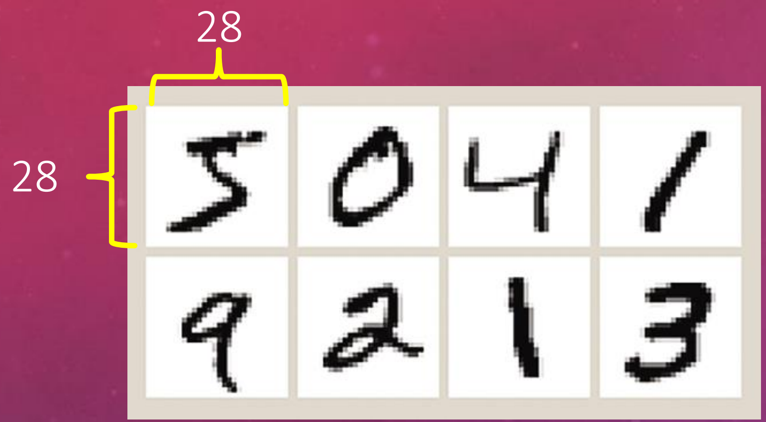
# 進階影像處理
# ADVANCED IMAGE PROCESSING

**張家瑋** 博士
助理教授
國立臺中科技大學資訊工程系

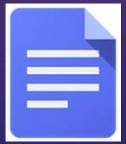
# MNIST 手寫數字辨識

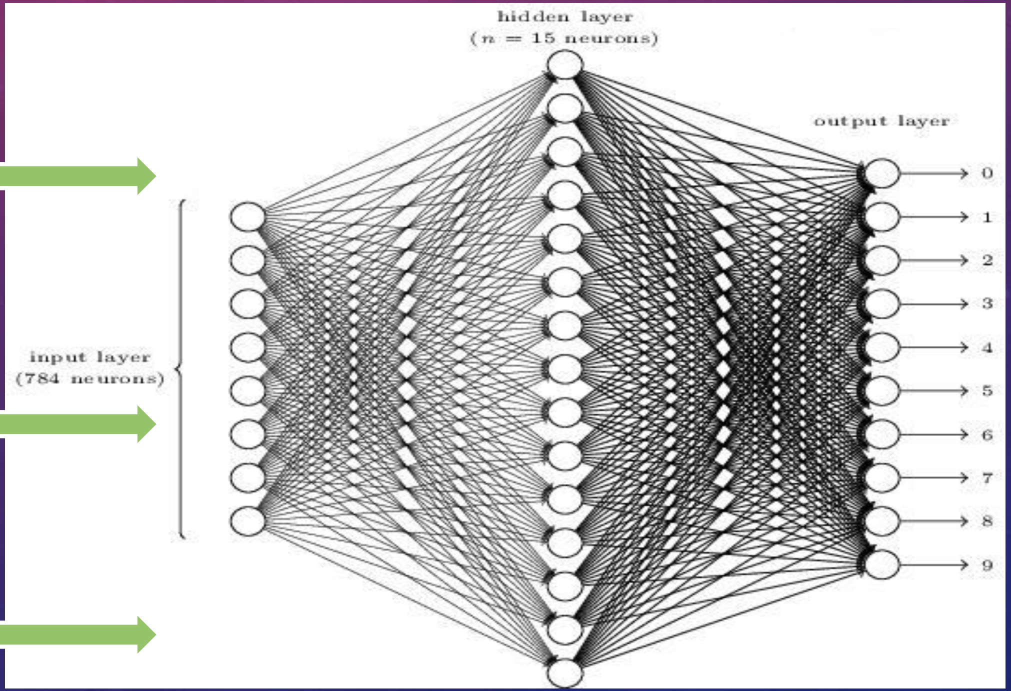



訓練資料

驗證資料

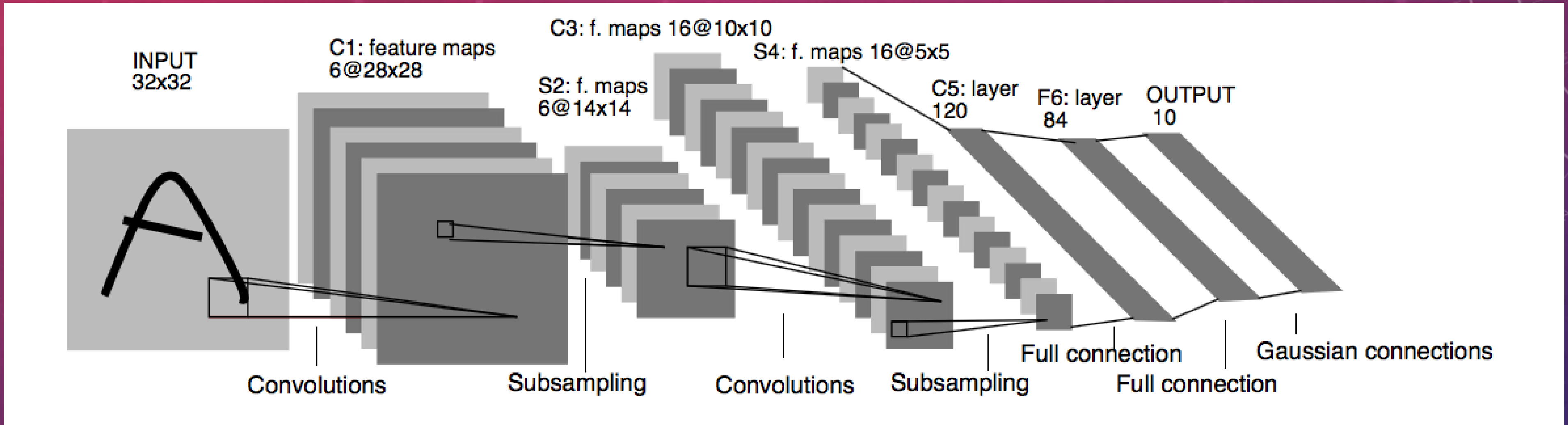
實際資料

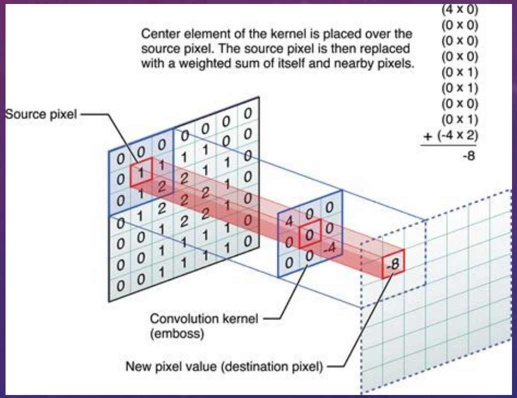


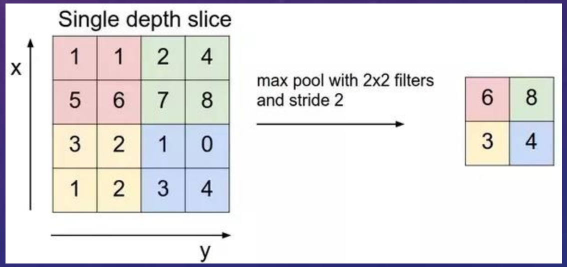



以最簡單的類神經網路架構，可達 **91%** 辨識率。若使用CNN則可高達 **99%** 辨識率。

# 卷積類神經網路




Max Pooing

Convolution

# 以 CNN 實作 - KERAS

- Step 1. 載入必要函式庫

```
import numpy as np
import matplotlib.pyplot as plt

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers import Conv2D, MaxPool2D, Flatten
from keras.utils import np_utils
```

