MSDS696_C70_Data Science Practicum II

Dilyor Mikhidinov

Using Convolutional Neural Networks to classify MRI Brain Images.

In [1]:
```
#importing neccessary libraries:
import numpy as np
import pandas as pd
import os

#ML libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, InputLayer
from tensorflow.keras.applications import MobileNetV2, VGG19, InceptionV3
from keras import layers
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image_dataset_from_directory as pull
```

In [2]:
```
import os
print(os.listdir("dataset")) #folder where the dataset is located
classes = os.listdir("dataset")
```

```
['Mild_Demented', 'Moderate_Demented', 'Non_Demented', 'Very_Mild_Demented']
```

In [3]:
```
#Visualizations
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
from PIL import Image
```

Checking if all the libraries are working properly. I have had very big issues with installing tensorflow_gpu and openCV

In [4]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2.8.0
```

In [5]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

In [6]:
```
import cv2
```

Next I'm going to create seperate directory to manage labelled data. For fulfilling this task I am using splitfolders

Data Preprocessing

Image Augmentation

Next I am going to use one of the powerful tools of tensorflow.keras called "ImageDataGenerator". This function allows us to take the path to a directory and generate batches of augmented data. Augmentation is important step in almost every Deep learning analysis. Since it allows to modify the existing data we are using in multiple manners so that the trained algorithm becomes capable of generating patterns for even more variety of images. The only case that Augmentation might not be applicable is when the goal is for example to predict the road signs for self driving cars. Signs are always fixed and do not appear for example in vertically flipped way in reality

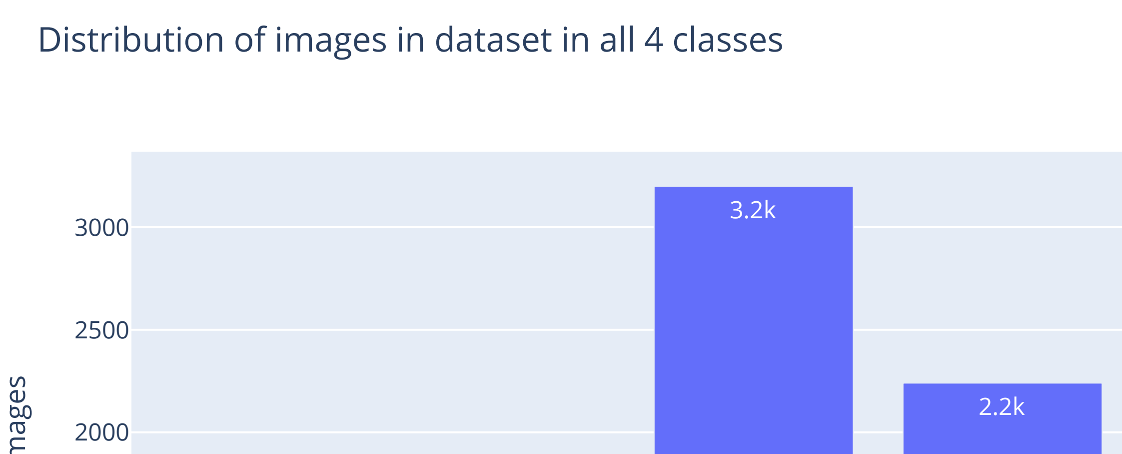
First let's take a look at the dataset distribution

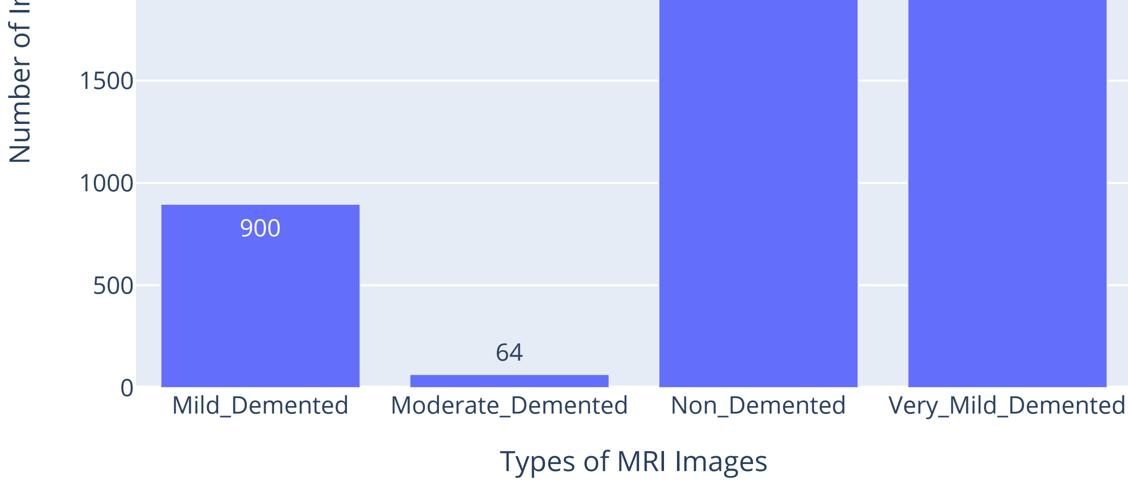
In [7]:

```
# creating variables and saving the number of files in each dataset class folder

mild = './dataset/Mild_Demented'
moderate = './dataset/Moderate_Demented'
non = './dataset/Non_Demented'
very_mild = './dataset/Very_Mild_Demented'

count_mild = 0
count_moderate = 0
count_non = 0
count_very_mild = 0

for path in os.listdir(mild):
    # check if current path is a file
    if os.path.isfile(os.path.join(mild, path)):
        count_mild += 1

for path in os.listdir(moderate):
    if os.path.isfile(os.path.join(moderate, path)):
        count_moderate += 1

for path in os.listdir(non):
    if os.path.isfile(os.path.join(non, path)):
        count_non += 1

for path in os.listdir(very_mild):
    if os.path.isfile(os.path.join(very_mild, path)):
        count_very_mild += 1
```

In [8]:

```
#Plotly visualization
size = [count_mild,count_moderate,count_non,count_very_mild]
fig = px.bar(y=size, x=classes, text_auto='.2s',
             labels={'x':'Types of MRI Images', 'y':'Number of Images'},
             title="Distribution of images in dataset in all 4 classes")
fig.show()
```





In [10]:

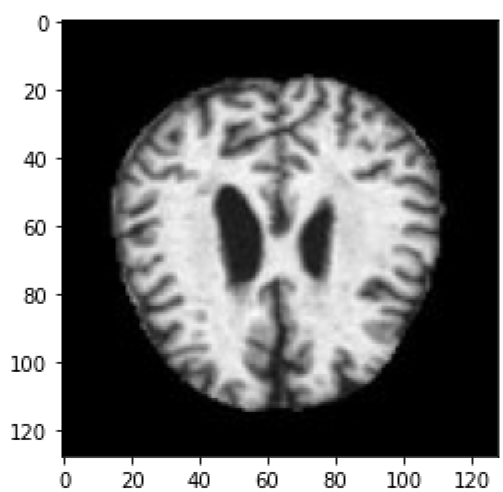
```
import imgaug.augmenters as iaa
import glob
```

In [11]:

```
images = []
images_path = glob.glob("./dataset/Moderate_Demented/*.jpg")
for img_path in images_path:
    img = cv2.imread(img_path)
    images.append(img)
```

In [12]:

```
imgplot = plt.imshow(images[1])
```



Let's Proceed to Image Augmentation and balance the number of images in the dataset

In [13]:

```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load

#following will the property of the new augmented images:
datagen = ImageDataGenerator(shear_range=0.2, zoom_range=0.2)
```

Mild Demented Image Augmentation

In [14]:

```
images_path = glob.glob("./dataset/Mild_Demented/*.jpg")

for f in images_path:
    img = load_img(f)
    x = img_to_array(img)
```

```
    # Reshape the input image
    x = x.reshape((1, ) + x.shape)
    i = 0

    # generate 5 new augmented images
    for batch in datagen.flow(x, batch_size = 1,
                          save_to_dir ='./dataset/Mild_Demented',
                          save_prefix ='mild', save_format ='jpeg'):
        i += 1
        if i > 5:
            break
```

Moderete Demented Image Augmentation

In [32]:

```
images_path = glob.glob("./dataset/Moderate_Demented/*.jpg")

for f in images_path:
    img = load_img(f)
    x = img_to_array(img)
    # Reshape the input image
    x = x.reshape((1, ) + x.shape)
    i = 0

    # generate 5 new augmented images
    for batch in datagen.flow(x, batch_size = 1,
                          save_to_dir ='./dataset/Moderate_Demented',
                          save_prefix ='moderate', save_format ='jpeg'):
        i += 1
        if i > 5:
            break
```

Non Demented Image Augmentation

In [ ]:

```
images_path = glob.glob("./dataset/Non_Demented/*.jpg")

for f in images_path:
    img = load_img(f)
    x = img_to_array(img)
    # Reshape the input image
    x = x.reshape((1, ) + x.shape)
    i = 0

    # generate 5 new augmented images
    for batch in datagen.flow(x, batch_size = 1,
                          save_to_dir ='./dataset/Non_Demented',
                          save_prefix ='non', save_format ='jpeg'):
        i += 1
        if i > 5:
            break
```

Very Mild Demented Demented Image Augmentation

In [ ]:

```
images_path = glob.glob("./dataset/Very_Mild_Demented/*.jpg")

for f in images_path:
    img = load_img(f)
    x = img_to_array(img)
    # Reshape the input image
    x = x.reshape((1, ) + x.shape)
    i = 0
```

```
    # generate 5 new augmented images
    for batch in datagen.flow(x, batch_size = 1,
                              save_to_dir ='./dataset/Very_Mild_Demented',
                              save_prefix ='verymild', save_format ='jpeg'):
        i += 1
        if i > 5:
            break
```

Now we used Data Augmentation and generated some new modified images the dataset looks much balanced between different image classes

Split Folders

In [35]:

