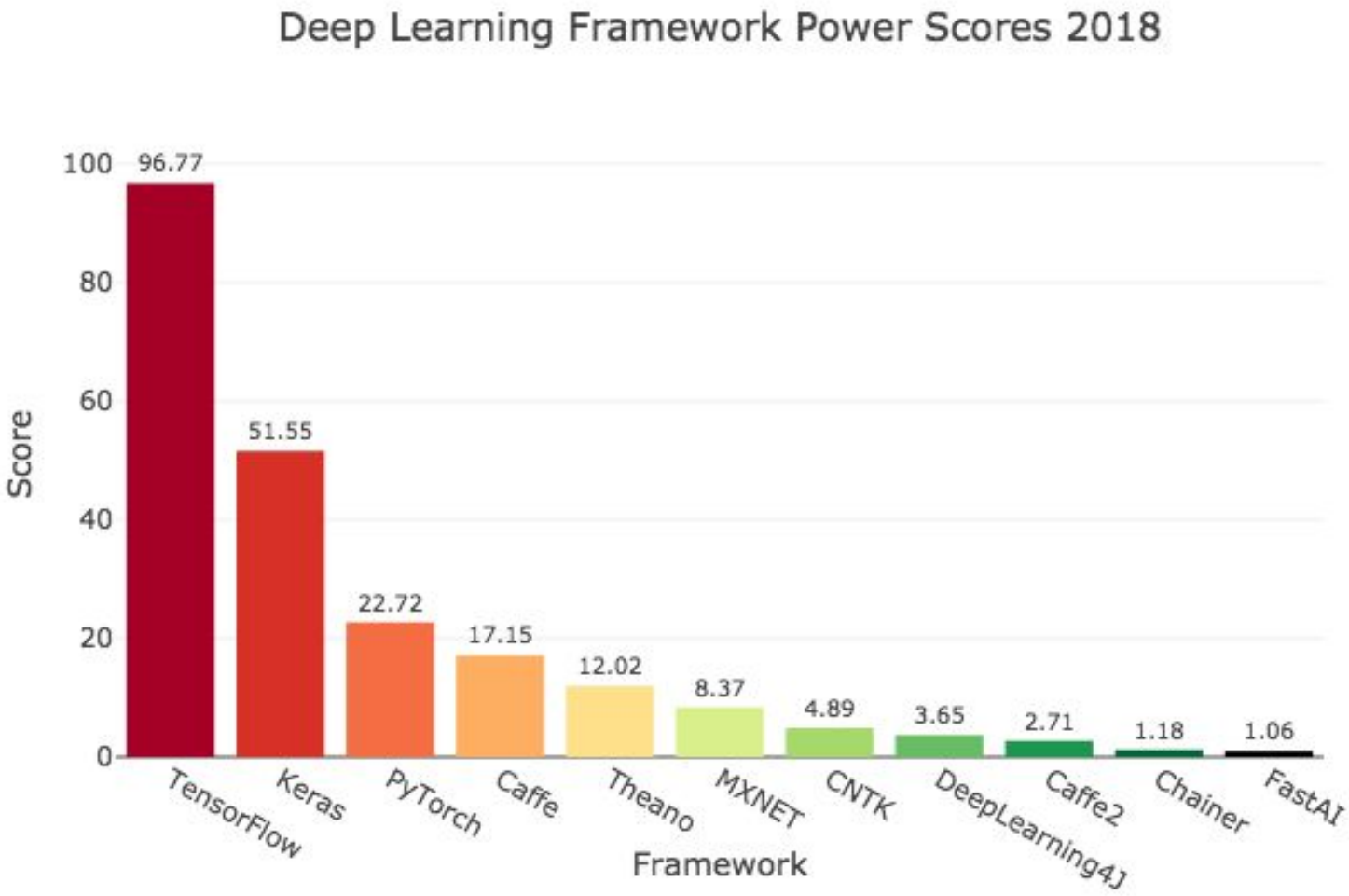
Deep Learning Tutorial # 2 CNN using Keras

CS 5102 Deep Learning

Agenda

1. Train a model that would differentiate images of Cats from Images of Dogs.
2. Implement CNN using Keras
3. Project - composed of 3 parts, one week per part.