- Step 2. 下載 MNIST 數據

```
nb_classes = 10
(x_train, y_train), (x_test, y_test) = mnist.load_data()

print(type(x_train))
print("x_train shape", x_train.shape)
print("y_train shape", y_train.shape)
```

# 以 CNN 實作 - KERAS

- Step 3. 顯示圖片

```
fig = plt.figure()
plt.subplot(2,1,1)
plt.imshow(x_train[0], cmap="binary",
interpolation="none")
plt.title("image" + str(y_train[0]))
plt.subplot(2,1,2)
plt.hist(x_train[0].reshape(784))
plt.title("Pixel Values")
plt.show()
```

- Step 4. 準備訓練資料

```
img_size_x, img_size_y = 28, 28
x_train = x_train.reshape(x_train.shape[0], img_size_x, img_size_y, 1)
x_test = x_test.reshape(x_test.shape[0], img_size_x, img_size_y, 1)
input_shape = (img_size_x, img_size_y, 1)
x_train = x_train.astype("float32")
x_test = x_test.astype("float32")
x_train /= 255
x_test /= 255
```

# 以 CNN 實作 - KERAS

- Step 5. 轉換為 One hot encoding

```
y_train = np_utils.to_categorical(y_train,nb_classes)
y_test = np_utils.to_categorical(y_test,nb_classes)
```

- Step 6. 定義類神經網路模型

Sequential可以讓我們按照順序將神經網路路串串起。深度學習為隱藏層有兩兩層或兩兩層以上.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3), activation="relu", input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3,3), activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
```

Loss:
https://keras.io/losses/
Optimizer:
https://keras.io/optimizers/

- Step 7. Compile

```
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

# 以 CNN 實作 - KERAS

- Step 8. 訓練模型

```
history = model.fit(x_train, y_train, batch_size=128, epochs=10, verbose=2, validation_data=(x_test, y_test))
```

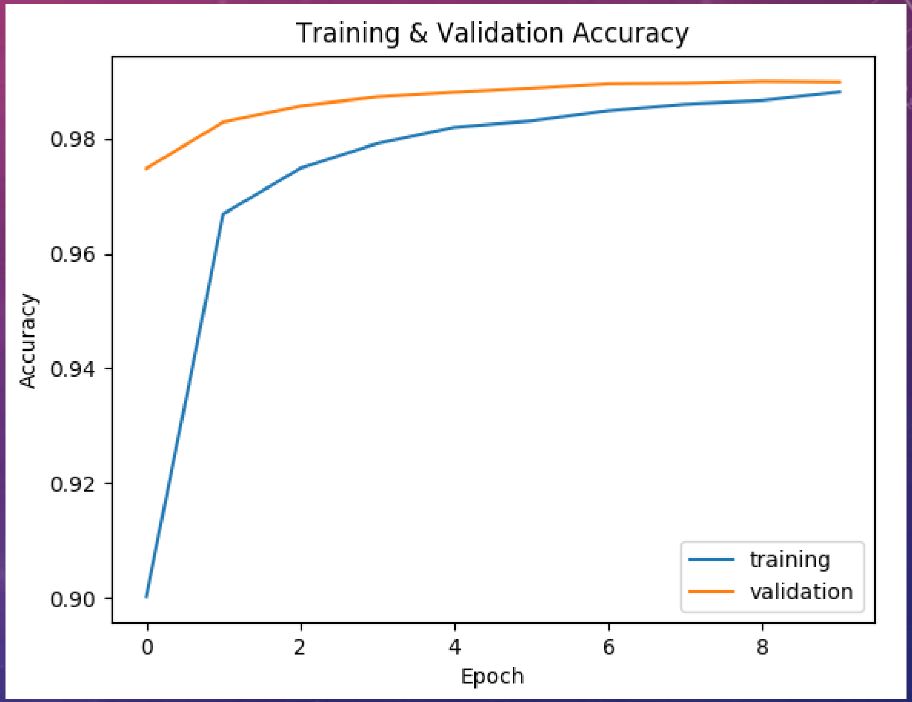
```
Epoch 1/10
125s - loss: 0.3322 - acc: 0.9002 - val_loss: 0.0789 - val_acc: 0.9748
Epoch 2/10
121s - loss: 0.1125 - acc: 0.9669 - val_loss: 0.0519 - val_acc: 0.9829
Epoch 3/10
123s - loss: 0.0844 - acc: 0.9748 - val_loss: 0.0424 - val_acc: 0.9857
Epoch 4/10
127s - loss: 0.0714 - acc: 0.9792 - val_loss: 0.0378 - val_acc: 0.9873
Epoch 5/10
124s - loss: 0.0617 - acc: 0.9820 - val_loss: 0.0364 - val_acc: 0.9881
Epoch 6/10
123s - loss: 0.0570 - acc: 0.9831 - val_loss: 0.0308 - val_acc: 0.9888
Epoch 7/10
124s - loss: 0.0506 - acc: 0.9849 - val_loss: 0.0294 - val_acc: 0.9896
Epoch 8/10
125s - loss: 0.0466 - acc: 0.9860 - val_loss: 0.0291 - val_acc: 0.9897
Epoch 9/10
124s - loss: 0.0441 - acc: 0.9867 - val_loss: 0.0286 - val_acc: 0.9900
Epoch 10/10
123s - loss: 0.0396 - acc: 0.9881 - val_loss: 0.0300 - val_acc: 0.9899
```