```
#Now I am creating a work folder where I will be splitting dataset into training,
#testing and validation folders:

import splitfolders
splitfolders.ratio('dataset', output="work", seed=1345, ratio=(.8, 0.1,0.1))
```

```
Copying files: 20649 files [01:30, 227.11 files/s]
```

In [36]:

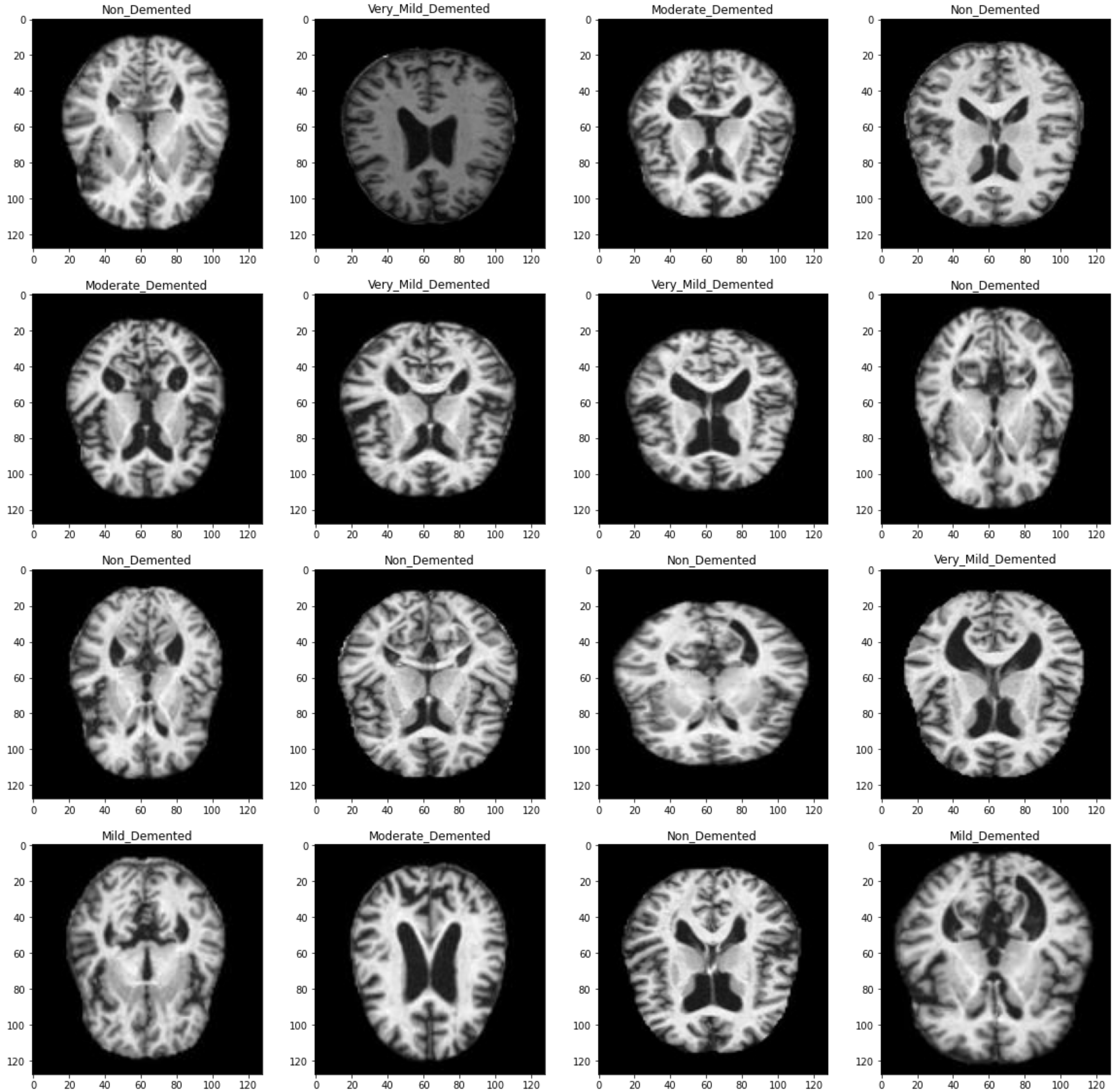
```
height = 128
width = 128
train_data = pull("./work/train",seed=123,image_size=(height, width),batch_size=64)

test_data = pull("./work/test",seed=123,image_size=(height, width),batch_size=64)

val_data = pull("./work/val",seed=123,image_size=(height, width),batch_size=64)
```

```
Found 16517 files belonging to 4 classes.
Found 2069 files belonging to 4 classes.
Found 2063 files belonging to 4 classes.
```

Some Visualizations

In [37]:

```
plt.figure(figsize=(20, 20))
for images, labels in train_data.take(1):
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(classes[labels[i]])
```



Let's now look at the distribution of images in 4 classes within training folder

In [9]:

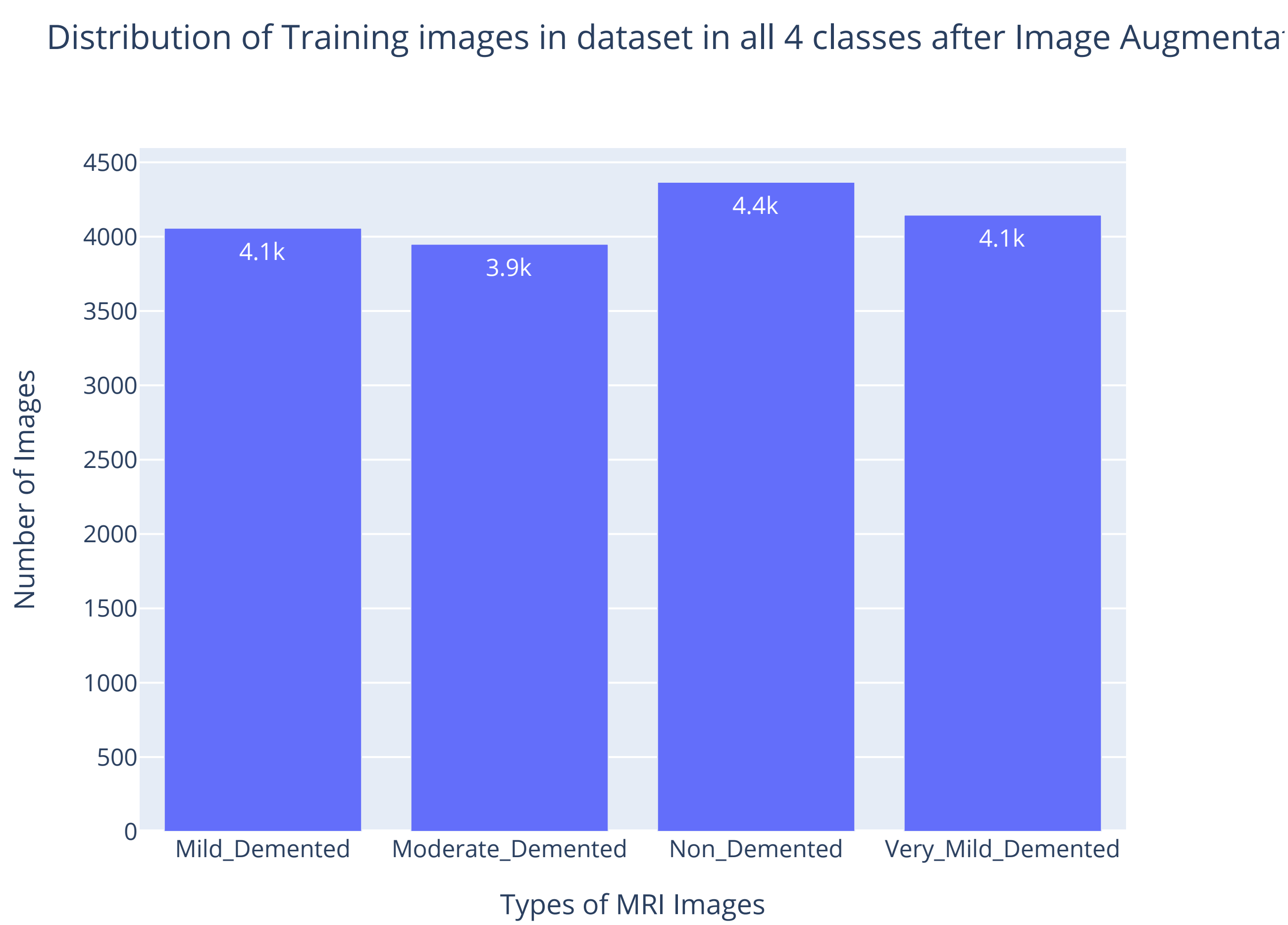
```
mild = './work/train/Mild_Demented'
moderate = './work/train/Moderate_Demented'
non = './work/train/Non_Demented'
very_mild = './work/train/Very_Mild_Demented'

count_mild = 0
count_moderate = 0
count_non = 0
count_very_mild = 0

for path in os.listdir(mild):
    # check if current path is a file
    if os.path.isfile(os.path.join(mild, path)):
        count_mild += 1

for path in os.listdir(moderate):
    # check if current path is a file
    if os.path.isfile(os.path.join(moderate, path)):
```

```
            count_moderate += 1

    for path in os.listdir(non):
        # check if current path is a file
        if os.path.isfile(os.path.join(non, path)):
            count_non += 1

    for path in os.listdir(very_mild):
        # check if current path is a file
        if os.path.isfile(os.path.join(very_mild, path)):
            count_very_mild += 1

    print('File count:', count_mild)
```

```
File count: 4057
```

In [10]:

