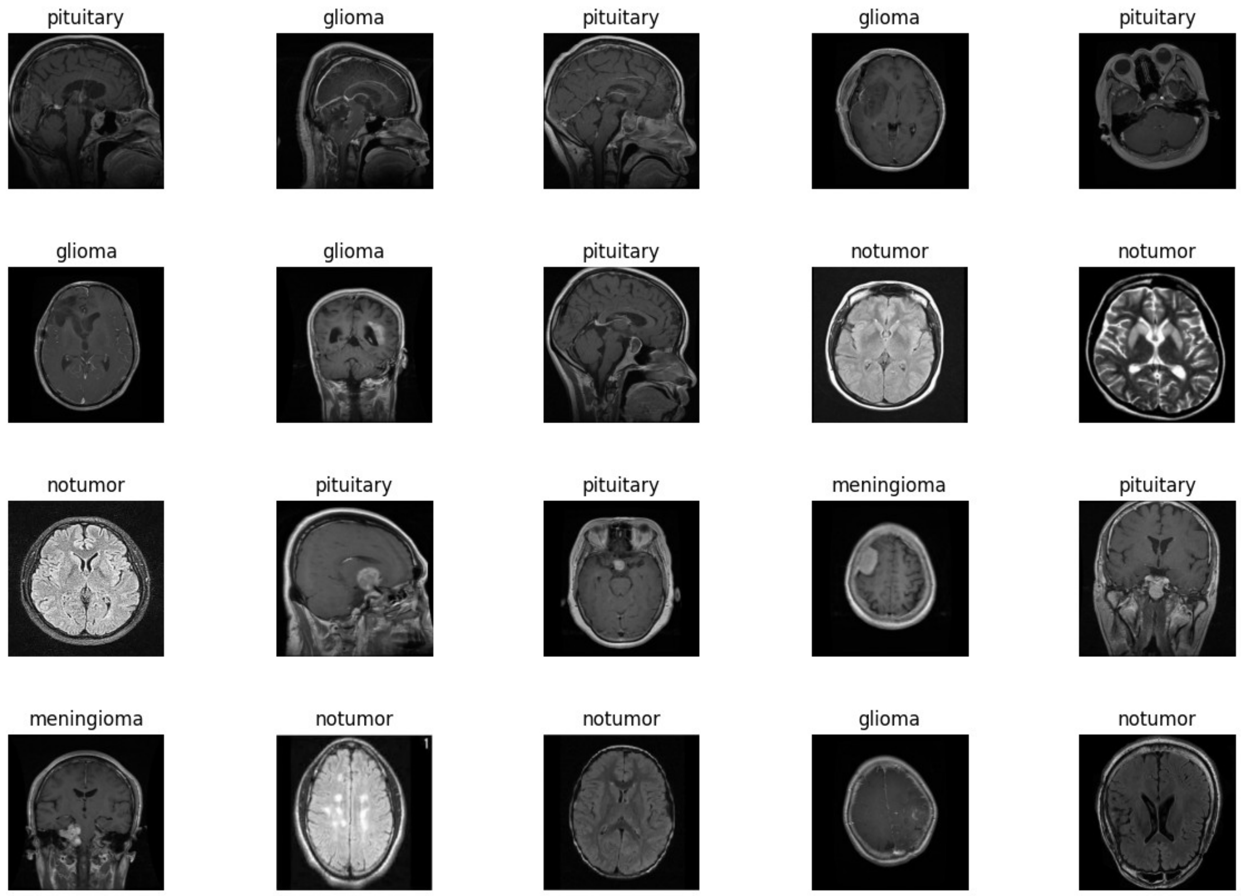
```python
import tensorflow as tf, matplotlib.pyplot as plt, numpy as np, cv2,
os, random

from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

train_dataset = 'dataset/training/'
test_dataset = 'dataset/testing/'
```