Deep Learning Framework Power Scores 2018
Score
100
80
60
40
20
0
96.77
51.55
22.72
17.15
12.02
8.37
4.89
3.65
2.71
1.18
1.06
TensorFlow
Keras
PyTorch
Caffe
Theano
MXNET
CNTK
DeepLearning4J
Caffe2
Chainer
FastAI
Framework

Dataset - Dogs Vs. Cats

1. Create an algorithm to distinguish dogs from cats.
2. Contains 25,000 images of dogs and cats.
3. Approach: Preprocessing followed by model training

Dogs Vs. Cats

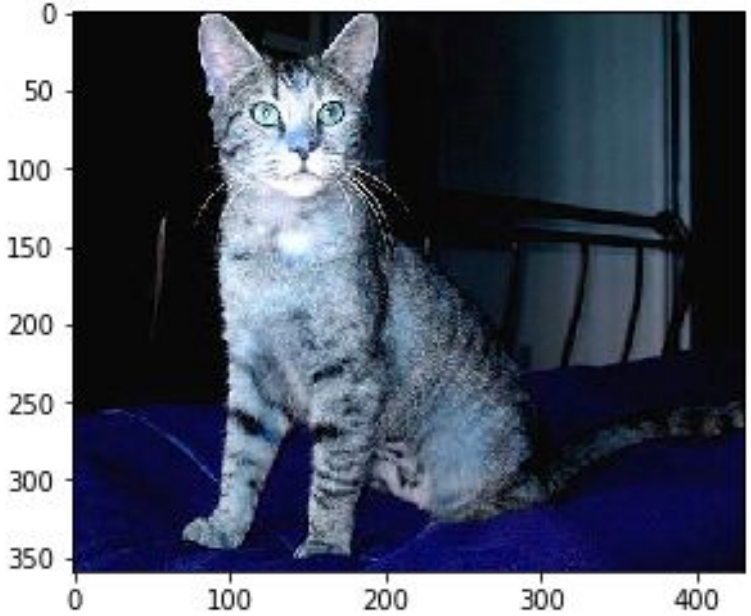
```
In [2]: import numpy as np
   ...: import pandas as pd
   ...: import cv2
   ...: import os
   ...: import matplotlib.pyplot as plt

In [3]: train = 'G:/Dogs vs. Cats/input/train/'
   ...: test = 'G:/Dogs vs. Cats/input/test1/'
```

Dogs Vs. Cats

```
In [5]: # Global Variables
   ...: data = []
   ...: test_data = []
   ...: test_id = []
   ...: label = []
   ...: im_width = 64
   ...: im_height = 64
   ...: train_image_files = [ f for f in os.listdir(train)
   ...:                       if os.path.isfile(os.path.join(train,f)) ]
   ...: test_image_files = [ f for f in os.listdir(test)
   ...:                       if os.path.isfile(os.path.join(test,f)) ]
   ...:
```

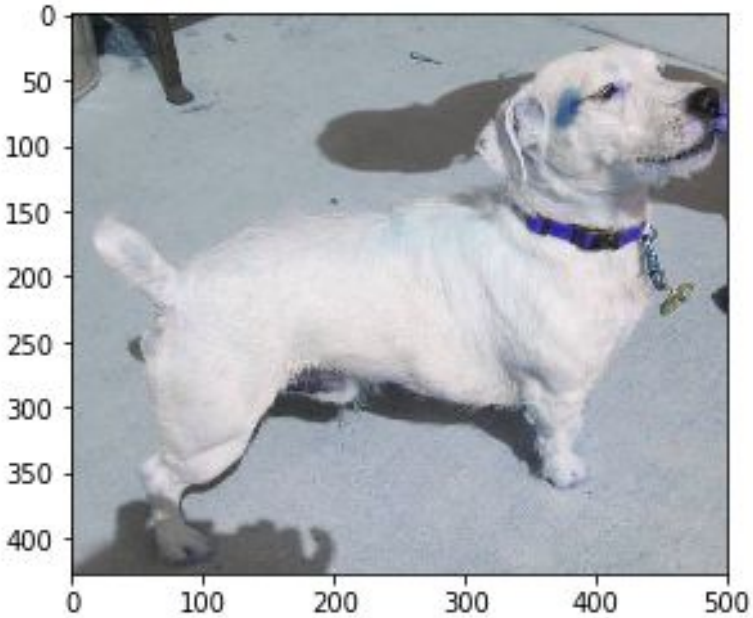
Dogs Vs. Cats

```
In [6]: image_file = str(train + train_image_files[5])
   ...: img = cv2.imread(image_file)
   ...: plt.imshow(img)
   ...:
Out[6]: <matplotlib.image.AxesImage at 0x1fa89a8ce10>
```