```
#Plotly visualization
size = [count_mild,count_moderate,count_non,count_very_mild]
fig = px.bar(y=size, x=classes, text_auto='.2s',
             labels={'x':'Types of MRI Images', 'y':'Number of Images'},
             title="Distribution of Training images in dataset in all 4 classes after Imag
fig.show()
```



Looking at different CNN models

During the project timeline I viewed different CNN models and checked their performances with evaluation metrics. And from the analysis I selected top 3 best performing models: DS-CNN, VGG19-InceptionV3

Conv2d Model 1

In [41]:

```
input_length = 128,128,3

model_1 = Sequential()
model_1.add(Conv2D(64,(3,3),strides =(1,1),  padding='valid', activation='relu',input_shap
model_1.add(MaxPool2D(2,2))
model_1.add(Dropout(0.2))

model_1.add(Conv2D(32,(3,3), activation='relu'))
model_1.add(MaxPool2D(2,2))
model_1.add(Dropout(0.2))

model_1.add(SeparableConv2D(16,(3,3),activation='relu'))
model_1.add(MaxPool2D(2,2))
model_1.add(Dropout(0.3))

model_1.add(Flatten())
model_1.add(Dense(16))

model_1.add(Dense(4,activation='sigmoid'))

opt = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999)
model_1.compile(optimizer= opt, loss='sparse_categorical_crossentropy', metrics=['accuracy
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 64)      1792

 max_pooling2d (MaxPooling2D  (None, 63, 63, 64)       0
 )

 dropout (Dropout)           (None, 63, 63, 64)        0

 conv2d_1 (Conv2D)           (None, 61, 61, 32)        18464

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 30, 30, 32)        0

 separable_conv2d (Separable  (None, 28, 28, 16)       816
 Conv2D)

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 16)       0
 2D)

 dropout_2 (Dropout)         (None, 14, 14, 16)        0

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 16)                50192

 dense_1 (Dense)             (None, 4)                 68

=================================================================
Total params: 71,332
Trainable params: 71,332
Non-trainable params: 0
_________________________________________________________________
```

In [42]:

```
history = model_1.fit(train_data,epochs=50,validation_data = val_data, batch_size=64, verb
```

```
Epoch 1/50
259/259 [==============================] - 26s 82ms/step - loss: 2.0915 - accuracy: 0.3967
- val_loss: 1.0015 - val_accuracy: 0.5919
Epoch 2/50
259/259 [==============================] - 18s 70ms/step - loss: 0.9674 - accuracy: 0.5892
- val_loss: 0.8500 - val_accuracy: 0.6587
Epoch 3/50
259/259 [==============================] - 43s 166ms/step - loss: 0.8718 - accuracy: 0.633
6 - val_loss: 0.7601 - val_accuracy: 0.7038
Epoch 4/50
259/259 [==============================] - 68s 260ms/step - loss: 0.8177 - accuracy: 0.660
0 - val_loss: 0.7350 - val_accuracy: 0.7009
Epoch 5/50
259/259 [==============================] - 67s 260ms/step - loss: 0.7672 - accuracy: 0.676
7 - val_loss: 0.7193 - val_accuracy: 0.7179
Epoch 6/50
259/259 [==============================] - 69s 268ms/step - loss: 0.7068 - accuracy: 0.704
4 - val_loss: 0.6774 - val_accuracy: 0.7159
Epoch 7/50
259/259 [==============================] - 68s 263ms/step - loss: 0.6664 - accuracy: 0.719
6 - val_loss: 0.6155 - val_accuracy: 0.7373
Epoch 8/50
259/259 [==============================] - 67s 257ms/step - loss: 0.6446 - accuracy: 0.730
4 - val_loss: 0.6183 - val_accuracy: 0.7484
Epoch 9/50
259/259 [==============================] - 67s 259ms/step - loss: 0.6331 - accuracy: 0.733
0 - val_loss: 0.5405 - val_accuracy: 0.7882
Epoch 10/50
259/259 [==============================] - 68s 260ms/step - loss: 0.5978 - accuracy: 0.748
9 - val_loss: 0.5438 - val_accuracy: 0.7790
Epoch 11/50
259/259 [==============================] - 67s 259ms/step - loss: 0.5707 - accuracy: 0.765
9 - val_loss: 0.5493 - val_accuracy: 0.7596
Epoch 12/50
259/259 [==============================] - 67s 259ms/step - loss: 0.5663 - accuracy: 0.762
2 - val_loss: 0.5033 - val_accuracy: 0.7964
Epoch 13/50
259/259 [==============================] - 60s 229ms/step - loss: 0.5448 - accuracy: 0.771
0 - val_loss: 0.4534 - val_accuracy: 0.8216
Epoch 14/50
259/259 [==============================] - 67s 258ms/step - loss: 0.5250 - accuracy: 0.781
2 - val_loss: 0.4562 - val_accuracy: 0.8168
Epoch 15/50
259/259 [==============================] - 68s 261ms/step - loss: 0.5082 - accuracy: 0.785
7 - val_loss: 0.4399 - val_accuracy: 0.8197
Epoch 16/50
259/259 [==============================] - 69s 266ms/step - loss: 0.5042 - accuracy: 0.792
4 - val_loss: 0.4581 - val_accuracy: 0.8051
Epoch 17/50
259/259 [==============================] - 69s 267ms/step - loss: 0.4899 - accuracy: 0.797
0 - val_loss: 0.4858 - val_accuracy: 0.7930
Epoch 18/50
259/259 [==============================] - 68s 260ms/step - loss: 0.4848 - accuracy: 0.798
4 - val_loss: 0.4360 - val_accuracy: 0.8163
Epoch 19/50
259/259 [==============================] - 68s 260ms/step - loss: 0.4690 - accuracy: 0.801
5 - val_loss: 0.4098 - val_accuracy: 0.8400
Epoch 20/50
259/259 [==============================] - 69s 266ms/step - loss: 0.4661 - accuracy: 0.806
9 - val_loss: 0.4092 - val_accuracy: 0.8371
Epoch 21/50
259/259 [==============================] - 67s 257ms/step - loss: 0.4550 - accuracy: 0.812
6 - val_loss: 0.3962 - val_accuracy: 0.8454
Epoch 22/50
259/259 [==============================] - 69s 264ms/step - loss: 0.4512 - accuracy: 0.811
2 - val_loss: 0.4030 - val_accuracy: 0.8415
```

```
Epoch 23/50
259/259 [==============================] - 64s 246ms/step - loss: 0.4388 - accuracy: 0.815
9 - val_loss: 0.3826 - val_accuracy: 0.8512
Epoch 24/50
259/259 [==============================] - 36s 136ms/step - loss: 0.4320 - accuracy: 0.821
2 - val_loss: 0.4244 - val_accuracy: 0.8250
Epoch 25/50
259/259 [==============================] - 20s 75ms/step - loss: 0.4429 - accuracy: 0.8154
- val_loss: 0.3784 - val_accuracy: 0.8560
Epoch 26/50
259/259 [==============================] - 20s 75ms/step - loss: 0.4285 - accuracy: 0.8233
- val_loss: 0.3990 - val_accuracy: 0.8429
Epoch 27/50
259/259 [==============================] - 19s 74ms/step - loss: 0.4133 - accuracy: 0.8314
- val_loss: 0.3404 - val_accuracy: 0.8691
Epoch 28/50
259/259 [==============================] - 20s 75ms/step - loss: 0.4093 - accuracy: 0.8304
- val_loss: 0.4024 - val_accuracy: 0.8381
Epoch 29/50
259/259 [==============================] - 20s 75ms/step - loss: 0.4175 - accuracy: 0.8274
- val_loss: 0.3652 - val_accuracy: 0.8619
Epoch 30/50
259/259 [==============================] - 20s 75ms/step - loss: 0.4054 - accuracy: 0.8348
- val_loss: 0.3723 - val_accuracy: 0.8546
Epoch 31/50
259/259 [==============================] - 19s 74ms/step - loss: 0.3884 - accuracy: 0.8467
- val_loss: 0.3700 - val_accuracy: 0.8585
Epoch 32/50
259/259 [==============================] - 19s 75ms/step - loss: 0.3934 - accuracy: 0.8382
- val_loss: 0.3346 - val_accuracy: 0.8745
Epoch 33/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3814 - accuracy: 0.8449
- val_loss: 0.3091 - val_accuracy: 0.8817
Epoch 34/50
259/259 [==============================] - 19s 75ms/step - loss: 0.3813 - accuracy: 0.8442
- val_loss: 0.3237 - val_accuracy: 0.8817
Epoch 35/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3862 - accuracy: 0.8448
- val_loss: 0.3275 - val_accuracy: 0.8841
Epoch 36/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3743 - accuracy: 0.8455
- val_loss: 0.3167 - val_accuracy: 0.8827
Epoch 37/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3689 - accuracy: 0.8525
- val_loss: 0.3268 - val_accuracy: 0.8783
Epoch 38/50
259/259 [==============================] - 20s 77ms/step - loss: 0.3752 - accuracy: 0.8467
- val_loss: 0.3221 - val_accuracy: 0.8812
Epoch 39/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3612 - accuracy: 0.8520
- val_loss: 0.2831 - val_accuracy: 0.8948
Epoch 40/50
259/259 [==============================] - 19s 74ms/step - loss: 0.3549 - accuracy: 0.8557
- val_loss: 0.2950 - val_accuracy: 0.9021
Epoch 41/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3572 - accuracy: 0.8554
- val_loss: 0.3113 - val_accuracy: 0.8817
Epoch 42/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3590 - accuracy: 0.8530
- val_loss: 0.3119 - val_accuracy: 0.8837
Epoch 43/50
259/259 [==============================] - 20s 76ms/step - loss: 0.3471 - accuracy: 0.8611
- val_loss: 0.3199 - val_accuracy: 0.8798
Epoch 44/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3458 - accuracy: 0.8611
- val_loss: 0.2908 - val_accuracy: 0.8948
```

