In [2]:
```
from google.colab import files
files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving HumanActivityRecognition.zip to HumanActivityRecognition.zip
```

In [0]:
```
import zipfile
zip_ref = zipfile.ZipFile("HumanActivityRecognition.zip", 'r')
zip_ref.extractall("Human")
zip_ref.close()
```

In [4]:
```
# Importing the required libraries

import pandas as pd
import numpy as np
from keras import backend as K
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
from keras.optimizers import RMSprop, SGD, Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
```

```
Using TensorFlow backend.
```

In [0]:
```
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}


# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]
```

In [0]:
```
# Defining Helper Functions

# Utility function to print the confusion matrix
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])


# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the signals
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'Human/HAR/UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))


def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = f'Human/HAR/UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()
```

```
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test


 # Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [0]:
```
X_train, X_test, Y_train, Y_test = load_data() # Loading the train and test data
```

In [8]:
```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

Model - 1 With One LSTM Layer with 32 Neurons

In [9]:
```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(32, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding a dense output layer with softmax activation
model.add(Dense(n_classes, activation='softmax'))
model.summary()

# Compiling the model
model.compile(loss='categorical_crossentropy',optimizer=RMSprop(lr = 0.001),metrics=['accuracy'])

# Declaring Callbacks

checkpoint = ModelCheckpoint("model_1.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)

earlystop = EarlyStopping(monitor = 'val_loss',
                          min_delta = 0,
                          patience = 3,
                          verbose = 1,
                          restore_best_weights = True)

reduce_lr = ReduceLROnPlateau(monitor = 'val_loss',
                              factor = 0.2,
                              patience = 3,
                              verbose = 1,
                              min_delta = 0.0001)

# we put our call backs into a callback list
callbacks = [earlystop, checkpoint, reduce_lr]



# Training the model
model_1 = model.fit(X_train, Y_train, batch_size=64,validation_data=(X_test, Y_test),epochs=50,callbacks = callbacks)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                5376
```

