Final Project:

By AJ, Tyler, and Justus

```
_time = time.time()

self.load(self.checkpoint_dir):
    print(" [*] Load SUCCESS")
else:
    print(" [!] Load failed...")

for epoch in xrange(config.epoch):
    if config.dataset == 'mnist':
        batch_idxs = min(len(data_X), config.train_s
    else:
        data = glob(os.path.join("./data", config.da
        batch_idxs = min(len(data), config.train_siz

    for idx in xrange(0, batch_idxs):
        if config.dataset == 'mnist':
            batch_images = data_X[idx*config.batch_s
            batch_labels = data_y[idx*config.batch_s
        else:
            batch_files = data[idx*config.batch_size
            batch = [get_image(batch_file, self.ima
```

Our Question:

Can we get an AI to recognize numbers in the MNIST data set?

Our Algorithms

We used a CNN, and used a total of 6 different architecture. Each of our models had 3 convo2d layers, and 2 dense layers.

4x4 Filter

Each of the 3 convo2d Layers have a filter size of 4x4

2x2 Filter

Each of the 3 convo2d Layers have a filter size of 2x2

3x3 Filter

Each of the 3 convo2d Layers have a filter size of 3x3

MaxPooling

The original code written by Tyler uses "MaxPooling" and "Dropout" Layers. I had one version of each model with and without the pooling and dropout layers

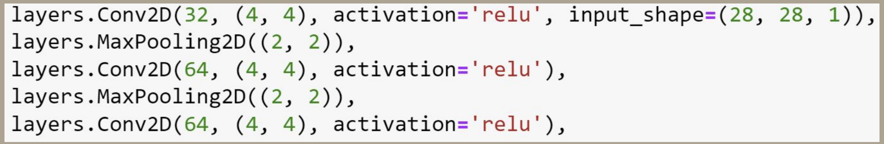
MaxPooling and Dropout Layers

MaxPooling

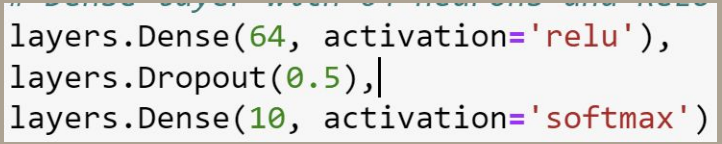
MaxPooling simplifies the data, making it easier for future layers to work with it, and make the model less prone to overfitting.

Dropout

Sets random neuron values to 0 during training to prevent overfitting.



```
layers.Conv2D(32, (4, 4), activation='relu', input_shape=(28, 28, 1)),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (4, 4), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (4, 4), activation='relu'),
```



```
layers.Dense(64, activation='relu'),
layers.Dropout(0.5),
layers.Dense(10, activation='softmax')
```

Measurements

Accuracy

How many of the testing data set it correctly identifies, measures how successfully we trained the algorithm.

Time to Learn

The time it takes to learn from the data set, as it taking too long can be a problem.

Performance

2x2 pooling

Test accuracy: 0.9921
Time elapsed: 45.48 seconds

3x3 pooling

Test accuracy: 0.9919
Time elapsed: 79.79 seconds

4x4 pooling

Test accuracy: 0.9932
Time elapsed: 46.82 seconds

2x2 no pooling

Test accuracy: 0.9871
Time elapsed: 219.68 seconds

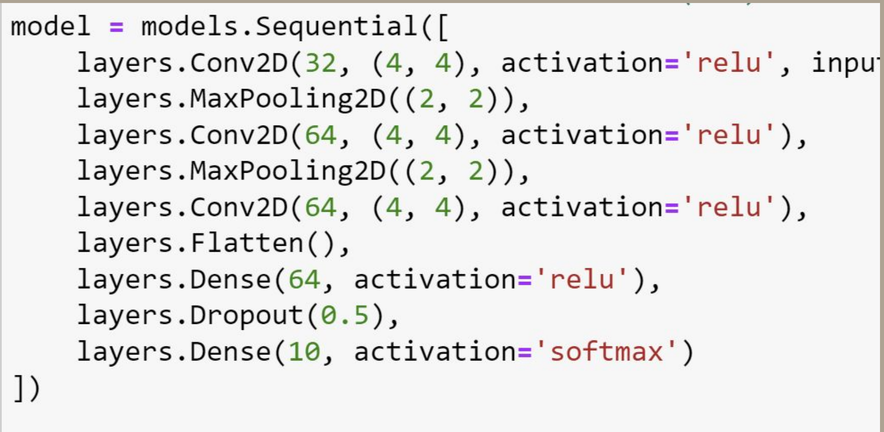
3x3 no pooling

Test accuracy: 0.9892
Time elapsed: 236.42 seconds

4x4 no pooling

Test accuracy: 0.9853
Time elapsed: 264.92 seconds

Best algorithm



```
model = models.Sequential([
    layers.Conv2D(32, (4, 4), activation='relu', inpu
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (4, 4), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (4, 4), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

We found that the best algorithm was a model with 4x4 filters, as it had the highest percentage correct at 99.32%, and barely had a worse time than 2x2 filters.

All the Pooling models also did better than the no pooling models, as all the pooling models got about 99% accuracy, but none of the no pooling models did, and the no pooling models took much longer.