```
Epoch 45/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3526 - accuracy: 0.8542
- val_loss: 0.3203 - val_accuracy: 0.8778
Epoch 46/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3450 - accuracy: 0.8595
- val_loss: 0.2888 - val_accuracy: 0.8856
Epoch 47/50
259/259 [==============================] - 20s 76ms/step - loss: 0.3396 - accuracy: 0.8627
- val_loss: 0.2865 - val_accuracy: 0.8968
Epoch 48/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3475 - accuracy: 0.8595
- val_loss: 0.2914 - val_accuracy: 0.8972
Epoch 49/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3316 - accuracy: 0.8670
- val_loss: 0.2792 - val_accuracy: 0.8992
Epoch 50/50
259/259 [==============================] - 20s 75ms/step - loss: 0.3306 - accuracy: 0.8687
- val_loss: 0.2778 - val_accuracy: 0.8909
```

In [43]:

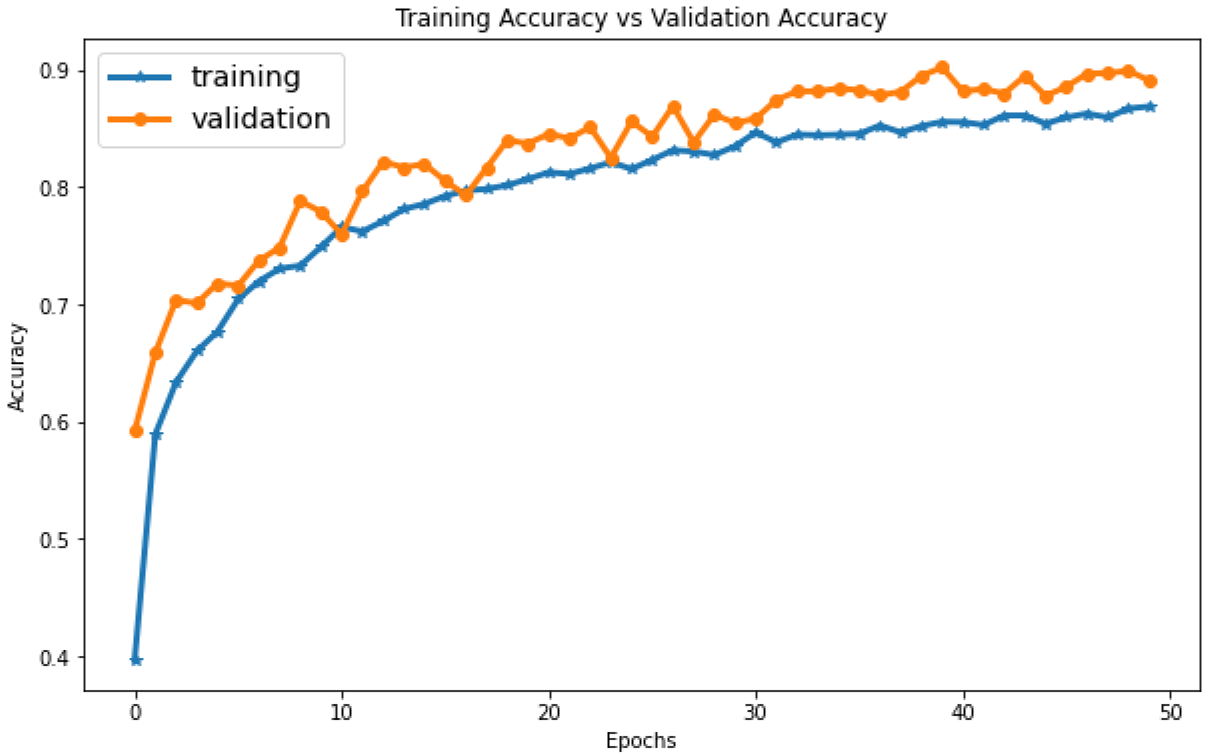
```
model1_s = 'model1_train.h5'
model_1.save(model1_s)
```

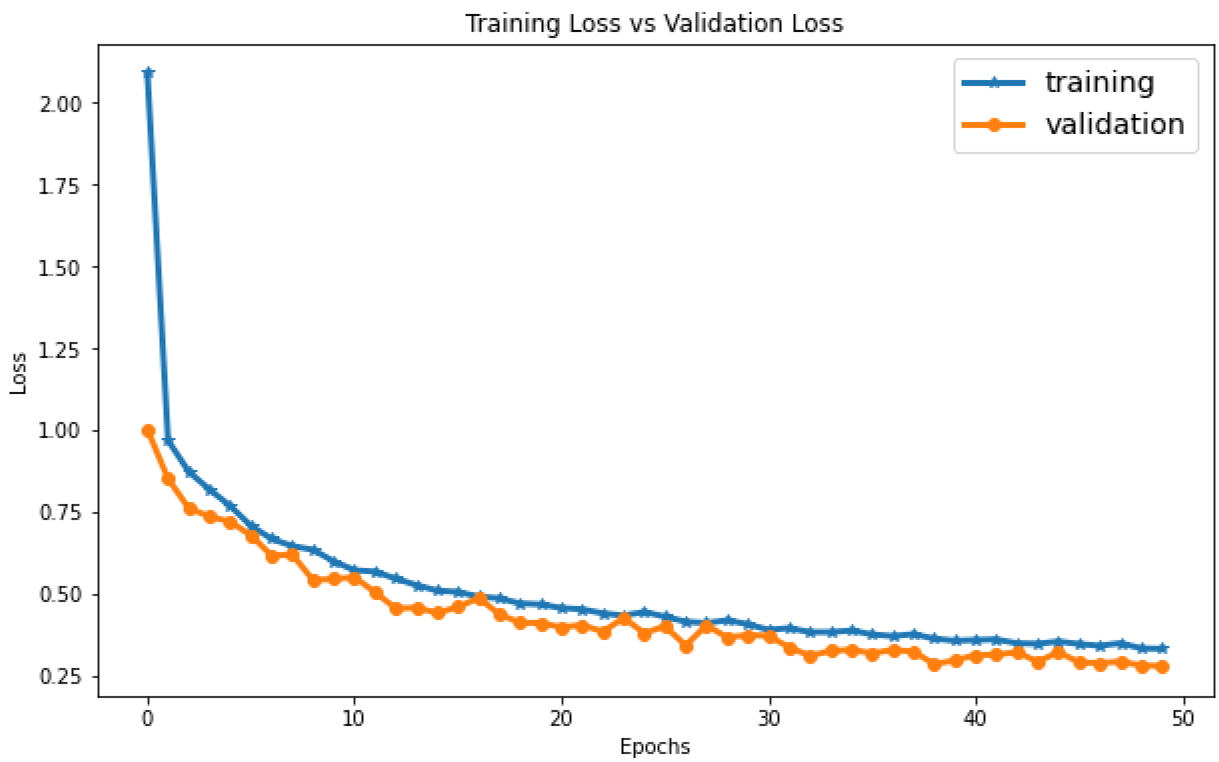
In [ ]:

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=history.history['val_accuracy'], y=random_y0,
                    mode='lines',
                    name='lines'))


fig.show()
```

In [86]:

```
def visualize_training(history, lw = 3):
    plt.figure(figsize=(10,6))
    plt.plot(history.history['accuracy'], label = 'training', marker = '*', linewidth = lw
    plt.plot(history.history['val_accuracy'], label = 'validation', marker = 'o', linewidt
    plt.title('Training Accuracy vs Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend(fontsize = 'x-large')
    plt.show()

    plt.figure(figsize=(10,6))
    plt.plot(history.history['loss'], label = 'training', marker = '*', linewidth = lw)
    plt.plot(history.history['val_loss'], label = 'validation', marker = 'o', linewidth =
    plt.title('Training Loss vs Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend(fontsize = 'x-large')
    plt.show()
visualize_training(history)
```





In [46]:

```
accuracy = history.history['accuracy']
loss = history.history['loss']
val_accuracy = history.history['val_accuracy']
val_loss = history.history['val_loss']

print(f'Training Accuracy: {np.max(accuracy)}')
print(f'Training Loss: {np.min(loss)}')
print(f'Validation Accuracy: {np.max(val_accuracy)}')
print(f'Validation Loss: {np.min(val_loss)}')
```

```
Training Accuracy: 0.8686807751655579
Training Loss: 0.3306235074996948
Validation Accuracy: 0.9020843505859375
Validation Loss: 0.27783992886543274
```

As we can see first Model 1 performaned not too bad on training

Conv2D Model 2

In [47]:

```
model_2 = keras.models.Sequential()
model_2.add(keras.layers.experimental.preprocessing.Rescaling(1./255, input_shape=(128,128
model_2.add(keras.layers.Conv2D(filters=16,kernel_size=(3,3),padding='same',activation='re
model_2.add(keras.layers.MaxPooling2D(pool_size=(2,2)))


model_2.add(keras.layers.Conv2D(filters=32,kernel_size=(3,3),padding='same',activation='re
model_2.add(keras.layers.MaxPooling2D(pool_size=(2,2)))

model_2.add(keras.layers.Dropout(0.20))

model_2.add(keras.layers.Conv2D(filters=64,kernel_size=(3,3),padding='same',activation='re
model_2.add(keras.layers.MaxPooling2D(pool_size=(2,2)))

model_2.add(keras.layers.Dropout(0.25))
model_2.add(keras.layers.Flatten())
model_2.add(keras.layers.Dense(128,activation="relu",kernel_initializer="he_normal"))
#model_2.add(keras.layers.Dense(64,"relu"))
model_2.add(keras.layers.Dense(4,"sigmoid"))
model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 rescaling (Rescaling)       (None, 128, 128, 3)       0         
                                                                 
 conv2d_2 (Conv2D)           (None, 128, 128, 16)      448       
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 64, 64, 16)       0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 64, 64, 32)        4640      
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 32, 32, 32)       0         
 2D)                                                             
                                                                 
 dropout_3 (Dropout)         (None, 32, 32, 32)        0         
                                                                 
 conv2d_4 (Conv2D)           (None, 32, 32, 64)        18496     
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 16, 16, 64)       0         
 2D)                                                             
                                                                 
 dropout_4 (Dropout)         (None, 16, 16, 64)        0         
                                                                 
 flatten_1 (Flatten)         (None, 16384)             0         
                                                                 
 dense_2 (Dense)             (None, 128)               2097280   
                                                                 
 dense_3 (Dense)             (None, 4)                 516       
                                                                 
=================================================================
Total params: 2,121,380
Trainable params: 2,121,380
Non-trainable params: 0
_________________________________________________________________
```