From 98% to 99%

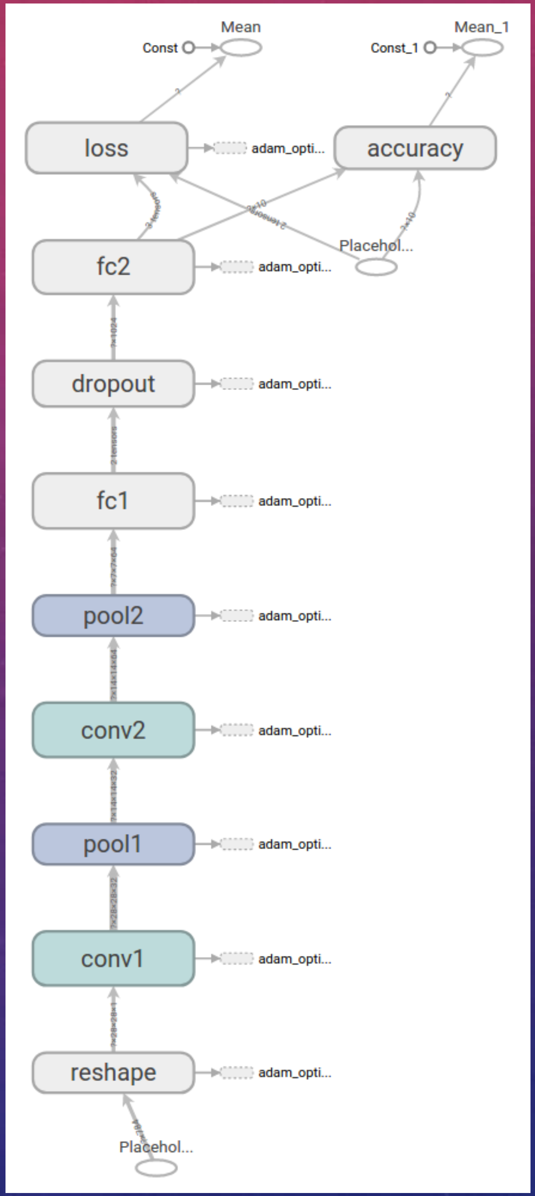
# 以 CNN 實作 - KERAS

- Step 9. 檢查準確度

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training data')
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.legend(['training','validation'],loc='lower right')
plt.show()
```



# 以 CNN 實作 – TENSORFLOW



# 以 CNN 實作 - TENSORFLOW

- Step 1. 準備副程式

Create tensor of shape, and the weights are normal-distribution. the input is the kernel filter size [height, width , channel, number]

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

strides stands for the moving of the kernel filter. [batch, height, width, channel]. padding='SAME' means the output shape = input shape

# 以 CNN 實作 - TENSORFLOW

- Step 2. 載入數據並準備 PlaceHolder