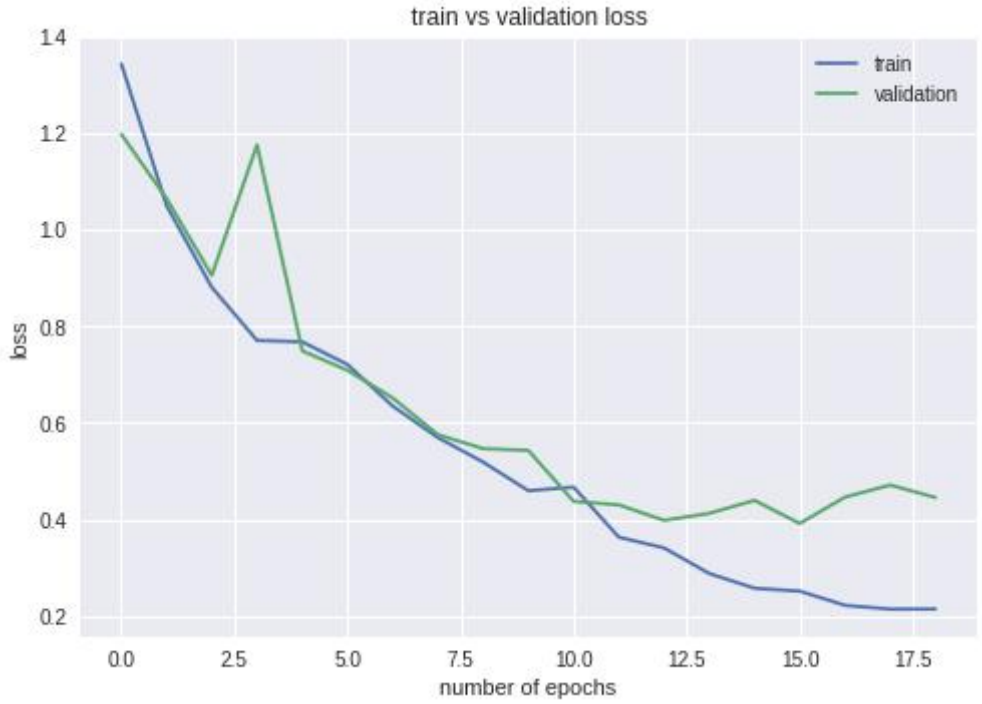
```
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 198
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
Train on 7352 samples, validate on 2947 samples
Epoch 1/50
7352/7352 [==============================] - 36s 5ms/step - loss: 1.3435 - acc: 0.4396 - val_loss: 1.1980 - val_acc:
0.4625

Epoch 00001: val_loss improved from inf to 1.19797, saving model to model_1.h5
Epoch 2/50
7352/7352 [==============================] - 34s 5ms/step - loss: 1.0503 - acc: 0.5522 - val_loss: 1.0647 - val_acc:
0.5453

Epoch 00002: val_loss improved from 1.19797 to 1.06472, saving model to model_1.h5
Epoch 3/50
7352/7352 [==============================] - 33s 5ms/step - loss: 0.8810 - acc: 0.6166 - val_loss: 0.9059 - val_acc:
0.5952

Epoch 00003: val_loss improved from 1.06472 to 0.90590, saving model to model_1.h5
Epoch 4/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.7710 - acc: 0.6442 - val_loss: 1.1759 - val_acc:
0.5531

Epoch 00004: val_loss did not improve from 0.90590
Epoch 5/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.7681 - acc: 0.6533 - val_loss: 0.7495 - val_acc:
0.6895

Epoch 00005: val_loss improved from 0.90590 to 0.74948, saving model to model_1.h5
Epoch 6/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.7212 - acc: 0.6738 - val_loss: 0.7091 - val_acc:
0.7116

Epoch 00006: val_loss improved from 0.74948 to 0.70915, saving model to model_1.h5
Epoch 7/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.6351 - acc: 0.7195 - val_loss: 0.6520 - val_acc:
```

```
0.7333

Epoch 00007: val_loss improved from 0.70915 to 0.65201, saving model to model_1.h5
Epoch 8/50
7352/7352 [==============================] - 33s 5ms/step - loss: 0.5698 - acc: 0.7715 - val_loss: 0.5758 - val_acc:
0.7879

Epoch 00008: val_loss improved from 0.65201 to 0.57576, saving model to model_1.h5
Epoch 9/50
7352/7352 [==============================] - 33s 5ms/step - loss: 0.5191 - acc: 0.7945 - val_loss: 0.5469 - val_acc:
0.8025

Epoch 00009: val_loss improved from 0.57576 to 0.54690, saving model to model_1.h5
Epoch 10/50
7352/7352 [==============================] - 33s 5ms/step - loss: 0.4598 - acc: 0.8229 - val_loss: 0.5432 - val_acc:
0.8066

Epoch 00010: val_loss improved from 0.54690 to 0.54321, saving model to model_1.h5
Epoch 11/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.4673 - acc: 0.8335 - val_loss: 0.4377 - val_acc:
0.8395

Epoch 00011: val_loss improved from 0.54321 to 0.43771, saving model to model_1.h5
Epoch 12/50
7352/7352 [==============================] - 33s 5ms/step - loss: 0.3636 - acc: 0.8776 - val_loss: 0.4305 - val_acc:
0.8616

Epoch 00012: val_loss improved from 0.43771 to 0.43050, saving model to model_1.h5
Epoch 13/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.3414 - acc: 0.8993 - val_loss: 0.3984 - val_acc:
0.8622

Epoch 00013: val_loss improved from 0.43050 to 0.39844, saving model to model_1.h5
Epoch 14/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.2883 - acc: 0.9104 - val_loss: 0.4130 - val_acc:
0.8663

Epoch 00014: val_loss did not improve from 0.39844
Epoch 15/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.2578 - acc: 0.9174 - val_loss: 0.4398 - val_acc:
0.8599
```

```
Epoch 00015: val_loss did not improve from 0.39844
Epoch 16/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.2518 - acc: 0.9206 - val_loss: 0.3920 - val_acc:
0.8673

Epoch 00016: val_loss improved from 0.39844 to 0.39199, saving model to model_1.h5
Epoch 17/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.2224 - acc: 0.9260 - val_loss: 0.4466 - val_acc:
0.8680

Epoch 00017: val_loss did not improve from 0.39199
Epoch 18/50
7352/7352 [==============================] - 33s 5ms/step - loss: 0.2146 - acc: 0.9268 - val_loss: 0.4716 - val_acc:
0.8629

Epoch 00018: val_loss did not improve from 0.39199
Epoch 19/50
7352/7352 [==============================] - 34s 5ms/step - loss: 0.2148 - acc: 0.9339 - val_loss: 0.4458 - val_acc:
0.8782
Restoring model weights from the end of the best epoch

Epoch 00019: val_loss did not improve from 0.39199

Epoch 00019: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 00019: early stopping
```