We now need to preprocess images within each folder and label them according to their classification.

```python
image_size_x, image_size_y = 224, 224

def preprocess_image(img_path):
    # read in the image
    if type(img_path) == np.ndarray:
        img = img_path
    else:
        img = cv2.imread(img_path)

    # convert image to grayscale
    # img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # normalize the image array
    # img = img / 255

    # resize the image for uniformity
    img = cv2.resize(img, (image_size_x, image_size_y))

    return img

encode = {'notumor': 0, 'meningioma': 1, 'glioma': 2, 'pituitary': 3}
decode = {0: 'notumor', 1: 'meningioma', 2: 'glioma', 3: 'pituitary'}

def load_data(dataset_path):
    x_data = [] # images
    y_data = [] # labels

    for folder in os.listdir(dataset_path):
        folder_path = os.path.join(dataset_path, folder)

        # check if the path is a directory and skip if not
        if not os.path.isdir(folder_path):
            continue

        for file in os.listdir(folder_path):
            # find path of the image
            file_path = os.path.join(folder_path, file)
            # preprocess the image
            img = preprocess_image(file_path)

            # append the new image to the x array
            x_data.append(img)

            # append the label to the y array
            y_data.append(encode[folder])

    x_data = np.array(x_data)
    y_data = np.array(y_data)

    # shuffle data for better training
    x_data, y_data = shuffle(x_data, y_data, random_state=101)

    return x_data, y_data

x_train, y_train = load_data(train_dataset)
x_test, y_test = load_data(test_dataset)

y_train = tf.keras.utils.to_categorical(y_train, num_classes=4)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=4)
```

Let us take a look at some images within our dataset.

```
fig, axes = plt.subplots(4, 5, figsize=(15, 10))
axes = axes.ravel()
for i in range(0, 20):
    index = random.randint(0, len(x_train))
    axes[i].imshow(x_train[index], cmap='gray')
    axes[i].set_title(decode[np.argmax(y_train[index])])
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.5)
plt.show()
```



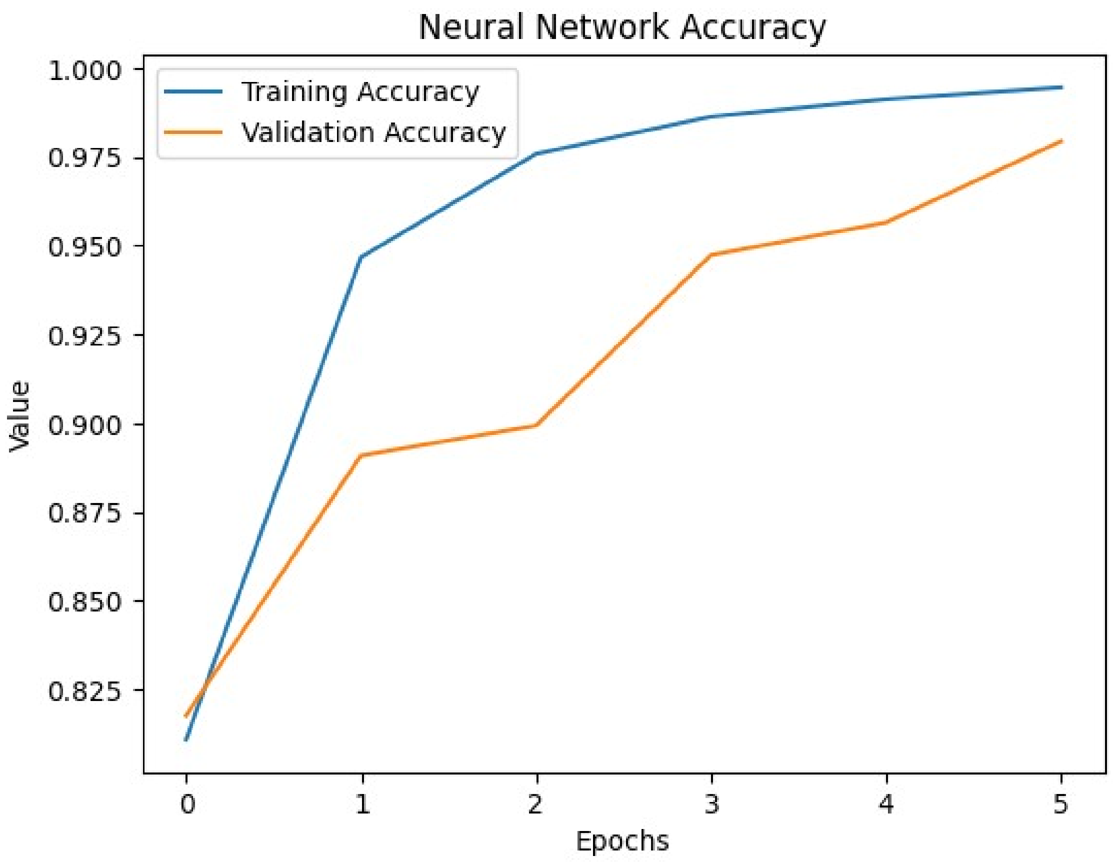
Now with our data we can create our model.

```
efficient_net =
tf.keras.applications.EfficientNetB0(weights='imagenet',
include_top=False, input_shape=(image_size_x, image_size_y, 3))

model = tf.keras.models.Sequential([
    efficient_net,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4, activation='softmax')
])

model.compile(optimizer=tf.optimizers.legacy.Adam(learning_rate=0.0001
), loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x_train, y_train, batch_size=64, epochs=6,
validation_data=(x_test, y_test))

Epoch 1/6
90/90 [==============================] - 546s 6s/step - loss: 0.5287 -
accuracy: 0.8109 - val_loss: 0.4223 - val_accuracy: 0.8177
Epoch 2/6
```