```python
# Load MNIST Data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# Start TensorFlow InteractiveSession
sess = tf.InteractiveSession()

x = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 10])
```

# 以 CNN 實作 - TENSORFLOW

- Step 3. 建立 Computation Graph

the input is the kernel filter size [height, width , channel, number]

```
# First Convolutional Layer
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

x_image = tf.reshape(x, [-1, 28, 28, 1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)


# Second Convolutional Layer
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```

```
# Densely Connected Layer
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Readout Layer
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

# 以 CNN 實作 - TENSORFLOW

- Step 4. 開始訓練並測試準確度

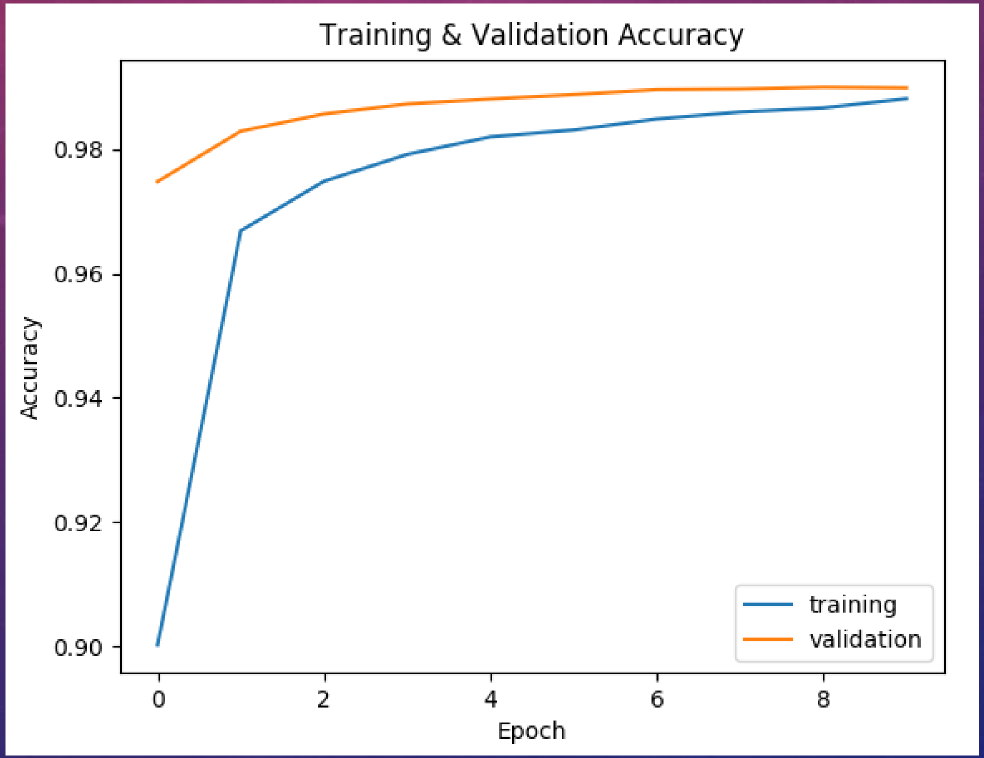
```
# Train and Evaluate the Model
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(20000):
        batch = mnist.train.next_batch(50)
        if i % 100 == 0:
            train_accuracy = accuracy.eval(feed_dict={x: batch[0], y_: batch[1], keep_prob: 1.0})
            print('step %d, training accuracy %g' % (i, train_accuracy))
        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

    print('test accuracy %g' % accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

tf.argmax:
when axis=0, it returns the max value row index each column
when axis=1, it returns the max value column index each row

tf.reduce_man: 計算平均值
# 'x' is [[1., 1.]
#         [2., 2.]]
tf.reduce_mean(x) ==> 1.5
tf.reduce_mean(x, 0) ==> [1.5, 1.5]
tf.reduce_mean(x, 1) ==> [1., 2.]