In [48]:

```
model_2.compile(loss="sparse_categorical_crossentropy",
              optimizer = "Adam",metrics=["accuracy"])
```

In [49]:
```
history2 = model_2.fit(train_data,validation_data=val_data,epochs=50, batch_size=64, verbo
```

```
Epoch 1/50
259/259 [==============================] - 12s 44ms/step - loss: 1.1412 - accuracy: 0.5580
- val_loss: 0.6041 - val_accuracy: 0.7557
Epoch 2/50
259/259 [==============================] - 12s 44ms/step - loss: 0.5443 - accuracy: 0.7678
- val_loss: 0.4400 - val_accuracy: 0.8216
Epoch 3/50
259/259 [==============================] - 12s 44ms/step - loss: 0.4114 - accuracy: 0.8279
- val_loss: 0.3339 - val_accuracy: 0.8817
Epoch 4/50
259/259 [==============================] - 12s 44ms/step - loss: 0.3134 - accuracy: 0.8726
- val_loss: 0.2923 - val_accuracy: 0.8880
Epoch 5/50
259/259 [==============================] - 12s 44ms/step - loss: 0.2506 - accuracy: 0.9011
- val_loss: 0.2490 - val_accuracy: 0.8997
Epoch 6/50
259/259 [==============================] - 11s 44ms/step - loss: 0.2164 - accuracy: 0.9151
- val_loss: 0.2124 - val_accuracy: 0.9161
Epoch 7/50
259/259 [==============================] - 12s 44ms/step - loss: 0.1683 - accuracy: 0.9342
- val_loss: 0.2054 - val_accuracy: 0.9258
Epoch 8/50
259/259 [==============================] - 12s 44ms/step - loss: 0.1342 - accuracy: 0.9498
- val_loss: 0.1069 - val_accuracy: 0.9641
Epoch 9/50
259/259 [==============================] - 11s 43ms/step - loss: 0.1184 - accuracy: 0.9548
- val_loss: 0.0862 - val_accuracy: 0.9719
Epoch 10/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0992 - accuracy: 0.9637
- val_loss: 0.0991 - val_accuracy: 0.9651
Epoch 11/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0864 - accuracy: 0.9687
- val_loss: 0.1246 - val_accuracy: 0.9530
Epoch 12/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0780 - accuracy: 0.9700
- val_loss: 0.0841 - val_accuracy: 0.9714
Epoch 13/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0646 - accuracy: 0.9759
- val_loss: 0.0681 - val_accuracy: 0.9787
Epoch 14/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0675 - accuracy: 0.9764
- val_loss: 0.0713 - val_accuracy: 0.9758
Epoch 15/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0596 - accuracy: 0.9769
- val_loss: 0.0668 - val_accuracy: 0.9767
Epoch 16/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0499 - accuracy: 0.9806
- val_loss: 0.0561 - val_accuracy: 0.9825
Epoch 17/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0462 - accuracy: 0.9835
- val_loss: 0.1112 - val_accuracy: 0.9636
Epoch 18/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0481 - accuracy: 0.9826
- val_loss: 0.0468 - val_accuracy: 0.9864
Epoch 19/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0407 - accuracy: 0.9848
- val_loss: 0.0444 - val_accuracy: 0.9864
Epoch 20/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0379 - accuracy: 0.9862
- val_loss: 0.0425 - val_accuracy: 0.9859
Epoch 21/50
```

```
259/259 [==============================] - 11s 44ms/step - loss: 0.0327 - accuracy: 0.9883
- val_loss: 0.0553 - val_accuracy: 0.9821
Epoch 22/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0341 - accuracy: 0.9870
- val_loss: 0.0440 - val_accuracy: 0.9850
Epoch 23/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0286 - accuracy: 0.9906
- val_loss: 0.0639 - val_accuracy: 0.9840
Epoch 24/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0332 - accuracy: 0.9886
- val_loss: 0.0518 - val_accuracy: 0.9835
Epoch 25/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0330 - accuracy: 0.9889
- val_loss: 0.0500 - val_accuracy: 0.9835
Epoch 26/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0255 - accuracy: 0.9916
- val_loss: 0.0497 - val_accuracy: 0.9864
Epoch 27/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0226 - accuracy: 0.9924
- val_loss: 0.0637 - val_accuracy: 0.9830
Epoch 28/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0263 - accuracy: 0.9910
- val_loss: 0.0519 - val_accuracy: 0.9869
Epoch 29/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0268 - accuracy: 0.9903
- val_loss: 0.0451 - val_accuracy: 0.9884
Epoch 30/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0240 - accuracy: 0.9910
- val_loss: 0.0368 - val_accuracy: 0.9855
Epoch 31/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0227 - accuracy: 0.9921
- val_loss: 0.0383 - val_accuracy: 0.9874
Epoch 32/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0214 - accuracy: 0.9927
- val_loss: 0.0593 - val_accuracy: 0.9850
Epoch 33/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0236 - accuracy: 0.9923
- val_loss: 0.0590 - val_accuracy: 0.9835
Epoch 34/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0206 - accuracy: 0.9930
- val_loss: 0.0350 - val_accuracy: 0.9884
Epoch 35/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0167 - accuracy: 0.9948
- val_loss: 0.0373 - val_accuracy: 0.9879
Epoch 36/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0154 - accuracy: 0.9941
- val_loss: 0.0598 - val_accuracy: 0.9879
Epoch 37/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0232 - accuracy: 0.9924
- val_loss: 0.0522 - val_accuracy: 0.9859
Epoch 38/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0171 - accuracy: 0.9945
- val_loss: 0.0405 - val_accuracy: 0.9874
Epoch 39/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0165 - accuracy: 0.9948
- val_loss: 0.0412 - val_accuracy: 0.9874
Epoch 40/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0205 - accuracy: 0.9935
- val_loss: 0.0523 - val_accuracy: 0.9850
Epoch 41/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0204 - accuracy: 0.9932
- val_loss: 0.0504 - val_accuracy: 0.9889
Epoch 42/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0210 - accuracy: 0.9926
- val_loss: 0.0482 - val_accuracy: 0.9884
Epoch 43/50
```

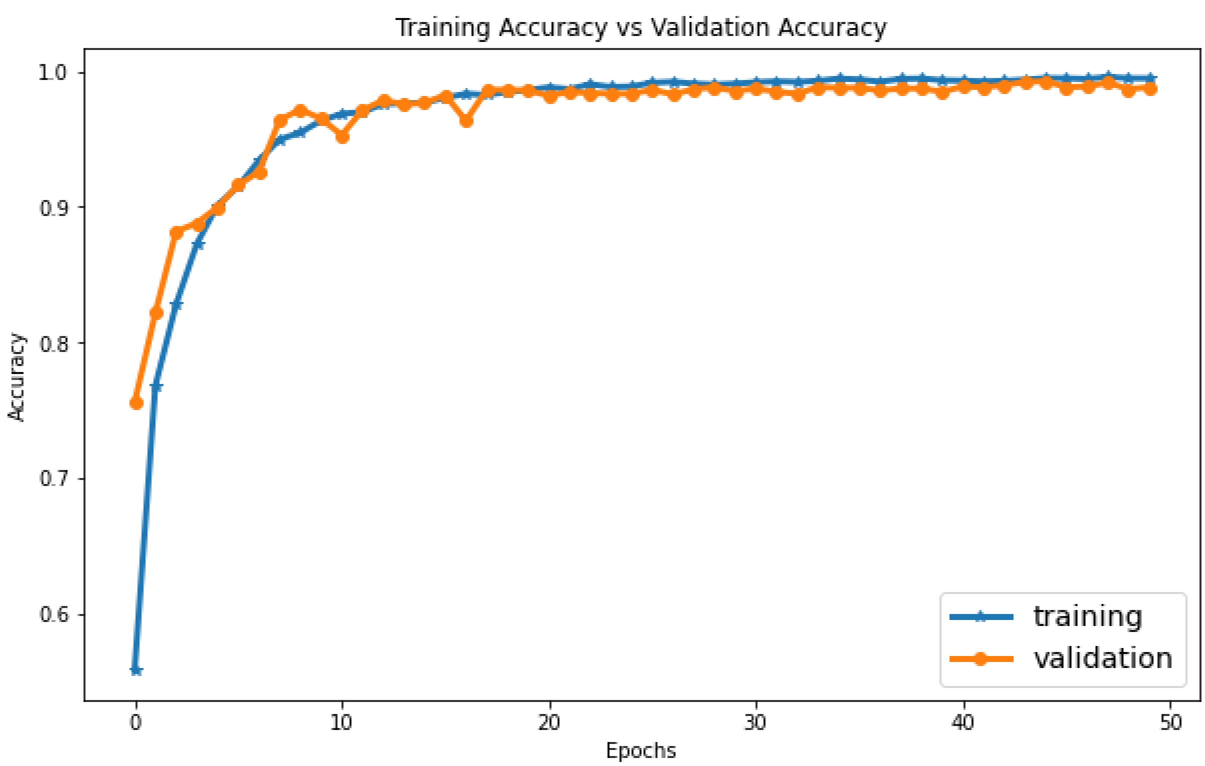
```
259/259 [==============================] - 12s 44ms/step - loss: 0.0197 - accuracy: 0.9931
- val_loss: 0.0401 - val_accuracy: 0.9898
Epoch 44/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0174 - accuracy: 0.9939
- val_loss: 0.0330 - val_accuracy: 0.9922
Epoch 45/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0149 - accuracy: 0.9951
- val_loss: 0.0331 - val_accuracy: 0.9927
Epoch 46/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0170 - accuracy: 0.9949
- val_loss: 0.0481 - val_accuracy: 0.9884
Epoch 47/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0176 - accuracy: 0.9945
- val_loss: 0.0449 - val_accuracy: 0.9889
Epoch 48/50
259/259 [==============================] - 12s 44ms/step - loss: 0.0122 - accuracy: 0.9959
- val_loss: 0.0351 - val_accuracy: 0.9918
Epoch 49/50
259/259 [==============================] - 11s 43ms/step - loss: 0.0154 - accuracy: 0.9950
- val_loss: 0.0370 - val_accuracy: 0.9869
Epoch 50/50
259/259 [==============================] - 11s 44ms/step - loss: 0.0161 - accuracy: 0.9949
- val_loss: 0.0303 - val_accuracy: 0.9884
```