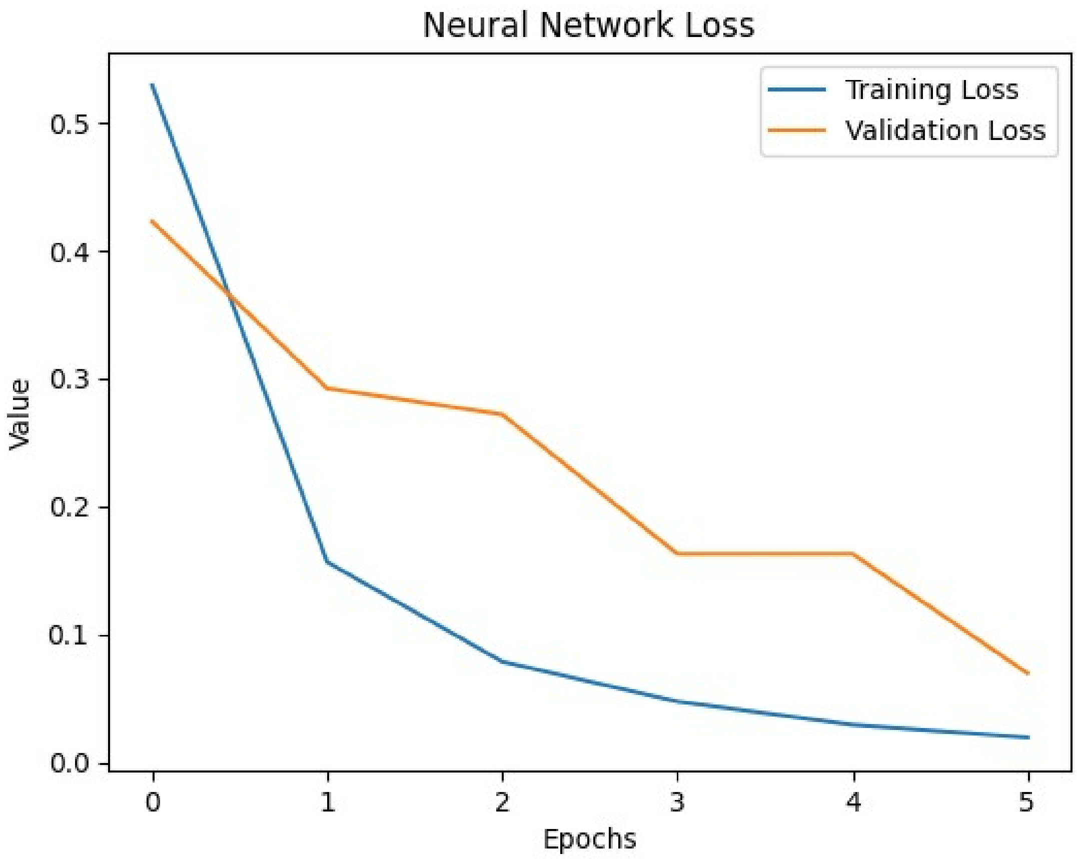
```
90/90 [==============================] - 528s 6s/step - loss: 0.1563 -
accuracy: 0.9468 - val_loss: 0.2920 - val_accuracy: 0.8909
Epoch 3/6
90/90 [==============================] - 522s 6s/step - loss: 0.0785 -
accuracy: 0.9758 - val_loss: 0.2717 - val_accuracy: 0.8993
Epoch 4/6
90/90 [==============================] - 676s 8s/step - loss: 0.0474 -
accuracy: 0.9863 - val_loss: 0.1628 - val_accuracy: 0.9474
Epoch 5/6
90/90 [==============================] - 522s 6s/step - loss: 0.0292 -
accuracy: 0.9912 - val_loss: 0.1628 - val_accuracy: 0.9565
Epoch 6/6
90/90 [==============================] - 519s 6s/step - loss: 0.0193 -
accuracy: 0.9946 - val_loss: 0.0695 - val_accuracy: 0.9794
```

With this working model, we can evaluate the model and make our observations via the following metrics.