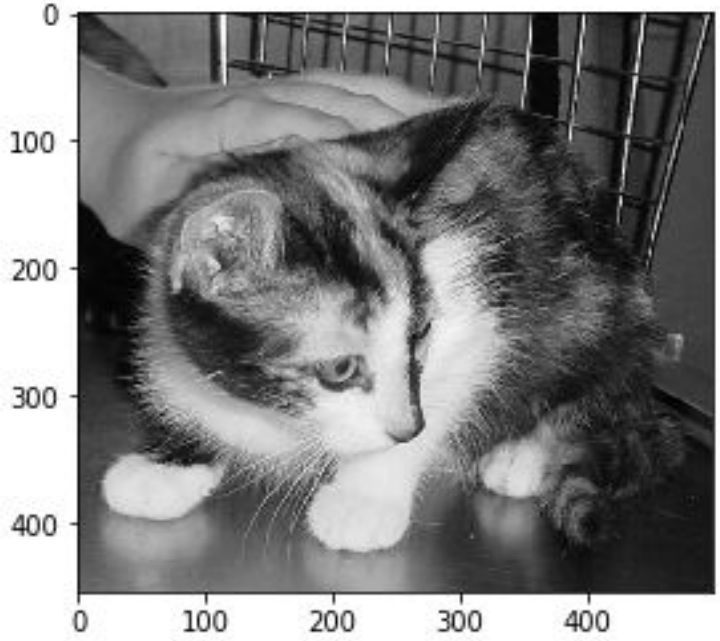
Dogs Vs. Cats

```
In [7]: image_file = str(train + train_image_files[22000])
   ...: img = cv2.imread(image_file)
   ...: plt.imshow(img)
   ...:
Out[7]: <matplotlib.image.AxesImage at 0x1fa8996ae48>
```



Dogs Vs. Cats

```
In [13]: image_file = str(train + train_image_files[24])
    ...: img = cv2.imread(image_file, cv2.IMREAD_GRAYSCALE) # read a grayscale image
    ...: plt.imshow(img, cmap='gray') # show a grayscale image
Out[13]: <matplotlib.image.AxesImage at 0x143de8f7fd0>
```



Dogs Vs. Cats

```
In [8]: def preprocessing(path):
   ...:     progress = 0
   ...:     image_files = [ f for f in os.listdir(path) if os.path.isfile(os.path.join(path,f)) ]
   ...:
   ...:     for file_name in image_files:
   ...:         image_file = str(path + file_name)
   ...:
   ...:         img = cv2.imread(image_file, cv2.IMREAD_GRAYSCALE)
   ...:         new_img = cv2.resize(img, (im_width, im_height))
   ...:
   ...:         if file_name[:3] == 'cat':
   ...:             label.append(0)
   ...:
   ...:         elif file_name[:3] == 'dog':
   ...:             label.append(1)
   ...:
   ...:         progress = progress + 1
   ...:         if progress % 1000 == 0:
   ...:             print('Progress: ' + str(progress) + ' Images Done')
   ...:
   ...:     print(len(data))
   ...:     print(len(label))
```

Dogs Vs. Cats

```
In [3]: preprocessing(train)
Progress: 1000 Images Done
Progress: 2000 Images Done
Progress: 3000 Images Done
Progress: 4000 Images Done
Progress: 5000 Images Done
Progress: 6000 Images Done
Progress: 7000 Images Done
Progress: 8000 Images Done
Progress: 9000 Images Done
Progress: 10000 Images Done
Progress: 11000 Images Done
Progress: 12000 Images Done
Progress: 13000 Images Done
Progress: 14000 Images Done
Progress: 15000 Images Done
Progress: 16000 Images Done
Progress: 17000 Images Done
Progress: 18000 Images Done
Progress: 19000 Images Done
Progress: 20000 Images Done
Progress: 21000 Images Done
Progress: 22000 Images Done
Progress: 23000 Images Done
Progress: 24000 Images Done
Progress: 25000 Images Done
25000
25000
```

