

# Text to Speech

Bree Yates, Phillip Hirsch, Caleb Smith

# Introduction

## Group Members:

### 1. Bree Yates:

- a. Co-programmer, concept director

### 2. Phillip Hirsch:

- a. Dataset Collector

### 3. Caleb Smith:

- a. Lead Programmer

# Problem and Motivation for our project

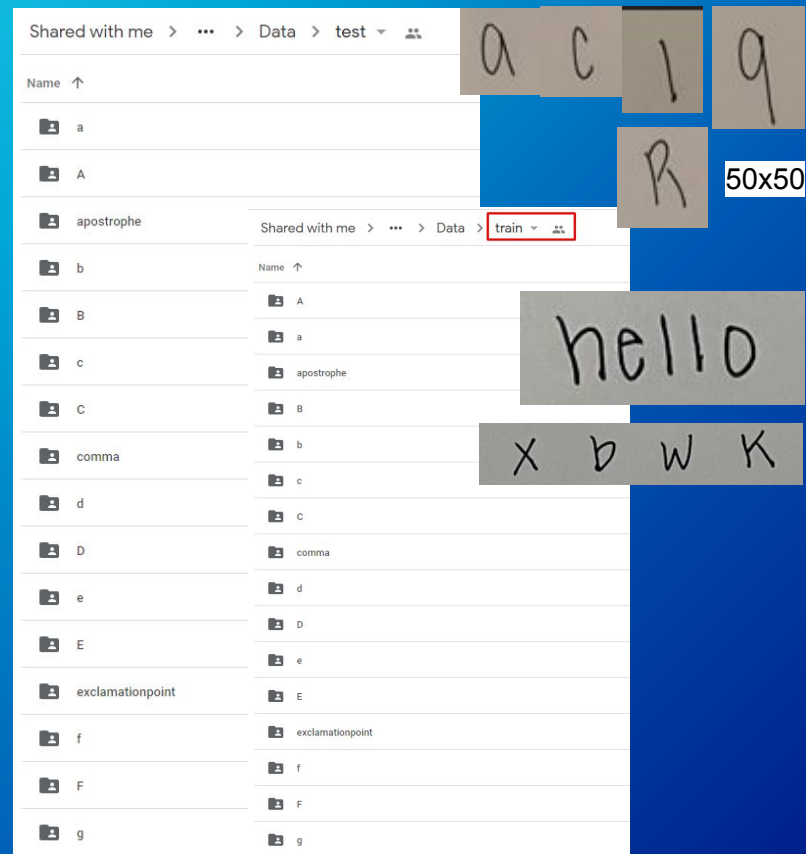
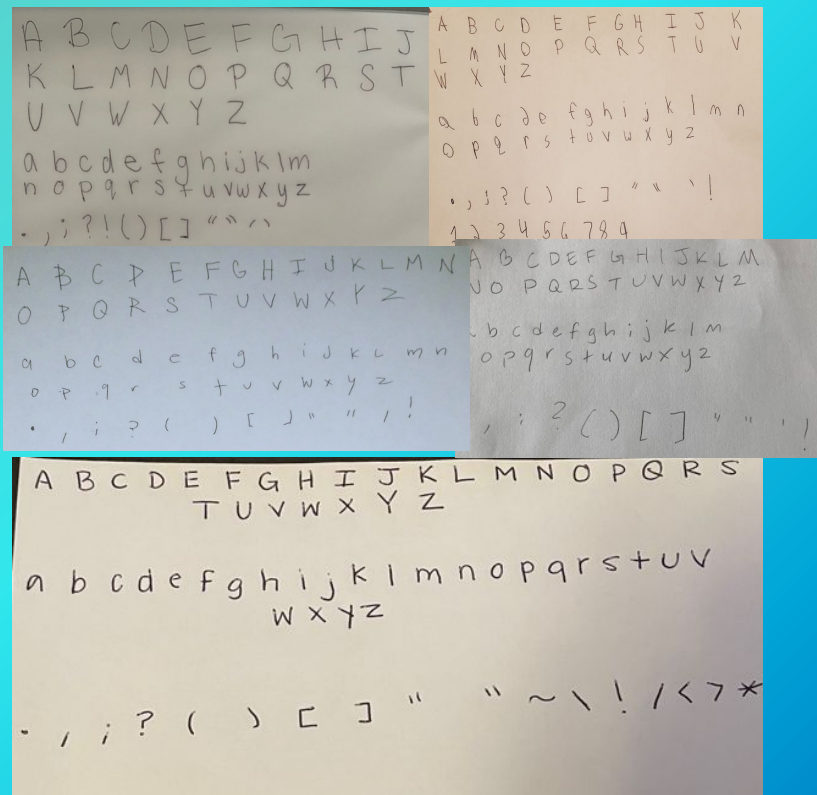
## Problem:

- Our solution is to assist the day to day life of the visually impaired and allow them to be more independent by comfortably navigating their environments. Our product is mostly focused on translating handwritten text to speech.