In [21]:
```
import matplotlib.pyplot as plt

scores = model.evaluate(X_test, Y_test, verbose=0)
print("Test Score: %f" % (scores[0]))
print("Test Accuracy: %f%%" % (scores[1]*100))


# plot train and validation loss

plt.plot(model_1.history['loss'])
plt.plot(model_1.history['val_loss'])
plt.title('train vs validation loss')
plt.ylabel('loss')
plt.xlabel('number of epochs')
plt.legend(['train','validation'],loc='upper right')
plt.show()
```

```
Test Score: 0.290512
Test Accuracy: 89.752290%
```



In [18]:
```
# Predictions

Y_predictions = model.predict(X_test)



# Confusion Matrix

model_1_cm = confusion_matrix(Y_test,Y_predictions)
model_1_cm
```

Out[18]:

Pred / True	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
LAYING	510	0	0	0	0	27
SITTING	0	374	109	2	0	6
STANDING	0	86	439	2	2	3
WALKING	1	0	0	446	42	7
WALKING_DOWNSTAIRS	0	0	0	13	373	34
WALKING_UPSTAIRS	0	0	1	18	38	414

In [0]:

In [0]:

Model -2 With Two LSTM Layers and 64 Neurons

In [26]:
```
# Initiliazing the sequential model
model_2 = Sequential()
# Configuring the parameters
model_2.add(LSTM(64,return_sequences=True, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model_2.add(Dropout(0.7))

# Configuring the parameters
model_2.add(LSTM(64))
# Adding a dropout layer
model_2.add(Dropout(0.7))
# Adding a dense output layer with softmax activation
model_2.add(Dense(n_classes, activation='softmax'))
model_2.summary()

# Compiling the model
model_2.compile(loss='categorical_crossentropy',optimizer= Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.

# Declaring Callbacks

checkpoint = ModelCheckpoint("model_2.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)

earlystop = EarlyStopping(monitor = 'val_loss',
                          min_delta = 0,
                          patience = 3,
                          verbose = 1,
                          restore_best_weights = True)

reduce_lr = ReduceLROnPlateau(monitor = 'val_loss',
                              factor = 0.2,
                              patience = 3,
                              verbose = 1,
                              min_delta = 0.0001)

# we put our call backs into a callback list
callbacks = [earlystop, checkpoint, reduce_lr]

# Training the model
model_two = model_2.fit(X_train,Y_train,batch_size=64,validation_data=(X_test, Y_test),epochs=50,callbacks=callbacks)
```