CNN using Keras with Tensorflow Backend

```
In [4]: from keras import Sequential
   ...: from keras.layers import Dense,MaxPooling2D,Conv2D,Flatten,Dropout
   ...:
   ...: model = Sequential() #  linear stack of layers.
   ...:
   ...: #  Stride: stride length of the convolution
   ...: #  Padding:
   ...: #  A zero padding is used such that the output has the same length as the original input.
   ...: #  One of "valid", "causal" or "same"
   ...: model.add(Conv2D(kernel_size=(3,3),filters=3,input_shape=(im_width, im_height, 1),
   ...:           activation="relu",padding="valid"))
   ...: model.add(Conv2D(kernel_size=(3,3),filters=10,activation="relu",padding="same"))
   ...: model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
   ...:
   ...: model.add(Conv2D(kernel_size=(3,3),filters=3,activation="relu",padding="same"))
   ...: model.add(Conv2D(kernel_size=(5,5),filters=5,activation="relu",padding="same"))
   ...: model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
Using TensorFlow backend.
```

CNN using Keras with Tensorflow Backend

```
In [5]: model.add(Conv2D(kernel_size=(2,2),strides=(2,2),filters=10))
   ...:
   ...: model.add(Flatten())
   ...: model.add(Dropout(0.2))
   ...: model.add(Dense(100,activation="sigmoid"))
   ...: model.add(Dense(1,activation="sigmoid"))
```

CNN using Keras with Tensorflow Backend

```
In [6]: model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 62, 62, 3)	30
conv2d_2 (Conv2D)	(None, 62, 62, 10)	280
max_pooling2d_1 (MaxPooling2	(None, 31, 31, 10)	0
conv2d_3 (Conv2D)	(None, 31, 31, 3)	273
conv2d_4 (Conv2D)	(None, 31, 31, 5)	380
max_pooling2d_2 (MaxPooling2	(None, 15, 15, 5)	0
conv2d_5 (Conv2D)	(None, 7, 7, 10)	210
flatten_1 (Flatten)	(None, 490)	0
dropout_1 (Dropout)	(None, 490)	0
dense_1 (Dense)	(None, 100)	49100
dense_2 (Dense)	(None, 1)	101

```
Total params: 50,374
Trainable params: 50,374
Non-trainable params: 0
```

CNN using Keras with Tensorflow Backend

```
In [7]: model.compile(optimizer="adadelta",loss="binary_crossentropy",metrics=["accuracy"])

In [8]: data = np.array(data)
   ...: print(data.shape)
(25000, 64, 64)

In [9]: data = data.reshape((data.shape)[0], (data.shape)[1], (data.shape)[2], 1)
   ...: print(data.shape)
(25000, 64, 64, 1)

In [10]: label = np.array(label)
    ...: print(label.shape)
(25000,)
```

CNN using Keras with Tensorflow Backend

```
In [12]: model.fit(data, label, batch_size=128, epochs=2, verbose=1, validation_split=0.3)
Train on 17500 samples, validate on 7500 samples
Epoch 1/2
17500/17500 [==============================] - 856s 49ms/step - loss: 0.6153 - acc: 0.7009 - val_loss: 1.2167 - val_acc: 0.0000e+00
Epoch 2/2
17500/17500 [==============================] - 851s 49ms/step - loss: 0.5975 - acc: 0.7140 - val_loss: 1.0837 - val_acc: 0.0000e+00
Out[12]: <keras.callbacks.History at 0x143deba02b0>
```

CNN using Keras with Tensorflow Backend

```
In [14]: # model.save("G:/Dogs vs. Cats/keras_model_attempt_1.h5")

In [15]: # model = load_model("G:/Dogs vs. Cats/keras_model_attempt_1.h5")

In [16]: def preprocessing_test(path):
    ...:     progress = 0
    ...:
    ...:     for file in test_image_files:
    ...:         image_file = str(path + file)
    ...:
    ...:         img = cv2.imread(image_file, cv2.IMREAD_GRAYSCALE)
    ...:         new_img = cv2.resize(img, (im_width, im_height))
    ...:
    ...:         test_id.append(file[:-4])
    ...:         test_data.append(new_img / 255)
    ...:
    ...:         progress = progress + 1
    ...:         if progress % 1000 == 0:
    ...:             print('Progress: ' + str(progress) + ' Images Done')
    ...:
    ...:     print(len(test_data))
    ...:     print(len(test_id))
    ...:
```

CNN using Keras with Tensorflow Backend

```
In [17]: preprocessing_test(test)
Progress: 1000 Images Done
Progress: 2000 Images Done
Progress: 3000 Images Done
Progress: 4000 Images Done
Progress: 5000 Images Done
Progress: 6000 Images Done
Progress: 7000 Images Done
Progress: 8000 Images Done
Progress: 9000 Images Done
Progress: 10000 Images Done
Progress: 11000 Images Done
Progress: 12000 Images Done
12500
12500

In [18]: test_data = np.array(test_data)
    ...: print(test_data.shape)
    ...:
(12500, 64, 64)

In [19]: test_data = test_data.reshape((test_data.shape)[0], (test_data.shape)[1], (test_data.shape)[2], 1)
    ...: print(test_data.shape)
    ...:
(12500, 64, 64, 1)
```

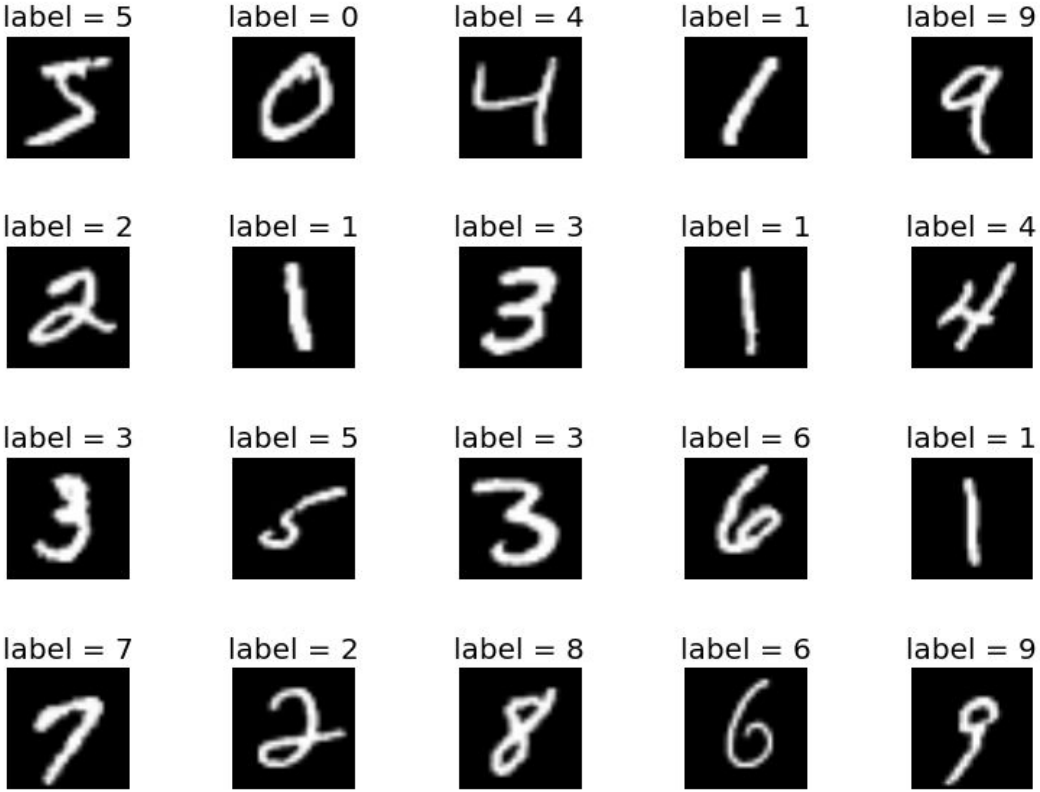
CNN using Keras with Tensorflow Backend

```
In [20]: predicted_labels = model.predict(test_data)

In [21]: final_labels = []
    ...: for value in predicted_labels:
    ...:     if value > 0.5:
    ...:         final_labels.append(1)
    ...:     else:
    ...:         final_labels.append(0)
    ...:

In [22]: final_submission = pd.DataFrame({"id" : test_id })

In [23]: final_submission["label"] = final_labels

In [24]: final_submission.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12500 entries, 0 to 12499
Data columns (total 2 columns):
id       12500 non-null object
label    12500 non-null int64
dtypes: int64(1), object(1)
memory usage: 195.4+ KB
```

CNN using Keras with Tensorflow Backend

```
In [25]: final_submission.head()
Out[25]:
      id  label
0      1      0
1     10      0
2    100      0
3   1000      0
4  10000      0
```

Project One - MNIST Classifier



Project One - MNIST Classifier

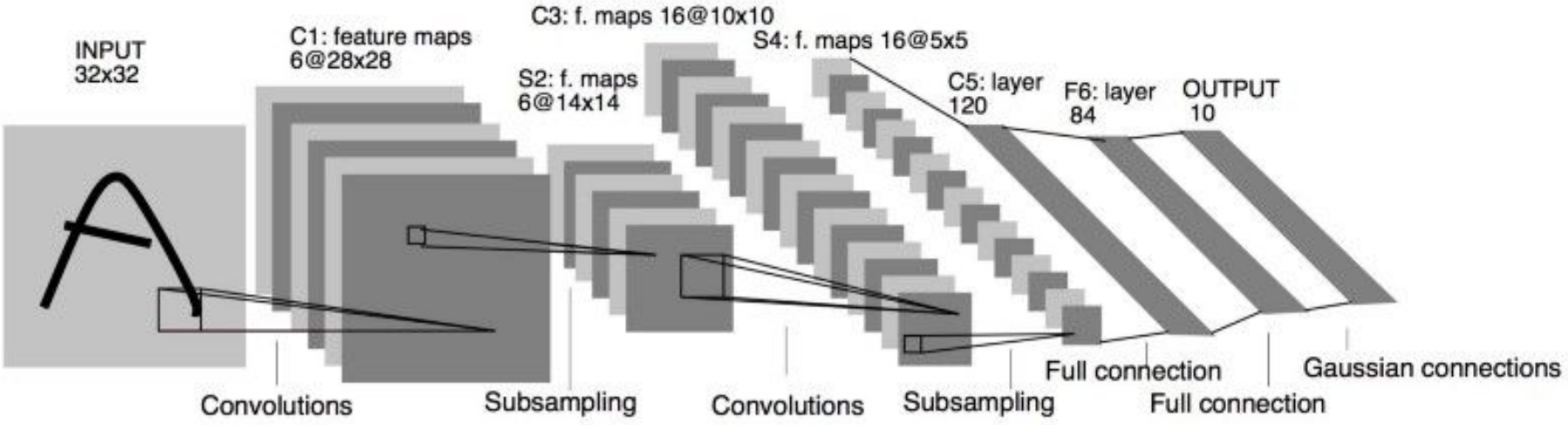
1. Composed of Three Parts
2. Each Part has a separate deadline
3. Dataset: MNIST Digit Dataset
4. The MNIST dataset comprises 60,000 training examples and 10,000 test examples of the handwritten digits 0–9, formatted as 28x28-pixel monochrome images.
5. Source: http://yann.lecun.com/exdb/mnist/
6. Alternate Source:

```
In [28]: from keras.datasets import mnist
    ...: (x_train, y_train), (x_test, y_test) = mnist.load_data()
    ...:
```

Project One - MNIST Classifier

Part 1: Convolutional Neural Network using Keras with Tensorflow Backend.

Implement CNN in Keras following the LeNet Architecture from Yann LeCun.



Deadline: 1st November 2018

Project One - MNIST Classifier

Part 2: Visualization:

1. Plot error vs epochs.
2. Visualize the architecture and plot the weights.
3. Use Tensorboard.
4. Save results as an image.

Deadline: 8th November 2018

Resources:

1. http://fizzylogic.nl/2017/05/08/monitor-progress-of-your-keras-based-neural-network-using-tensorboard/
2. https://keras.io/callbacks/
3. https://www.youtube.com/watch?v=eBbEDRsCmv4

Project One - MNIST Classifier

Part 3: Hyperparameter Tuning:

1. Common training problems: oscillating error, overfit.
2. Readjust the learning rate, weight initialization, and momentum.
3. Identify Overfit and Underfit.
4. Recreate the architecture with minimum layers while accuracy remains more than 90 %.
5. Recreate the architecture with maximum layers until the accuracy no longer improves significantly.
6. Write a report, more instructions will be sent shortly.

Deadline: 15th November 2018