## Motivation:

- The American Foundation for the Blind notes that as of 2018 roughly 13% of adult Americans (32.2 million) are blind or have difficulty seeing.
- The NCBI also notes that data conducted in 2015 determined that 89% of those who are visually impaired live in low-middle income communities.
- Our product specifically targets these lower-income communities, including those without access to expensive OCR software.

# Our Dataset



# Our Dataset



A B C D E F G H I J

HELLO

# Our Dataset

## The EMNIST Dataset

### What is it?

The EMNIST dataset is a set of handwritten character digits derived from the [NIST Special Database 19](#) and converted to a 28x28 pixel image format and dataset structure that directly matches the [MNIST dataset](#). Further information on the dataset contents and conversion process can be found in the paper available at <https://arxiv.org/abs/1702.05373v1>.

### Formats

The dataset is provided in two file formats. Both versions of the dataset contain identical information, and are provided entirely for the sake of convenience. The first dataset is provided in a Matlab format that is accessible through both Matlab and Python (using the `scipy.io.loadmat` function). The second version of the dataset is provided in the same binary format as the original MNIST dataset as outlined in <http://yann.lecun.com/exdb/mnist/>.

### Dataset Summary

There are six different splits provided in this dataset. A short summary of the dataset is provided below:

- EMNIST ByClass: 814,255 characters. 62 unbalanced classes.
- EMNIST ByMerge: 814,255 characters. 47 unbalanced classes.
- EMNIST Balanced: 131,600 characters. 47 balanced classes.
- EMNIST Letters: 145,600 characters. 26 balanced classes.
- EMNIST Digits: 280,000 characters. 10 balanced classes.
- EMNIST MNIST: 70,000 characters. 10 balanced classes.

# FastAI?

```
[ ] model = cnn_learner(data, models.resnet50, metrics=accuracy, loss_func = nn.CrossEntropyLoss())
```

## ▾ Finding best learning rate for the model

▶ `model.lr_find()`

50.00% [1/2 11:15<11:15]

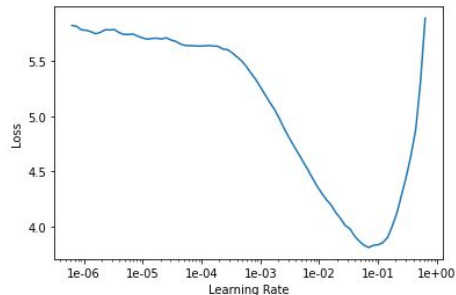
epoch	train_loss	valid_loss	accuracy	time
-------	------------	------------	----------	------

0	5.263926	#na#		11:15
---	----------	------	--	-------

76.47% [39/51 01:41<00:31 14.1875]

LR Finder is complete, type {learner\_name}.recorder.plot() to see the graph.

```
[ ] model.recorder.plot()
```