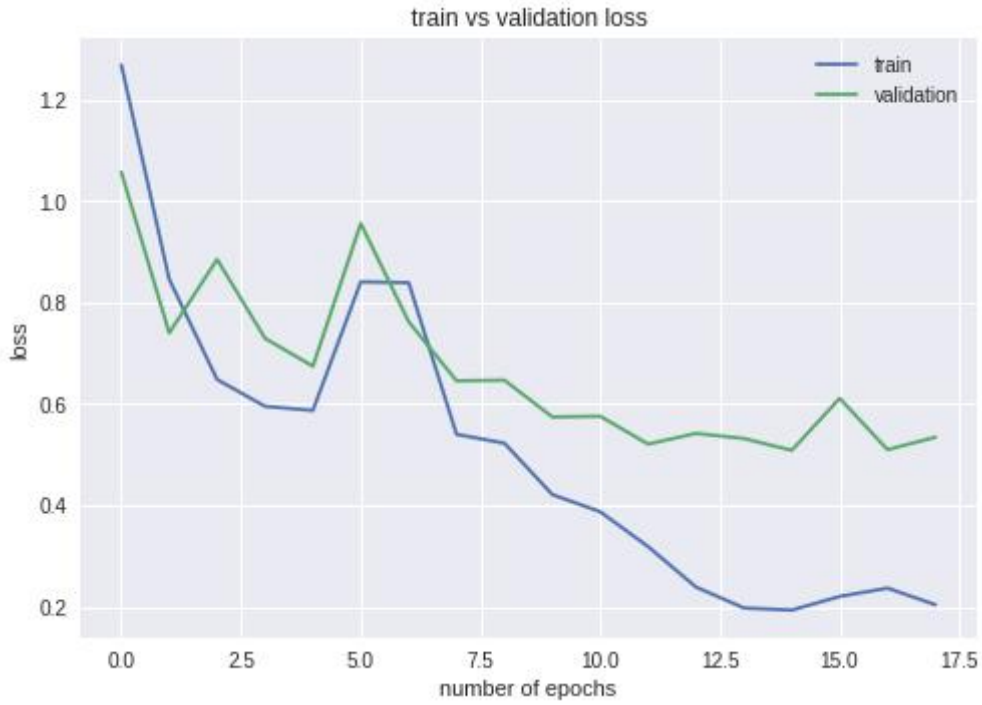
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 128, 64)           18944
_________________________________________________________________
dropout_6 (Dropout)          (None, 128, 64)           0
_________________________________________________________________
lstm_7 (LSTM)                (None, 64)                33024
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 390
=================================================================
Total params: 52,358
Trainable params: 52,358
Non-trainable params: 0
_________________________________________________________________
Train on 7352 samples, validate on 2947 samples
Epoch 1/50
7352/7352 [==============================] - 69s 9ms/step - loss: 1.2688 - acc: 0.4585 - val_loss: 1.0573 - val_acc:
0.5558

Epoch 00001: val_loss improved from inf to 1.05732, saving model to model_2.h5
Epoch 2/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.8460 - acc: 0.6424 - val_loss: 0.7402 - val_acc:
0.7072

Epoch 00002: val_loss improved from 1.05732 to 0.74018, saving model to model_2.h5
Epoch 3/50
7352/7352 [==============================] - 67s 9ms/step - loss: 0.6483 - acc: 0.7259 - val_loss: 0.8853 - val_acc:
0.6271

Epoch 00003: val_loss did not improve from 0.74018
Epoch 4/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.5955 - acc: 0.7326 - val_loss: 0.7296 - val_acc:
0.6709

Epoch 00004: val_loss improved from 0.74018 to 0.72959, saving model to model_2.h5
Epoch 5/50
7352/7352 [==============================] - 67s 9ms/step - loss: 0.5873 - acc: 0.7422 - val_loss: 0.6746 - val_acc:
0.7306
```

```
Epoch 00005: val_loss improved from 0.72959 to 0.67457, saving model to model_2.h5
Epoch 6/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.8410 - acc: 0.7069 - val_loss: 0.9561 - val_acc:
0.6081

Epoch 00006: val_loss did not improve from 0.67457
Epoch 7/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.8394 - acc: 0.6593 - val_loss: 0.7624 - val_acc:
0.7374

Epoch 00007: val_loss did not improve from 0.67457
Epoch 8/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.5402 - acc: 0.7903 - val_loss: 0.6457 - val_acc:
0.7923

Epoch 00008: val_loss improved from 0.67457 to 0.64573, saving model to model_2.h5
Epoch 9/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.5227 - acc: 0.7847 - val_loss: 0.6471 - val_acc:
0.7723

