ko dest=ja text=こんにちは。 pronunciation=Kon'nichiwa.>
>>> translator.translate('veritas lux mea', src='la')
# <Translated src=la dest=en text=The truth is my light pronunciation=The truth is my light>
```

Frontend

We use React for the frontend.

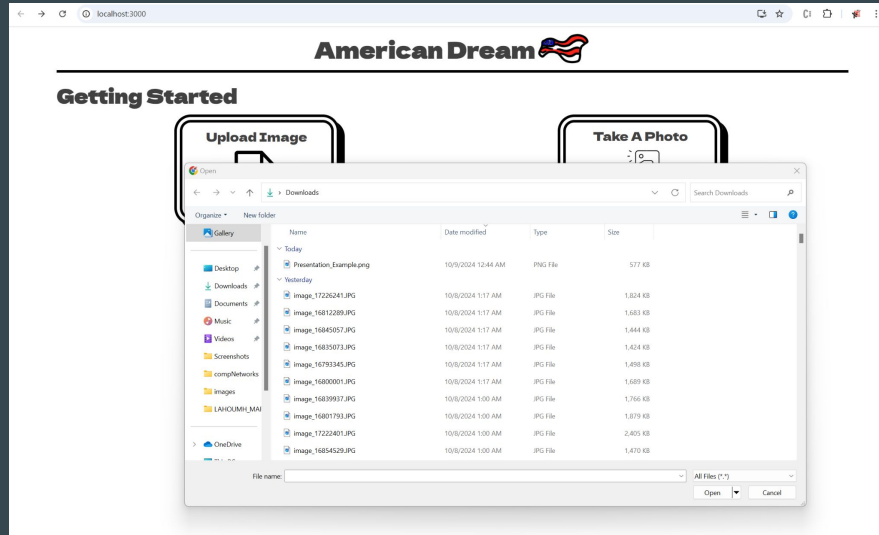
2 options, upload photo or capture a photo



Frontend

Upload Image option opens up users file system and allows them to choose a photo.

Upon selecting, the user is automatically directed to /translate page



Frontend

The /translate page creates a form of the image and the language that you type in.

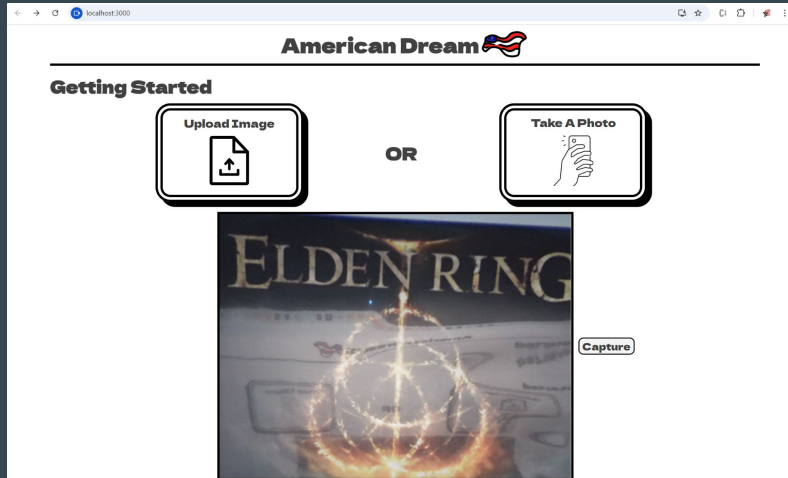
When you click on translate, it will send the data to the backend and run the image through our model



Frontend

Another way a user can share an image is by capturing the photo in real time.

When selecting 'Take A Photo', there webcam will open allowing them to capture an image



Backend

For the backend, we decided to use flask as it is easy to work with and it is one that most of us have knowledge in. Additionally, we thought it will be easier to connect the model to it as they are both written in python.

When a user uploads or captures a photo, that photo is sent to the /upload route in our backend

```
const uploadFile = async (file) => {
  const formData = new FormData();
  formData.append('file', file);

  try {
    const response = await axios.post('http://localhost:5000/upload', formData);
    console.log(response.data);
    //send user to /translate and send the imgPath with it so it can show up on next page
    navigate('/translate', { state: { imgPath: response.data.imgPath } });
  } catch (err) {
    console.error('Error uploading file:', err);
    alert('File upload failed');
  }
};
```

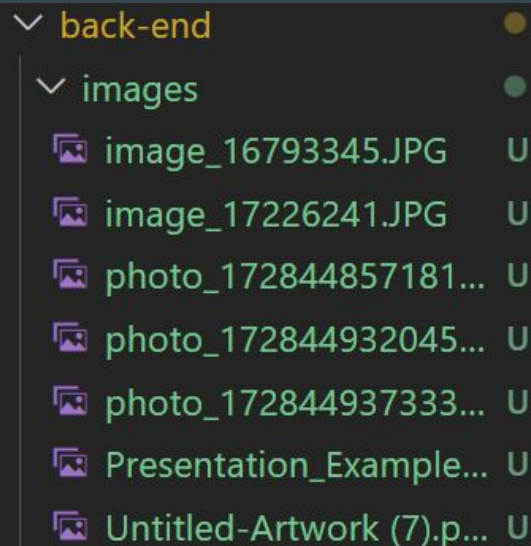
Backend

The image is then saved into the /images route and returned back to the front end so that their photo can be shown to them

```
@app.route('/upload', methods=['POST'])
def upload_file():
    #If no file sent, send back error
    if 'file' not in request.files:
        return jsonify({"error": "No File Selected"}), 400

    #If there was, get the file
    img = request.files['file']
    #If the file has no name, then there is nothing to save, send back an error
    if img.filename == '':
        return jsonify({"error": "Non Existing File"}), 400

    if img:
        #Save the image to the IMG_FOLDER path
        img.save(os.path.join(app.config['IMG_FOLDER'], img.filename))
        return jsonify({"imgPath": f"/{app.config['IMG_FOLDER']}/{img.filename}"}) , 200
```








Issues and Next Steps

If we are to continue working on this project, we would address the problems and implement the features below:

- Customize OCR model to recognize **transformed text** (rotation, shear, etc.)
- Fix errors with recognizing or translating certain **non-alphabetical languages**
- **Overlaying text** on image with proper text color and size
- **User account functionality** to save past inferences
- **Mobile application** interface

GitHub Repository

- GitHub is split up between the model work, documents, and the client component (web app)
- Read me updated with all progress reports

 Model Proposal client images .gitignore README.md

Conclusion and Reflection

All of us in this group have had no experience with machine learning development until this class. Even if our application uses a model we did not train, it was a big learning curve for us.

Given the time and resource constraints, we likely would still not have enough time to complete our application without using pretrained models, or at the very least accurate ones.

If we could attempt this project with more time and the knowledge we have learned now, we may be able to implement a full OCR pipeline from scratch.

Thank you!