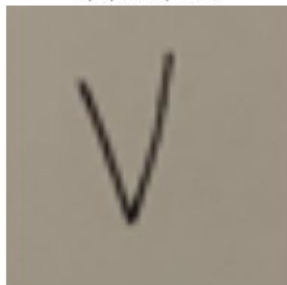




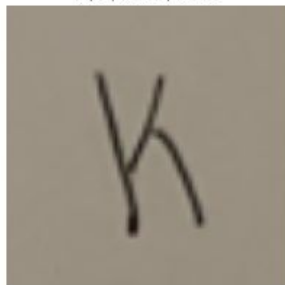
# Methodology

**Prediction/Actual/Loss/Probability**

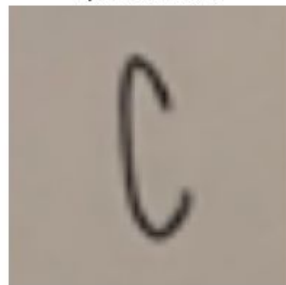
V/v / 7.40 / 0.00



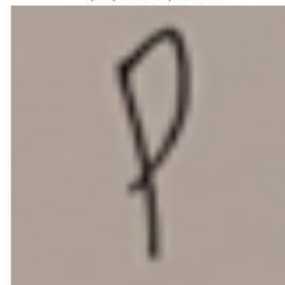
K/k / 7.14 / 0.00



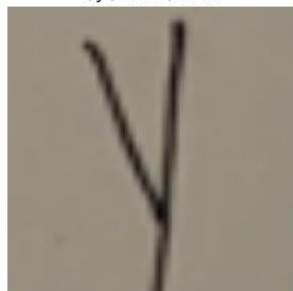
q/c / 6.91 / 0.00



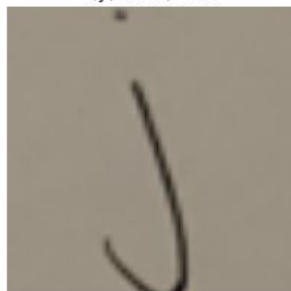
l/P / 6.74 / 0.00



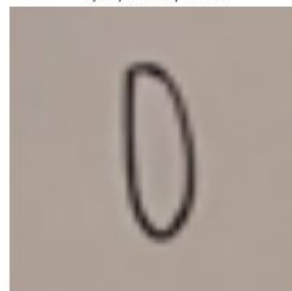
i/y / 6.63 / 0.00



L/j / 6.62 / 0.00



e/O / 6.45 / 0.00



semicolon/apostrophe / 5.77 / 0.00





# Methodology

Prediction/Actual/Loss/Probability

F/M / 8.82 / 0.00

z/t / 7.99 / 0.00

S/s / 6.93 / 0.00

k/t / 6.20 / 0.00

g/a / 5.87 / 0.00

h/b / 5.86 / 0.00

d/f / 5.68 / 0.00

Q/d / 5.57 / 0.00

y/U / 5.36 / 0.00

c/w / 5.27 / 0.01

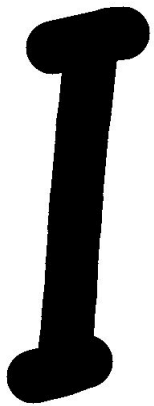
S/s / 5.18 / 0.01

j/f / 5.17 / 0.01

# Methodology

```
test_path = '/content/drive/MyDrive/Stage 4/Data/Img/img019-010.png'
test_img = cv2.imread(test_path)
pred, actual, _ = model.predict(open_image(test_path))
print(f'Predicted Label: {pred}')
#cv2_imshow(test_img)
bb_letter(test_img)
```

Predicted Label: I



# FastAI Results



```
lr = 1e-6
```

```
model.fit_one_cycle(8, slice(lr/10, lr))
```



epoch	train_loss	valid_loss	accuracy	time
-------	------------	------------	----------	------

0	0.547608	0.724975	0.807692	02:22
---	----------	----------	----------	-------

1	0.565245	0.737135	0.811189	02:24
---	----------	----------	----------	-------

2	0.572463	0.743994	0.804196	02:24
---	----------	----------	----------	-------

3	0.547550	0.723401	0.814685	02:25
---	----------	----------	----------	-------

4	0.553852	0.721093	0.818182	02:24
---	----------	----------	----------	-------

5	0.544028	0.724581	0.811189	02:24
---	----------	----------	----------	-------

6	0.516556	0.717181	0.821678	02:24
---	----------	----------	----------	-------

7	0.520729	0.720755	0.814685	02:23
---	----------	----------	----------	-------

```
[65] preds, targets = model.get_preds(ds_type=DatasetType.Valid)
acc = accuracy(preds, targets)
```

```
C
```

```
[66] cfm = interp.confusion_matrix()
cfm = np.nan_to_num(cfm)
```

```
FP = cfm.sum(axis=0) - np.diag(cfm)
FN = cfm.sum(axis=1) - np.diag(cfm)
TP = np.diag(cfm)
TN = cfm.sum() - (FP + FN + TP)
```

```
precision = (TP / (TP + FP))
recall = (TP / (TP + FN))
```

```
precision = np.nan_to_num(precision)
recall = np.nan_to_num(recall)
```



```
print(f"Percision: {precision.sum()} \nRecall: {recall.sum()} \nAccuracy: {acc * 100}%")
```

```
Percision: 43.356460206460206
Recall: 41.64754689754689
Accuracy: 81.4685287475586%
```

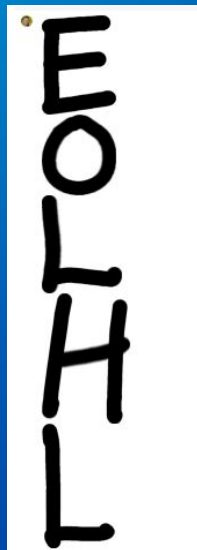
# FastAI Results

1) Trained our model using fastai

2)

```
[ ] def extract_letter(img):  
    inv = 255 - img  
    gray = cv2.cvtColor(inv, cv2.COLOR_BGR2GRAY)  
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU + cv2.THRESH_BINARY)[1]  
    contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  
    contours = contours[0] if len(contours) == 2 else contours[1]  
  
    index = 0  
    for box in contours:  
        x,y,w,h = cv2.boundingRect(box)  
        bb = img[y:y+h, x:x+w]  
        cv2.imshow(bb)  
        cv2.imwrite(f'/content/drive/MyDrive/ITCS 5152/PG-13: Text to Speech/Stage 4/Handwriting Test/temp/img_{index}.png', bb)  
        index += 1
```