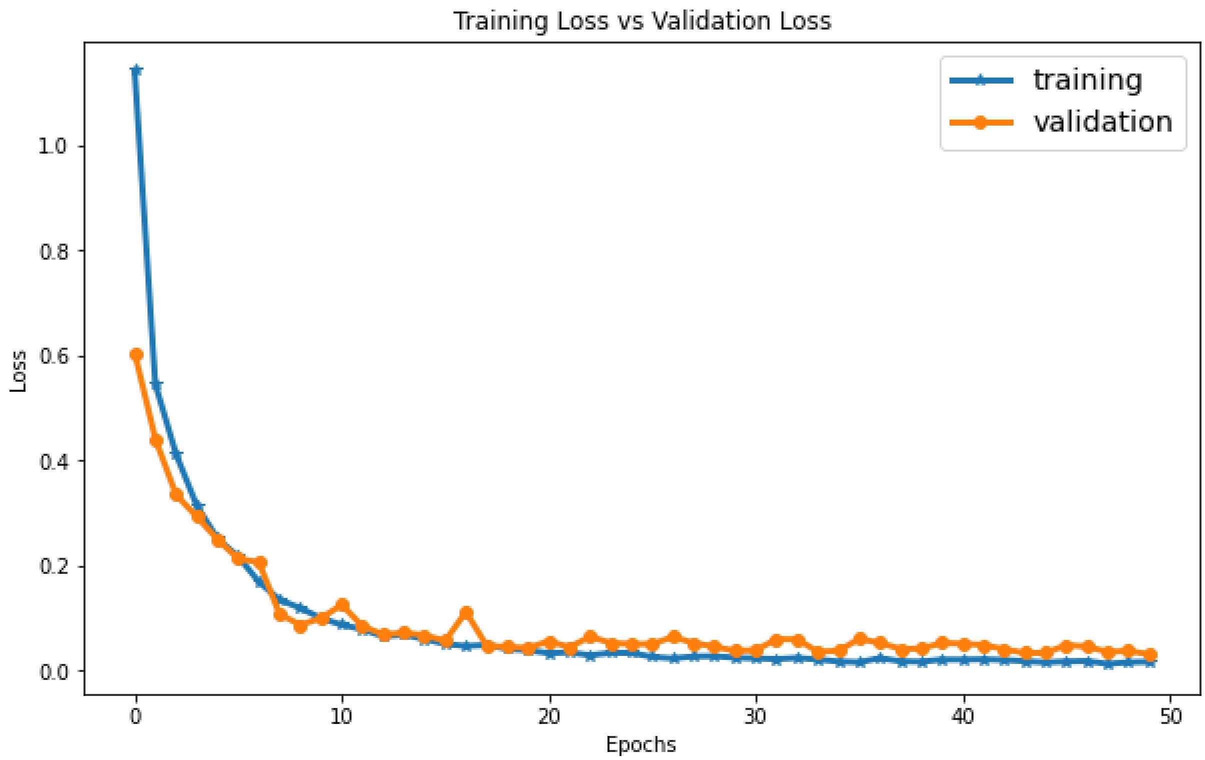
In [50]:

```
model2_s = 'model2_train.h5'
model_2.save(model2_s)
```

In [51]:

```
visualize_training(history2)
```





In [52]:

```
accuracy = history2.history['accuracy']
loss = history2.history['loss']
val_accuracy = history2.history['val_accuracy']
val_loss = history2.history['val_loss']

print(f'Training Accuracy: {np.max(accuracy)}')
print(f'Training Loss: {np.min(loss)}')
print(f'Validation Accuracy: {np.max(val_accuracy)}')
print(f'Validation Loss: {np.min(val_loss)}')
```

```
Training Accuracy: 0.995943546295166
Training Loss: 0.012162092141807
Validation Accuracy: 0.9927290081977844
Validation Loss: 0.03030194155871868
```

VGG19 Model

In [53]:

```
vgg_model = Sequential()
vgg_model.add(VGG19(include_top=False,weights='imagenet', input_shape=(128, 128, 3)))
vgg_model.add(Flatten())
vgg_model.add(Dense(64,activation='relu'))
vgg_model.add(Dense(16,activation='relu'))
vgg_model.add(Dense(4,activation = 'softmax'))

#vgg_model.layers[0].trainable = False

opt = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999)
vgg_model.compile(optimizer= opt, loss='sparse_categorical_crossentropy', metrics=['accura
vgg_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 4, 4, 512)         20024384

 flatten_2 (Flatten)         (None, 8192)              0
```

```
 dense_4 (Dense)             (None, 64)                524352

 dense_5 (Dense)             (None, 16)                1040

 dense_6 (Dense)             (None, 4)                 68

=================================================================
Total params: 20,549,844
Trainable params: 20,549,844
Non-trainable params: 0
_________________________________________________________________
```

In [54]:

```
vgg_history = vgg_model.fit(train_data,epochs=50,validation_data = val_data,verbose=1)
```

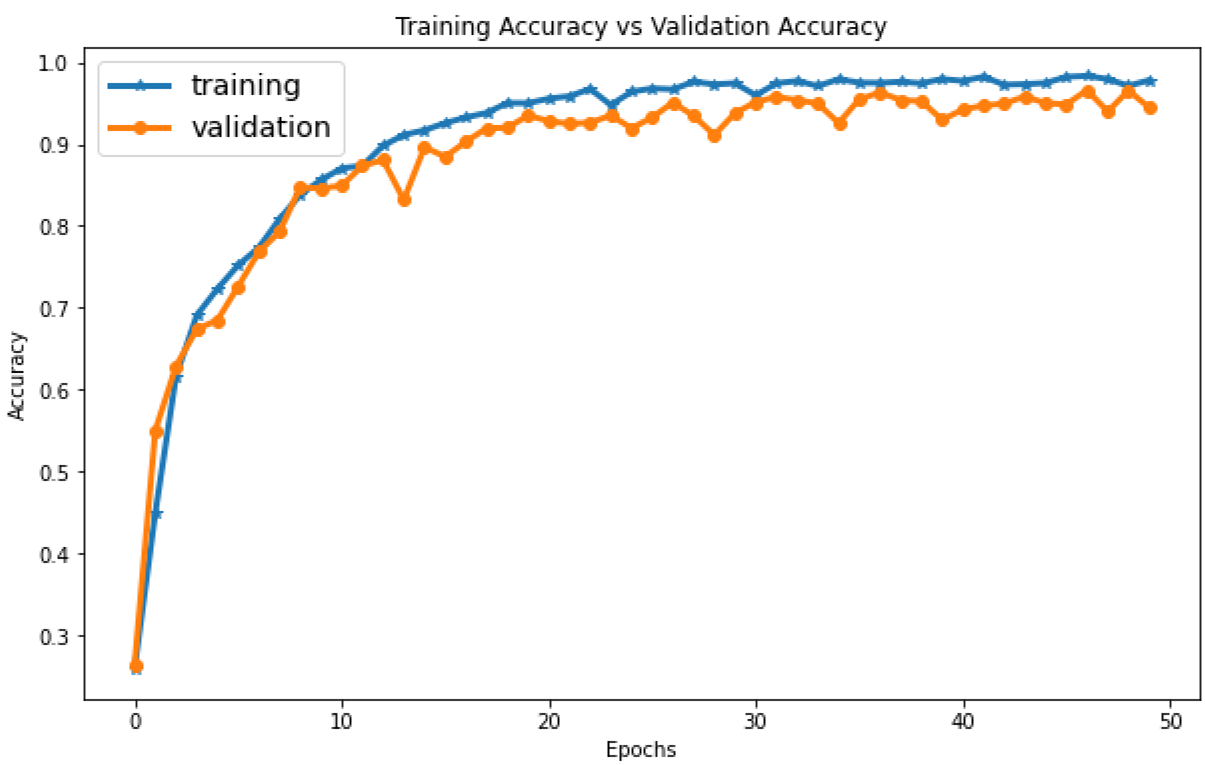
```
Epoch 1/50
259/259 [==============================] - 95s 354ms/step - loss: 2.3667 - accuracy: 0.258
5 - val_loss: 1.3829 - val_accuracy: 0.2642
Epoch 2/50
259/259 [==============================] - 90s 348ms/step - loss: 1.1679 - accuracy: 0.449
5 - val_loss: 0.9779 - val_accuracy: 0.5502
Epoch 3/50
259/259 [==============================] - 91s 349ms/step - loss: 0.8422 - accuracy: 0.616
3 - val_loss: 0.7985 - val_accuracy: 0.6287
Epoch 4/50
259/259 [==============================] - 91s 349ms/step - loss: 0.6621 - accuracy: 0.690
7 - val_loss: 0.7520 - val_accuracy: 0.6738
Epoch 5/50
259/259 [==============================] - 91s 350ms/step - loss: 0.6022 - accuracy: 0.723
2 - val_loss: 0.6791 - val_accuracy: 0.6849
Epoch 6/50
259/259 [==============================] - 91s 349ms/step - loss: 0.5427 - accuracy: 0.752
0 - val_loss: 0.5981 - val_accuracy: 0.7261
Epoch 7/50
259/259 [==============================] - 90s 349ms/step - loss: 0.5017 - accuracy: 0.773
6 - val_loss: 0.5022 - val_accuracy: 0.7683
Epoch 8/50
259/259 [==============================] - 91s 349ms/step - loss: 0.4285 - accuracy: 0.808
6 - val_loss: 0.4851 - val_accuracy: 0.7930
Epoch 9/50
259/259 [==============================] - 91s 349ms/step - loss: 0.3760 - accuracy: 0.838
0 - val_loss: 0.3517 - val_accuracy: 0.8473
Epoch 10/50
259/259 [==============================] - 91s 350ms/step - loss: 0.3335 - accuracy: 0.856
3 - val_loss: 0.3702 - val_accuracy: 0.8459
Epoch 11/50
259/259 [==============================] - 91s 350ms/step - loss: 0.2983 - accuracy: 0.870
5 - val_loss: 0.3495 - val_accuracy: 0.8492
Epoch 12/50
259/259 [==============================] - 91s 351ms/step - loss: 0.2983 - accuracy: 0.873
5 - val_loss: 0.3375 - val_accuracy: 0.8735
Epoch 13/50
259/259 [==============================] - 92s 353ms/step - loss: 0.2415 - accuracy: 0.898
2 - val_loss: 0.2695 - val_accuracy: 0.8808
Epoch 14/50
259/259 [==============================] - 91s 351ms/step - loss: 0.2143 - accuracy: 0.911
3 - val_loss: 0.4234 - val_accuracy: 0.8318
Epoch 15/50
259/259 [==============================] - 90s 346ms/step - loss: 0.2084 - accuracy: 0.917
1 - val_loss: 0.2490 - val_accuracy: 0.8963
Epoch 16/50
259/259 [==============================] - 81s 314ms/step - loss: 0.1777 - accuracy: 0.925
8 - val_loss: 0.3263 - val_accuracy: 0.8846
Epoch 17/50
259/259 [==============================] - 80s 310ms/step - loss: 0.1683 - accuracy: 0.932
7 - val_loss: 0.2495 - val_accuracy: 0.9031
```

