Code

https://github.com/Viraj-Joshi/MLDS-EOCR https://colab.research.google.com/drive/1nhbb48cq19swmoiRfQ78UXvprjV9v8lj?usp=sharing

Language Choice

To me, this task seemed less of an EDA one at first, so I did not see the reason for R and went with the far superior modeling libraries in Python, and more specifically PyTorch.

Colab was used for visualization and their GPU.

Data

With regard to the data, I did no cleaning, throwing out, or anything of the sort. I used it in its entirety without attempts to decode meaning by eye.

I did reconstruct the rows of the csv as images(because I wanted to apply image transforms) and upload them to CS servers for fast access.

Model

The architecture of the model is quite big considering it only process 20x20 images, but as this was my first run through, I erred on the side of more complexity.

I think looking at the code is the best way to explain how this works, but essentially init() of models.py sets up the construction of the network. forward() will evaluate a batch of images and return the model output, the class predictions. There are Python classes which simply allow resuse of repeated blocks of the network.

The part of the code implementing skip connections and seemingly strange attempts to fix padding (line 58) has to do with how convolutions work, but there a lot of great resources online.

Training

train.py is the code involved in training of the network. Luckily, Pytorch will randomnly batch and apply transforms for us so long as we use the DataLoader object.

I trained until the tensorboard indicated no improvement. Training was slow, but as it was my first attempt, there was no attempt to optimize for future runs.

Points to make

1. https://arxiv.org/abs/1708.02002 - is a paper describing focal loss. Essentially, my network was designed for segmentation, so I used this technique to improve performance. I adapted it to classification by global average pooling the output.
2. Confusion Matrices and accuracy logging was crucial to seeing when I should stop training.

Metrics over Epochs

1. 83.7% global accuracy after 1 hour of training
2. 96.8% global accuracy after an additional 3 hours 50 minutes
3. 98.5% global accuracy after an additional 30 minutes

Using HMM

The wikipedia article on the viterbi algorithm was defintely an interesting read as I just had taken Stochastic Processes that semester.

I used it as a means to correct the predictions my model had made on the test set.

Using their notation,

1. the transition matrix A is made such that an entry p_{ij} is the probability of transitioning from class i to class j. I used the ground truth labels to construct this.
2. The initial conditions C is a normalized array of the ground truth's frequency counts.
3. The emission matrix B is the last confusion matrix outputted during training.
4. The O array are the predictions.

Look at eval.py and utils.py for this.

In this case, I thought of the hidden process as the true labels my model should predict and the observed process as my model's predictions.

Improvements

1. split original dataset into training and test
2. see if the viterbi algorithm made a kaggle score difference
3. More image transformations such as rotations,contrast,etc.

Code for HMM

In [ ]:
```
def generate_matrices():
    LABEL_NAMES = [str(i) for i in range(0,43)]
    states = []
    with open('labels.csv') as f:
        reader = csv.reader(f)
        for fname, label in reader:
            if label in LABEL_NAMES:
                states.append(int(label))

    def transition_matrix(transitions):
        n = 1+ max(transitions) #number of states

        M = [[0]*n for _ in range(n)]

        for (i,j) in zip(transitions,transitions[1:]):
            M[i][j] += 1

        #now convert to probabilities:
        for row in M:
            s = sum(row)
            if s > 0:
                row[:] = [f/s for f in row]
        return M
    def initial_conditions(states):
        I = [0] * 43
        for i in states:
            I[i]+=1
        return np.array(I)/len(states)

    M = transition_matrix(states)
    I = initial_conditions(states)
    # for row in M: print(' '.join('{0:.2f}'.format(x) for x in row))

    return M, I
```

and that's it!

In [ ]:
```
A,C = generate_matrices()
A = np.array(A)
C = np.array(C)
B = np.array(EMISSION_MATRIX)
S_opt, D_log, E = viterbi_log(A, C, B, O)

predictions = pd.DataFrame(data = S_opt, columns = ["category"], index = range(0,28051))
predictions.index.name = 'Id'
predictions.to_csv("predictions_HMM.csv")
```