3)



4)

Results from model

```
[ ] string = ''  
    for t in pred_letters:  
        string += str(t)  
    print(string)
```

BOLFI

# FastAI Results

Results from model

```
[79] string = ''  
    for t in pred_letters:  
        string += str(t)  
    print(string)
```

BOLFI



# PyTorch?

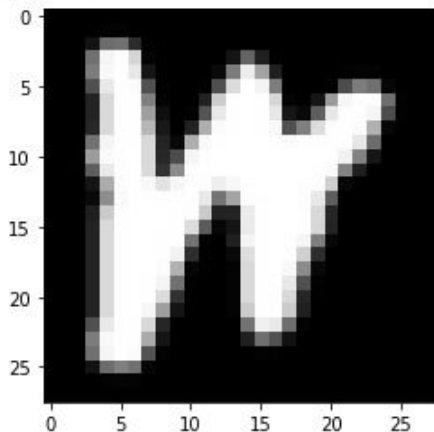
LeNet-5 implementation

```
✓ [8] model = nn.Sequential(nn.Conv2d(1, 6, kernel_size = 5, padding=(2,2)), nn.Tanh(),  
  nn.AvgPool2d(2),  
  nn.Conv2d(6, 16, kernel_size = 5), nn.Tanh(),  
  nn.AvgPool2d(2),  
  nn.Conv2d(16, 120, kernel_size=5), nn.Tanh(),  
  nn.Flatten(),  
  nn.Linear(120, 84), nn.Tanh(),  
  nn.Linear(84, 27))
```

# Method

```
[7] plt.imshow(img[0], cmap="gray")  
    print(f"Label: {labels[label]}")
```

Label: w



## ▾ Bounding Boxes

```
[12] def extract_letter(img):  
    inv = 255 - img  
    gray = cv2.cvtColor(inv, cv2.COLOR_BGR2GRAY)  
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU + cv2.THRESH_BINARY)[1]  
    contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  
    contours = contours[0] if len(contours) == 2 else contours[1]  
    contours.sort(key=lambda x: get_contour_precedence(x, thresh.shape[1]))  
  
    index = 0  
    for box in contours:  
        x, y, w, h = cv2.boundingRect(box)  
        bb = thresh[y:y + h, x:x + w]  
        bb = cv2.resize(bb, (28, 28))  
        cv2.imshow(bb)  
        cv2.imwrite(f'/content/drive/MyDrive/ITCS 5152/PG-13: Text to Speech/Stage 4/Handwriting Test/temp/a/img_{index}.png', bb)  
        index += 1  
  
    def get_contour_precedence(contour, cols):  
        tolerance_factor = 30  
        origin = cv2.boundingRect(contour)  
        return ((origin[1] // tolerance_factor) * tolerance_factor) * cols + origin[0]  
  
[25] test_path = '/content/drive/MyDrive/PG-13: Text to Speech/Stage 4/Handwriting Test/hello3.png'  
    test_img = cv2.imread(test_path)  
  
    extract_letter(test_img)
```



# PyTorch Results

```
Epoch: 1, training loss = 0.415, valid accuracy = 0.828
Epoch: 2, training loss = 0.216, valid accuracy = 0.922
Epoch: 3, training loss = 0.123, valid accuracy = 0.938
Epoch: 4, training loss = 0.098, valid accuracy = 0.953
Epoch: 5, training loss = 0.108, valid accuracy = 0.906
Epoch: 6, training loss = 0.168, valid accuracy = 0.938
Epoch: 7, training loss = 0.155, valid accuracy = 0.891
Epoch: 8, training loss = 0.313, valid accuracy = 0.953
Epoch: 9, training loss = 0.225, valid accuracy = 0.891
Epoch: 10, training loss = 0.237, valid accuracy = 0.922
```

```
Predicted Label: m
Predicted Label: l
Predicted Label: e
Predicted Label: l
Predicted Label: o
```

```
string = ''
for t in pred_labels:
    string += labels[t]
print(string)

mlelo
```

```
test_path = '/content/drive/MyDrive/PG-13: Text to Speech/Stage 4/Handwriting Test/hello3.png'
test_img = cv2.imread(test_path)

extract_letter(test_img)
```



hello



# Conclusion

- We learned later on that using softmax over MSE was better suited for our type of problem.
- Our project was way harder than we thought it would be. Although we were able to do character recognition, we could not accurately compute words.
- It's *very* difficult to train a network to understand handwriting.

**Any Questions?**