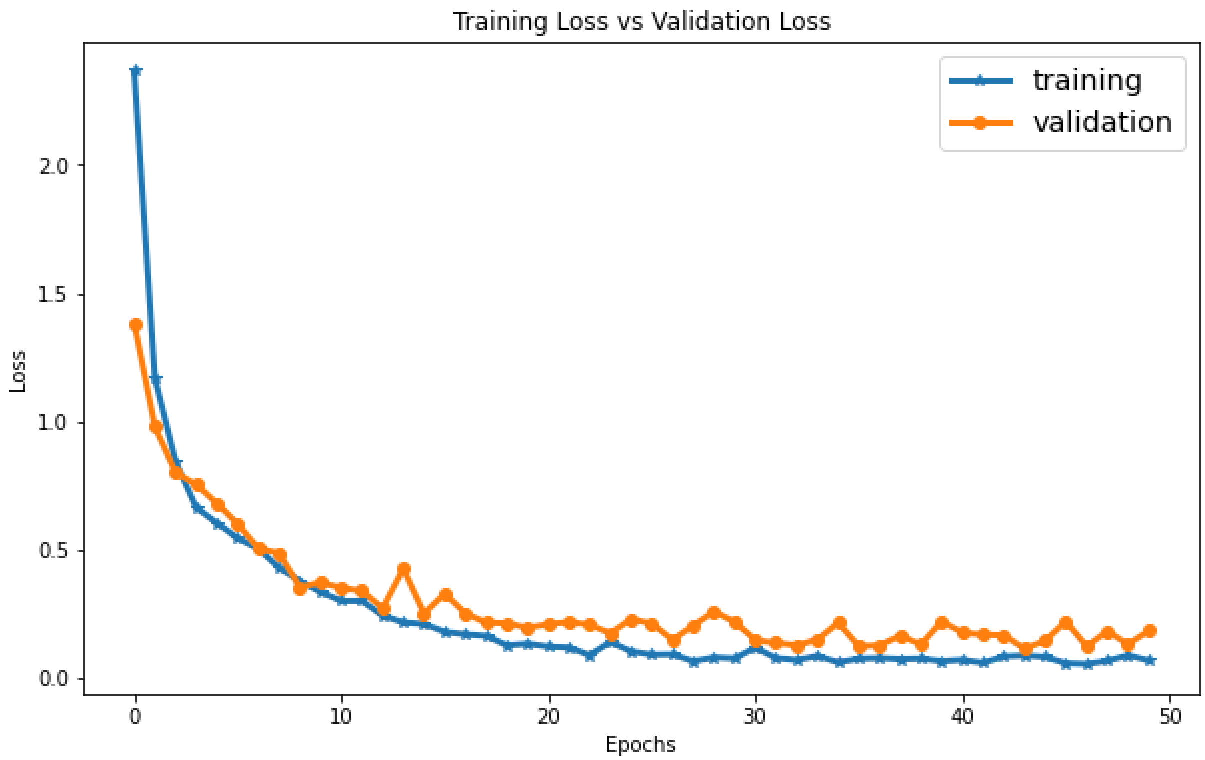
```
Epoch 18/50
259/259 [==============================] - 80s 307ms/step - loss: 0.1621 - accuracy: 0.938
2 - val_loss: 0.2136 - val_accuracy: 0.9195
Epoch 19/50
259/259 [==============================] - 83s 320ms/step - loss: 0.1252 - accuracy: 0.950
4 - val_loss: 0.2114 - val_accuracy: 0.9200
Epoch 20/50
259/259 [==============================] - 91s 352ms/step - loss: 0.1314 - accuracy: 0.950
1 - val_loss: 0.1935 - val_accuracy: 0.9350
Epoch 21/50
259/259 [==============================] - 91s 350ms/step - loss: 0.1202 - accuracy: 0.955
7 - val_loss: 0.2078 - val_accuracy: 0.9283
Epoch 22/50
259/259 [==============================] - 90s 347ms/step - loss: 0.1167 - accuracy: 0.958
6 - val_loss: 0.2159 - val_accuracy: 0.9249
Epoch 23/50
259/259 [==============================] - 91s 351ms/step - loss: 0.0870 - accuracy: 0.967
5 - val_loss: 0.2058 - val_accuracy: 0.9263
Epoch 24/50
259/259 [==============================] - 91s 351ms/step - loss: 0.1383 - accuracy: 0.947
9 - val_loss: 0.1716 - val_accuracy: 0.9350
Epoch 25/50
259/259 [==============================] - 90s 348ms/step - loss: 0.1013 - accuracy: 0.964
1 - val_loss: 0.2273 - val_accuracy: 0.9195
Epoch 26/50
259/259 [==============================] - 90s 348ms/step - loss: 0.0888 - accuracy: 0.968
2 - val_loss: 0.2102 - val_accuracy: 0.9326
Epoch 27/50
259/259 [==============================] - 90s 348ms/step - loss: 0.0899 - accuracy: 0.967
1 - val_loss: 0.1463 - val_accuracy: 0.9510
Epoch 28/50
259/259 [==============================] - 90s 348ms/step - loss: 0.0629 - accuracy: 0.976
3 - val_loss: 0.2018 - val_accuracy: 0.9346
Epoch 29/50
259/259 [==============================] - 90s 349ms/step - loss: 0.0779 - accuracy: 0.973
1 - val_loss: 0.2564 - val_accuracy: 0.9113
Epoch 30/50
259/259 [==============================] - 91s 350ms/step - loss: 0.0740 - accuracy: 0.974
5 - val_loss: 0.2166 - val_accuracy: 0.9380
Epoch 31/50
259/259 [==============================] - 90s 346ms/step - loss: 0.1159 - accuracy: 0.959
7 - val_loss: 0.1457 - val_accuracy: 0.9515
Epoch 32/50
259/259 [==============================] - 91s 349ms/step - loss: 0.0753 - accuracy: 0.974
5 - val_loss: 0.1342 - val_accuracy: 0.9578
Epoch 33/50
259/259 [==============================] - 89s 344ms/step - loss: 0.0680 - accuracy: 0.977
2 - val_loss: 0.1250 - val_accuracy: 0.9540
Epoch 34/50
259/259 [==============================] - 89s 343ms/step - loss: 0.0834 - accuracy: 0.971
1 - val_loss: 0.1474 - val_accuracy: 0.9496
Epoch 35/50
259/259 [==============================] - 91s 352ms/step - loss: 0.0589 - accuracy: 0.979
4 - val_loss: 0.2127 - val_accuracy: 0.9258
Epoch 36/50
259/259 [==============================] - 92s 354ms/step - loss: 0.0729 - accuracy: 0.975
2 - val_loss: 0.1212 - val_accuracy: 0.9549
Epoch 37/50
259/259 [==============================] - 91s 349ms/step - loss: 0.0754 - accuracy: 0.974
9 - val_loss: 0.1256 - val_accuracy: 0.9632
Epoch 38/50
259/259 [==============================] - 90s 348ms/step - loss: 0.0713 - accuracy: 0.976
3 - val_loss: 0.1621 - val_accuracy: 0.9540
Epoch 39/50
259/259 [==============================] - 91s 350ms/step - loss: 0.0734 - accuracy: 0.974
3 - val_loss: 0.1288 - val_accuracy: 0.9525
```

```
Epoch 40/50
259/259 [==============================] - 90s 348ms/step - loss: 0.0632 - accuracy: 0.979
7 - val_loss: 0.2183 - val_accuracy: 0.9297
Epoch 41/50
259/259 [==============================] - 92s 353ms/step - loss: 0.0684 - accuracy: 0.977
3 - val_loss: 0.1748 - val_accuracy: 0.9423
Epoch 42/50
259/259 [==============================] - 90s 349ms/step - loss: 0.0566 - accuracy: 0.982
3 - val_loss: 0.1667 - val_accuracy: 0.9472
Epoch 43/50
259/259 [==============================] - 91s 349ms/step - loss: 0.0818 - accuracy: 0.972
7 - val_loss: 0.1647 - val_accuracy: 0.9496
Epoch 44/50
259/259 [==============================] - 91s 350ms/step - loss: 0.0853 - accuracy: 0.973
4 - val_loss: 0.1121 - val_accuracy: 0.9583
Epoch 45/50
259/259 [==============================] - 90s 349ms/step - loss: 0.0797 - accuracy: 0.974
8 - val_loss: 0.1428 - val_accuracy: 0.9501
Epoch 46/50
259/259 [==============================] - 90s 349ms/step - loss: 0.0551 - accuracy: 0.982
3 - val_loss: 0.2184 - val_accuracy: 0.9486
Epoch 47/50
259/259 [==============================] - 91s 349ms/step - loss: 0.0523 - accuracy: 0.983
4 - val_loss: 0.1210 - val_accuracy: 0.9661
Epoch 48/50
259/259 [==============================] - 91s 350ms/step - loss: 0.0644 - accuracy: 0.979
5 - val_loss: 0.1800 - val_accuracy: 0.9399
Epoch 49/50
259/259 [==============================] - 90s 348ms/step - loss: 0.0862 - accuracy: 0.971
3 - val_loss: 0.1294 - val_accuracy: 0.9656
Epoch 50/50
259/259 [==============================] - 90s 348ms/step - loss: 0.0657 - accuracy: 0.978
1 - val_loss: 0.1814 - val_accuracy: 0.9447
```