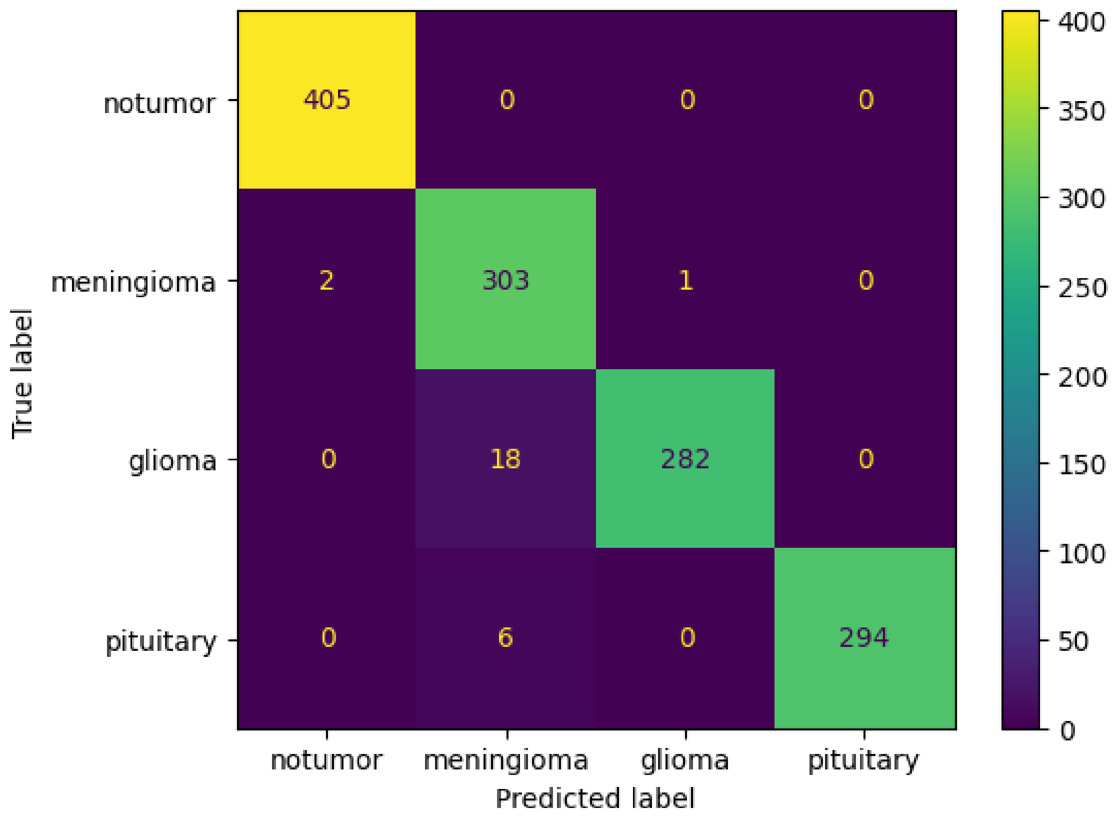
```
def plot_accuracy(history):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.xlabel('Epochs')
    plt.ylabel('Value')
    plt.legend(['Training Accuracy', 'Validation Accuracy'])
    plt.title('Neural Network Accuracy')
    plt.show()

plot_accuracy(history)
```



```python
def plot_loss(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.xlabel('Epochs')
    plt.ylabel('Value')
    plt.legend(['Training Loss', 'Validation Loss'])
    plt.title('Neural Network Loss')
    plt.show()

plot_loss(history)
```



```
def plot_confusion_matrix(model, x_test, y_test):
    y_pred = model.predict(x_test)
    y_pred = np.argmax(y_pred, axis=1)
    y_test = np.argmax(y_test, axis=1)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=encode.keys())
    disp.plot()

plot_confusion_matrix(model, x_test, y_test)

41/41 [==============================] - 16s 397ms/step
```



Finally, we can use our model to make predictions on images outside of our dataset with the following function.

```
def predict_image(model, img):
    img = preprocess_image(img)
    img = np.expand_dims(img, axis=0)
    prediction = model.predict(img)
    prediction = np.argmax(prediction)
    return decode[prediction]

fig, axes = plt.subplots(4, 5, figsize=(15, 10))
axes = axes.ravel()
for i in range(0, 20):
    index = random.randint(0, len(x_test))
    axes[i].imshow(x_test[index], cmap='gray')
    axes[i].set_title(f'Actual: {decode[np.argmax(y_test[index])]} \n
Predicted: {predict_image(model, x_test[index])}')
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.5)
plt.show()
```

```
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
```

```
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
```