# CNN模型準確度

CNN (2 Layers & Dropout)

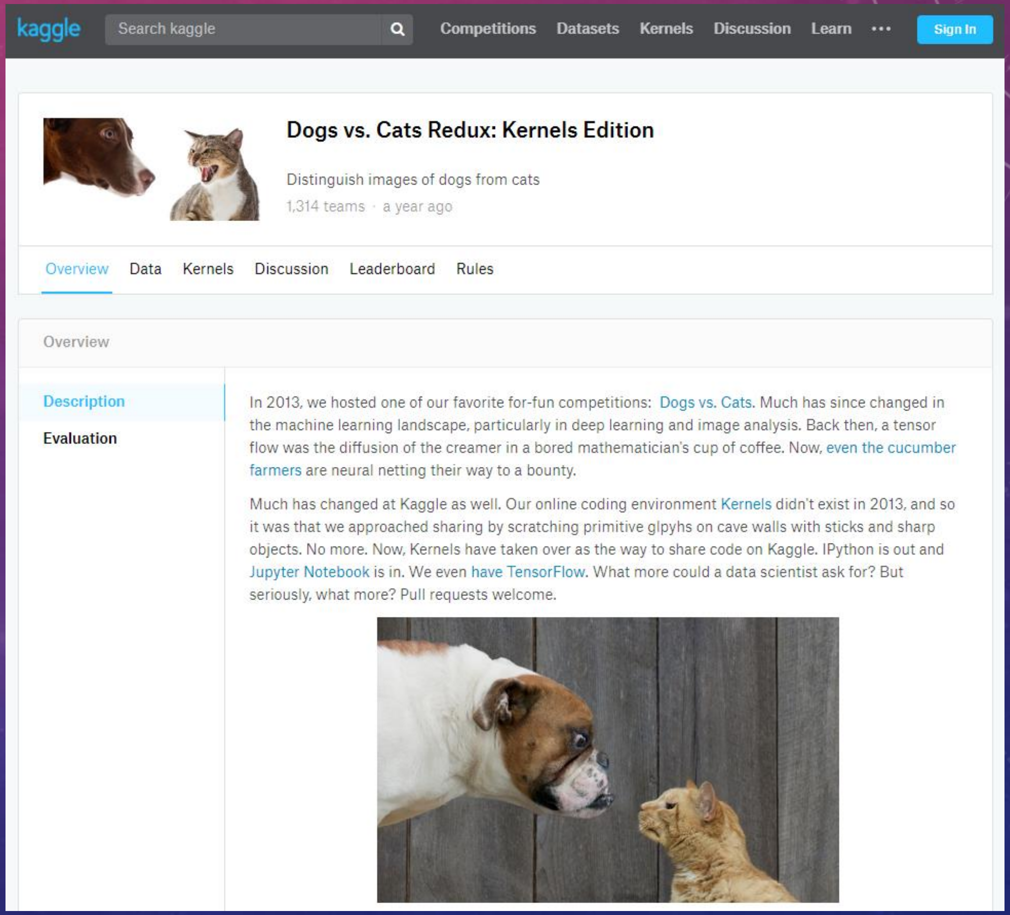


# ADVANCED EXERCISE

## DOG & CAT

# DOGS VS. CATS

- https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition
  - Training data: 25000 images
  - Test data: 12500 images
  - For each image in the test set, you should predict a probability that the image is a dog (1 = dog, 0 = cat).



# THANK YOU