Epoch 00009: val_loss did not improve from 0.64573
Epoch 10/50
7352/7352 [==============================] - 67s 9ms/step - loss: 0.4213 - acc: 0.8403 - val_loss: 0.5745 - val_acc:
0.8286

Epoch 00010: val_loss improved from 0.64573 to 0.57449, saving model to model_2.h5
Epoch 11/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.3875 - acc: 0.8901 - val_loss: 0.5759 - val_acc:
0.8246

Epoch 00011: val_loss did not improve from 0.57449
Epoch 12/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.3186 - acc: 0.9042 - val_loss: 0.5209 - val_acc:
0.8592

Epoch 00012: val_loss improved from 0.57449 to 0.52088, saving model to model_2.h5
Epoch 13/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.2389 - acc: 0.9256 - val_loss: 0.5421 - val_acc:
0.8517

Epoch 00013: val_loss did not improve from 0.52088
Epoch 14/50
```

```
7352/7352 [==============================] - 66s 9ms/step - loss: 0.1976 - acc: 0.9357 - val_loss: 0.5319 - val_acc:
0.8690

Epoch 00014: val_loss did not improve from 0.52088
Epoch 15/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.1934 - acc: 0.9362 - val_loss: 0.5086 - val_acc:
0.8734

Epoch 00015: val_loss improved from 0.52088 to 0.50856, saving model to model_2.h5
Epoch 16/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.2203 - acc: 0.9168 - val_loss: 0.6111 - val_acc:
0.7957

Epoch 00016: val_loss did not improve from 0.50856
Epoch 17/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.2368 - acc: 0.9125 - val_loss: 0.5100 - val_acc:
0.8741

Epoch 00017: val_loss did not improve from 0.50856
Epoch 18/50
7352/7352 [==============================] - 66s 9ms/step - loss: 0.2039 - acc: 0.9278 - val_loss: 0.5345 - val_acc:
0.8690
Restoring model weights from the end of the best epoch

Epoch 00018: val_loss did not improve from 0.50856

Epoch 00018: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 00018: early stopping
```

In [27]:
```
scores_2 = model_2.evaluate(X_test, Y_test, verbose=0)
print("Test Score: %f" % (scores_2[0]))
print("Test Accuracy: %f%%" % (scores_2[1]*100))


# plot train and validation loss

plt.plot(model_two.history['loss'])
plt.plot(model_two.history['val_loss'])
plt.title('train vs validation loss')
plt.ylabel('loss')
plt.xlabel('number of epochs')
plt.legend(['train','validation'],loc='upper right')
plt.show()
```

```
Test Score: 0.508556
Test Accuracy: 87.343061%
```



In [28]:
```
# Predictions

Y_predictions = model_2.predict(X_test)



# Confusion Matrix

model_2_cm = confusion_matrix(Y_test,Y_predictions)
model_2_cm
```

Out[28]:

Pred / True	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
LAYING	510	0	0	0	0	27
SITTING	0	371	95	3	0	22
STANDING	0	74	442	12	0	4
WALKING	0	0	0	451	2	43
WALKING_DOWNSTAIRS	0	0	0	26	394	0
WALKING_UPSTAIRS	0	0	0	60	5	406

In [0]:
```

```

Comparing Model Performance

In [29]:
```
from prettytable import PrettyTable


name = ["Model 1","Model 2"]
train_acc = [max(model_1.history['acc']),max(model_two.history['acc'])]

test_acc =[scores[1],scores_2[1]]

numbering = [1,2]
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",name)
ptable.add_column("Training Accuracy",train_acc)
ptable.add_column("Test Accuracy",test_acc)
print(ptable)
```

```
+-------+---------+-------------------+--------------------+
| S.NO. |  MODEL  | Training Accuracy |   Test Accuracy    |
+-------+---------+-------------------+--------------------+
|   1   | Model 1 | 0.933895538823519 | 0.8975229046487954 |
|   2   | Model 2 | 0.936207834537971 | 0.8734306073973532 |
+-------+---------+-------------------+--------------------+
```

In [0]: