# Project 2- ECE 188, Spring 2022 - Deadline: June 10th, 2022. Please upload a link to your Github on Gradescope.

## Train a Deep Learning model on the MNIST dataset and evaluate it on a test set.

The first part in this notebook will take you though the training of the MNIST Dataset which is a collection of handwritten digits.

We will build a Deep Learning based system that can recognise handwritten digits using Keras and Tensorflow.

Note that the entire project will be done on a Google Colab. A tutorial to some basic colab instructions can be found here.

Note that all the code needed is provided until Task 1, **you will only have to write code after the task 1 header**

## Import Necessary Packages

We import Keras and Tensorflow to train our network and numpy to work with our data.

We also import matplotlib to view our images.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import layers
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.rcParams['figure.figsize'] = (8, 8)
mpl.rcParams['axes.grid'] = False
```

✓ 0s completed at 2:27 PM

## Load the Data

We perform some data preprocessing. Feel free to skip over this and use the train and test data as x_train and x_test

```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")


# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

## Define a model in keras.

We define a simple simple network with 2 hidden layers. One fully connected dense layer with 8 neurons and onr flatten layer that flattens the model into 1D structure which is then fed into a softmax output.

This is a barebones keras structure. In task 1 you will be tasked with improving this model.

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Dense(8,activation="relu"),
        layers.Flatten(),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 28, 28, 8)         16        
                                                                 
 flatten (Flatten)           (None, 6272)              0         
                                                                 
 dense_1 (Dense)             (None, 10)                62730     
                                                                 
=================================================================
Total params: 62,746
Trainable params: 62,746
Non-trainable params: 0
_________________________________________________________________
```

## Compile and train the model

We know train our model. Before we do that we need to set some parameters to train the model. Hence we define the loss and optimizer to be used as well as batch size. The number of epochs refers to the number of training iterations.

```
batch_size = 128
epochs = 15

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

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
    Epoch 1/15
    422/422 [==============================] - 5s 4ms/step - loss: 0.5369 - accuracy: 0.8628 - val_loss: 0.2629 - val_a
    Epoch 2/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.3058 - accuracy: 0.9128 - val_loss: 0.2388 - val_a
    Epoch 3/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2854 - accuracy: 0.9197 - val_loss: 0.2298 - val_a
    Epoch 4/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2752 - accuracy: 0.9226 - val_loss: 0.2314 - val_a
    Epoch 5/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2700 - accuracy: 0.9241 - val_loss: 0.2269 - val_a
    Epoch 6/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2652 - accuracy: 0.9259 - val_loss: 0.2250 - val_a
    Epoch 7/15
    422/422 [==============================] - 2s 5ms/step - loss: 0.2626 - accuracy: 0.9264 - val_loss: 0.2273 - val_a
    Epoch 8/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2588 - accuracy: 0.9281 - val_loss: 0.2261 - val_a
    Epoch 9/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2565 - accuracy: 0.9285 - val_loss: 0.2245 - val_a
    Epoch 10/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2551 - accuracy: 0.9290 - val_loss: 0.2289 - val_a
    Epoch 11/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2534 - accuracy: 0.9293 - val_loss: 0.2269 - val_a
    Epoch 12/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2514 - accuracy: 0.9294 - val_loss: 0.2290 - val_a
    Epoch 13/15
    422/422 [==============================] - 2s 4ms/step - loss: 0.2505 - accuracy: 0.9301 - val_loss: 0.2262 - val_a
    Epoch 14/15
```

```
422/422 [==============================] - 2s 4ms/step - loss: 0.2491 - accuracy: 0.9307 - val_loss: 0.2317 - val_a
Epoch 15/15
422/422 [==============================] - 2s 4ms/step - loss: 0.2476 - accuracy: 0.9311 - val_loss: 0.2281 - val_a
<keras.callbacks.History at 0x7f8960279290>
```

## Evaluate the model

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.2704991102218628
Test accuracy: 0.9264000058174133
```

We achieve an accuracy on the test set of 92% but can we do better?

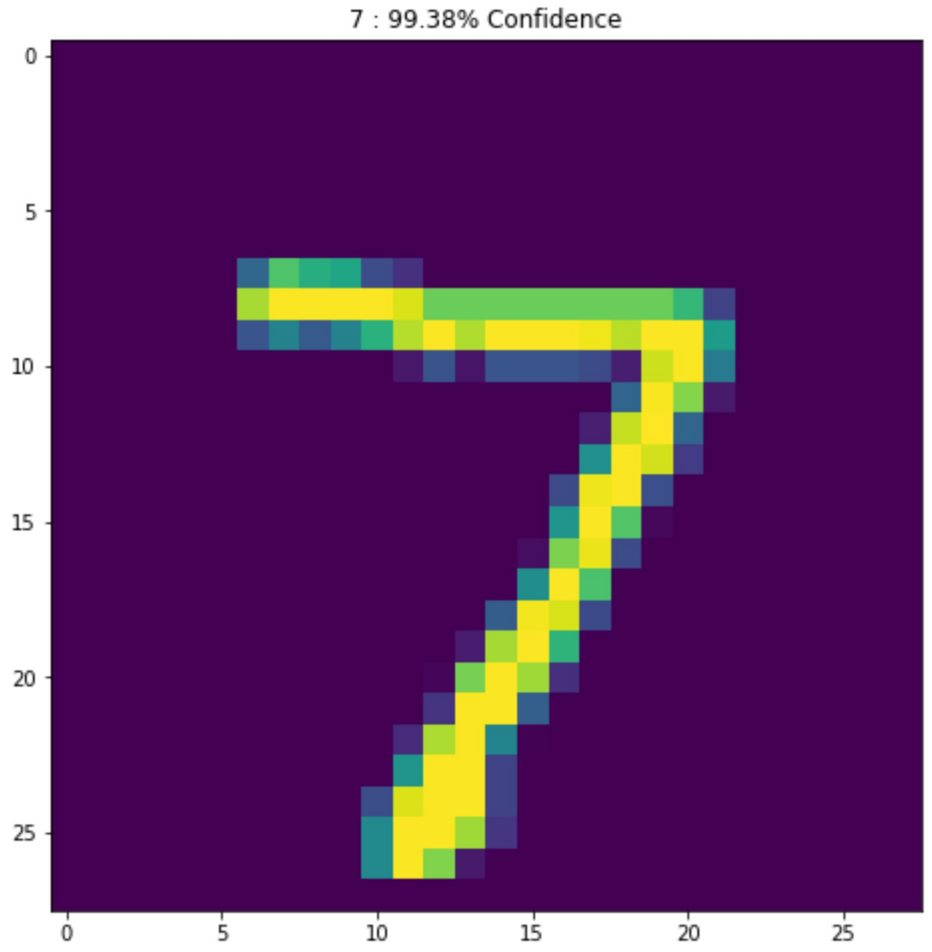
## Looking at a prediction.

Lets look at the prediction for a particular image in our model. We will look at one corect and one incorrect prediction and see if we can improve on the incorrect prediction.

```
def get_mnist_label(image_probs):

  return np.argmax(image_probs), np.max(image_probs)
```

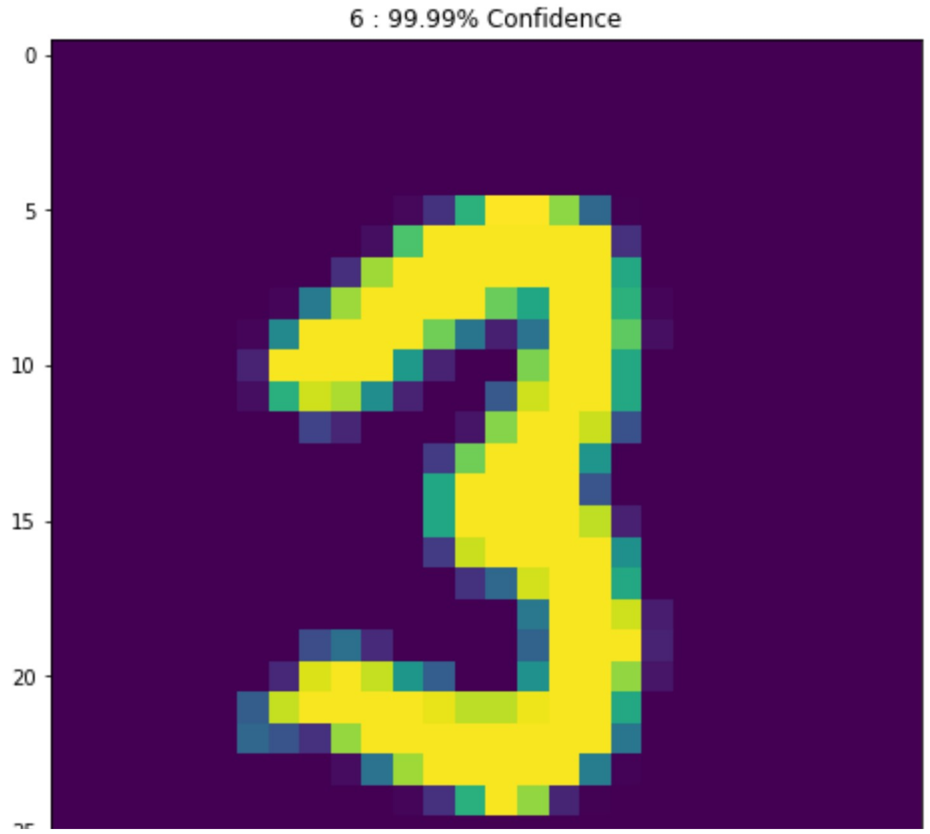
```
preds = model.predict(x_test)

image = x_test[0].reshape(input_shape[0], input_shape[1])
label = np.argmax(y_test[0])
image_probs = preds[0]
```

```
plt.figure()
plt.imshow(image*0.5 + 0.5)  # To change [-1, 1] to [0,1]
image_class, class_confidence = get_mnist_label(image_probs)
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()
```



```
# Store the incorrect prediction index here
incorrect_index = 0
```

```
for i in range(preds.shape[0]):

  if np.argmax(preds[i]) != np.argmax(y_test[i]):

    incorrect_index = i

image_incorrect = x_test[incorrect_index].reshape(input_shape[0], input_shape[1])


plt.figure()
plt.imshow(image_incorrect*0.5 + 0.5)  # To change [-1, 1] to [0,1]
image_class, class_confidence = get_mnist_label(preds[i])
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()
```



We see that a 3 has been predicted with high confidence as a 6 does this change with our improved model below?

## TASK 1: Improve the accuracy of the model

Your goal in task one is to improve the accuracy of the model on the test set.

For image recogniton tasks convolutional neural networks work better than most other type of architectures. Your goal is to push the test accuracy above 99% percent.

The following steps will get you there.

1. Convert the 1st hidden Dense layer to a conv layer with 32 filters, a filter size of 3x3 and with relu activation. (I've done this step)
2. Add a Maxpooling layer of size 2x2.
3. Add another Conv layer with 64 filters, with a filter size of 3x3 with relu activation.
4. Add a Maxpooling layer of size 2x2.
5. Add Dropout to the layer after the Flatten layer.

```
improved_model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        # Add Maxpooling layer here
        layers.MaxPool2D(
          pool_size=(2, 2),
          strides=2
        ),
        ,,
        # Add Conv2d layer here
        layers.Conv2D(
          64, #filters
          kernel_size=(3, 3),
          activation='relu'
          ),
        #Add another Maxpooling layer here
         layers.MaxPool2D(
          pool_size=(2, 2),
          strides=2
        ),

        layers.Flatten(),
        # Add dropout of 0.5 here
        layers.Dropout(0.5, input_shape=(2,)),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

improved_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 1600)              0

 dropout (Dropout)           (None, 1600)              0
 dense_2 (Dense)             (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

## Complile and train the improved model.

```
batch_size = 128
epochs = 15

improved_model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

improved_model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
Epoch 1/15
422/422 [==============================] - 12s 6ms/step - loss: 0.3736 - accuracy: 0.8865 - val_loss: 0.0877 - val_
Epoch 2/15
422/422 [==============================] - 2s 5ms/step - loss: 0.1162 - accuracy: 0.9648 - val_loss: 0.0623 - val_a
Epoch 3/15
422/422 [==============================] - 2s 5ms/step - loss: 0.0865 - accuracy: 0.9737 - val_loss: 0.0499 - val_a
Epoch 4/15
422/422 [==============================] - 2s 5ms/step - loss: 0.0723 - accuracy: 0.9777 - val_loss: 0.0450 - val_a
Epoch 5/15
422/422 [==============================] - 2s 5ms/step - loss: 0.0626 - accuracy: 0.9805 - val_loss: 0.0405 - val_a
Epoch 6/15
422/422 [==============================] - 2s 5ms/step - loss: 0.0578 - accuracy: 0.9819 - val_loss: 0.0358 - val_a
Epoch 7/15
422/422 [==============================] - 2s 5ms/step - loss: 0.0519 - accuracy: 0.9839 - val_loss: 0.0385 - val_a
Epoch 8/15
422/422 [==============================] - 2s 5ms/step - loss: 0.0490 - accuracy: 0.9847 - val_loss: 0.0324 - val_a
Epoch 9/15
422/422 [==============================] - 2s 5ms/step - loss: 0.0452 - accuracy: 0.9856 - val_loss: 0.0318 - val_a
Epoch 10/15
```

```
    422/422 [==============================] - 2s 5ms/step - loss: 0.0416 - accuracy: 0.9868 - val_loss: 0.0299 - val_a
    Epoch 11/15
    422/422 [==============================] - 2s 5ms/step - loss: 0.0387 - accuracy: 0.9876 - val_loss: 0.0298 - val_a
    Epoch 12/15
    422/422 [==============================] - 2s 5ms/step - loss: 0.0384 - accuracy: 0.9876 - val_loss: 0.0304 - val_a
    Epoch 13/15
    422/422 [==============================] - 2s 5ms/step - loss: 0.0366 - accuracy: 0.9880 - val_loss: 0.0295 - val_a
    Epoch 14/15
    422/422 [==============================] - 2s 5ms/step - loss: 0.0325 - accuracy: 0.9894 - val_loss: 0.0296 - val_a
    Epoch 15/15
    422/422 [==============================] - 2s 5ms/step - loss: 0.0333 - accuracy: 0.9888 - val_loss: 0.0281 - val_a
    <keras.callbacks.History at 0x7f88ee005f90>
```

## Evaluate the Improved model.

You should achieve a test accuracy of greater than 99%!

If not the code will throw an assertion error.

```
score = improved_model.evaluate(x_test, y_test, verbose=0)
print("Improved Test loss:", score[0])
print("Improved Test accuracy:", score[1])

assert score[1] > 0.99
```

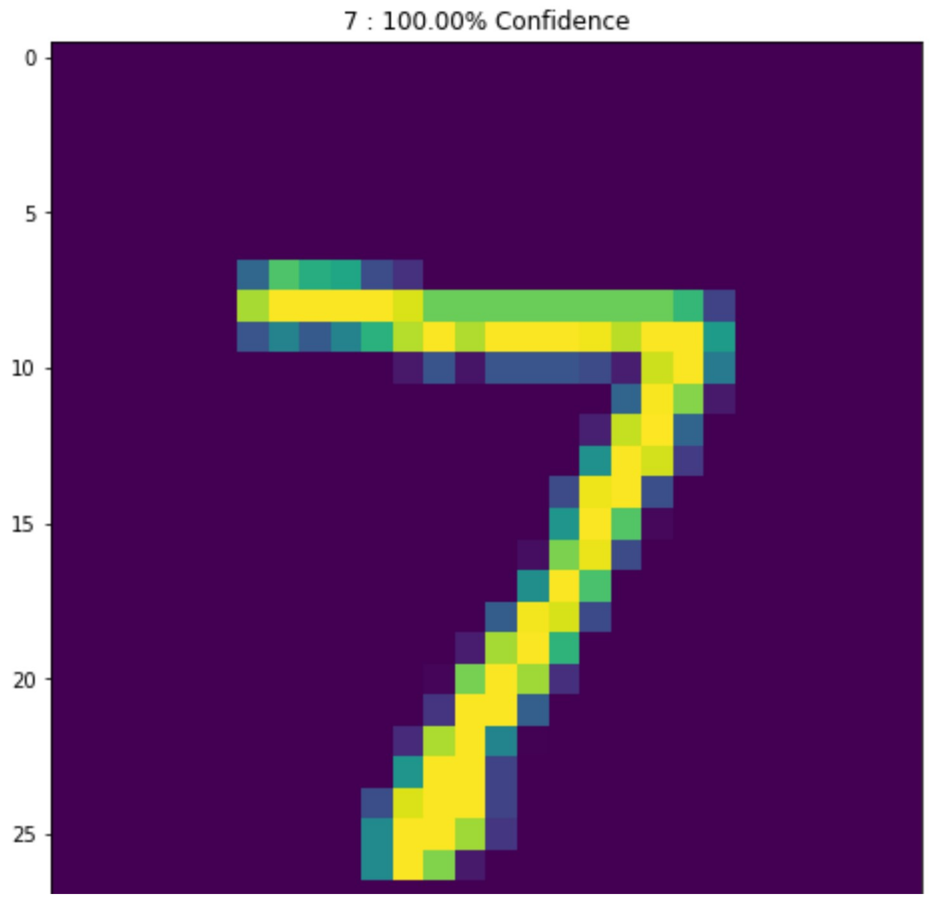
```
    Improved Test loss: 0.024961022660136223
    Improved Test accuracy: 0.991599977016449
```

## Looking at a prediction.

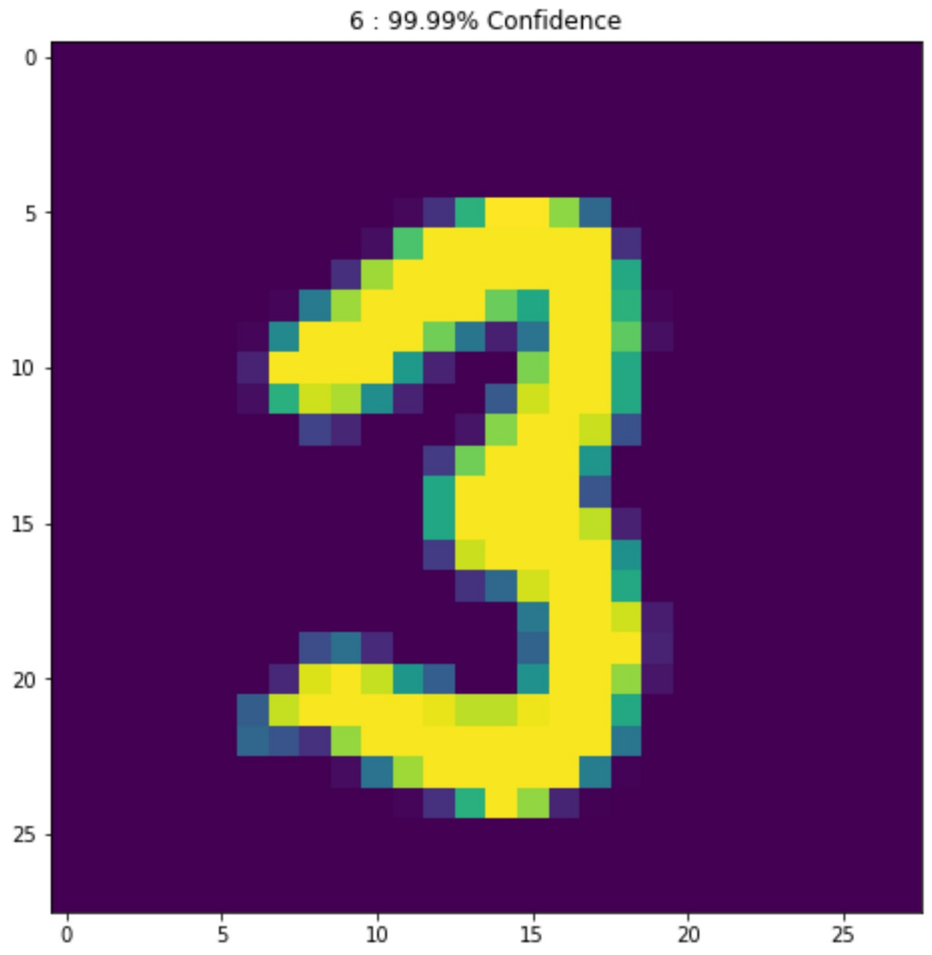
Lets look at the prediction for a particular image in our model. We will look at one corect and one incorrect prediction and see if we can improve on the incorrect prediction.

Check the predictions on the improved model, do you see improvements on the incorrect sample.

```
improved_preds = improved_model.predict(x_test)

image = x_test[0].reshape(input_shape[0], input_shape[1])
label = np.argmax(y_test[0])



plt.figure()
plt.imshow(image*0.5 + 0.5)  # To change [-1, 1] to [0,1]
image_class, class_confidence = get_mnist_label(improved_preds[0])
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()
```



```
plt.figure()
plt.imshow(image_incorrect*0.5 + 0.5)  # To change [-1, 1] to [0,1]
image_class, class_confidence = get_mnist_label(improved_preds[i])
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()
```


6 : 99.99% Confidence