In [55]:
```
vgg_model_s = 'vgg_train.h5'
vgg_model.save(vgg_model_s)
```

In [56]:
```
visualize_training(vgg_history)
```





In [57]:

```
accuracy = vgg_history.history['accuracy']
loss = vgg_history.history['loss']
val_accuracy = vgg_history.history['val_accuracy']
val_loss = vgg_history.history['val_loss']

print(f'Training Accuracy: {np.max(accuracy)}')
print(f'Training Loss: {np.min(loss)}')
print(f'Validation Accuracy: {np.max(val_accuracy)}')
print(f'Validation Loss: {np.min(val_loss)}')
```

```
Training Accuracy: 0.9834110140800476
Training Loss: 0.05231398344039917
Validation Accuracy: 0.9660688042640686
Validation Loss: 0.11205992847681046
```

VGG19 Model is also performing incredibly well!

Inception Model

In [58]:

```
inc_model = Sequential()
inc_model.add(InceptionV3(include_top=False, weights='imagenet', input_shape=(128, 128, 3)
inc_model.add(Flatten())
inc_model.add(Dense(64,activation='relu'))
vgg_model.add(Dense(16,activation='relu'))
inc_model.add(Dense(4,activation = 'softmax'))

opt = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.7, beta_2=0.8)
inc_model.compile(optimizer= opt, loss='sparse_categorical_crossentropy', metrics=['accura
inc_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 2, 2, 2048)        21802784

 flatten_3 (Flatten)         (None, 8192)              0
```

```
 dense_7 (Dense)             (None, 64)                524352

 dense_9 (Dense)             (None, 4)                 260

=================================================================
Total params: 22,327,396
Trainable params: 22,292,964
Non-trainable params: 34,432
_________________________________________________________________
```

In [59]:

```
inc_history = inc_model.fit(train_data,epochs=50,validation_data = val_data,verbose=1)
```

```
Epoch 1/50
259/259 [==============================] - 57s 189ms/step - loss: 0.5803 - accuracy: 0.761
2 - val_loss: 80.8907 - val_accuracy: 0.3296
Epoch 2/50
259/259 [==============================] - 47s 180ms/step - loss: 0.2359 - accuracy: 0.916
0 - val_loss: 1.0237 - val_accuracy: 0.7179
Epoch 3/50
259/259 [==============================] - 48s 185ms/step - loss: 0.1433 - accuracy: 0.952
7 - val_loss: 17.8383 - val_accuracy: 0.7033
Epoch 4/50
259/259 [==============================] - 47s 181ms/step - loss: 0.1151 - accuracy: 0.967
2 - val_loss: 1.9834 - val_accuracy: 0.7222
Epoch 5/50
259/259 [==============================] - 47s 179ms/step - loss: 0.0945 - accuracy: 0.973
9 - val_loss: 0.7402 - val_accuracy: 0.9132
Epoch 6/50
259/259 [==============================] - 47s 182ms/step - loss: 0.0650 - accuracy: 0.981
4 - val_loss: 1.3651 - val_accuracy: 0.7998
Epoch 7/50
259/259 [==============================] - 47s 180ms/step - loss: 0.0750 - accuracy: 0.980
2 - val_loss: 0.6066 - val_accuracy: 0.8943
Epoch 8/50
259/259 [==============================] - 45s 174ms/step - loss: 0.0510 - accuracy: 0.984
7 - val_loss: 43.4049 - val_accuracy: 0.7901
Epoch 9/50
259/259 [==============================] - 45s 174ms/step - loss: 0.0481 - accuracy: 0.988
3 - val_loss: 0.9399 - val_accuracy: 0.8769
Epoch 10/50
259/259 [==============================] - 45s 174ms/step - loss: 0.0820 - accuracy: 0.985
5 - val_loss: 31.8999 - val_accuracy: 0.8032
Epoch 11/50
259/259 [==============================] - 45s 174ms/step - loss: 0.0354 - accuracy: 0.990
0 - val_loss: 0.6037 - val_accuracy: 0.9132
Epoch 12/50
259/259 [==============================] - 45s 174ms/step - loss: 0.0476 - accuracy: 0.990
5 - val_loss: 2.0675 - val_accuracy: 0.7077
Epoch 13/50
259/259 [==============================] - 45s 174ms/step - loss: 0.0347 - accuracy: 0.991
0 - val_loss: 0.2643 - val_accuracy: 0.9476
Epoch 14/50
259/259 [==============================] - 45s 175ms/step - loss: 0.0424 - accuracy: 0.991
6 - val_loss: 0.2433 - val_accuracy: 0.9287
Epoch 15/50
259/259 [==============================] - 45s 174ms/step - loss: 0.0384 - accuracy: 0.992
6 - val_loss: 0.1259 - val_accuracy: 0.9840
Epoch 16/50
259/259 [==============================] - 45s 174ms/step - loss: 0.0322 - accuracy: 0.992
3 - val_loss: 0.7194 - val_accuracy: 0.9433
Epoch 17/50
259/259 [==============================] - 45s 174ms/step - loss: 0.0288 - accuracy: 0.993
0 - val_loss: 2.5935 - val_accuracy: 0.7998
Epoch 18/50
259/259 [==============================] - 46s 175ms/step - loss: 0.0312 - accuracy: 0.993
```

```
5 - val_loss: 1.3744 - val_accuracy: 0.9743
Epoch 19/50
259/259 [==============================] - 47s 181ms/step - loss: 0.0226 - accuracy: 0.994
8 - val_loss: 2.6346 - val_accuracy: 0.8294
Epoch 20/50
259/259 [==============================] - 47s 182ms/step - loss: 0.0259 - accuracy: 0.994
5 - val_loss: 14.0314 - val_accuracy: 0.9404
Epoch 21/50
259/259 [==============================] - 47s 183ms/step - loss: 0.0218 - accuracy: 0.994
9 - val_loss: 0.6923 - val_accuracy: 0.9724
Epoch 22/50
259/259 [==============================] - 48s 184ms/step - loss: 0.0406 - accuracy: 0.994
7 - val_loss: 2.7265 - val_accuracy: 0.8754
Epoch 23/50
259/259 [==============================] - 48s 184ms/step - loss: 0.0236 - accuracy: 0.994
9 - val_loss: 1.2923 - val_accuracy: 0.8812
Epoch 24/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0301 - accuracy: 0.993
6 - val_loss: 0.1586 - val_accuracy: 0.9661
Epoch 25/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0193 - accuracy: 0.996
2 - val_loss: 0.3511 - val_accuracy: 0.9491
Epoch 26/50
259/259 [==============================] - 47s 182ms/step - loss: 0.0211 - accuracy: 0.996
4 - val_loss: 0.3459 - val_accuracy: 0.9375
Epoch 27/50
259/259 [==============================] - 47s 183ms/step - loss: 0.0214 - accuracy: 0.994
7 - val_loss: 0.3236 - val_accuracy: 0.9331
Epoch 28/50
259/259 [==============================] - 47s 182ms/step - loss: 0.0273 - accuracy: 0.997
1 - val_loss: 0.1798 - val_accuracy: 0.9767
Epoch 29/50
259/259 [==============================] - 47s 181ms/step - loss: 0.0357 - accuracy: 0.995
5 - val_loss: 0.1930 - val_accuracy: 0.9729
Epoch 30/50
259/259 [==============================] - 47s 182ms/step - loss: 0.0241 - accuracy: 0.996
7 - val_loss: 0.1986 - val_accuracy: 0.9690
Epoch 31/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0176 - accuracy: 0.996
5 - val_loss: 0.1004 - val_accuracy: 0.9879
Epoch 32/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0201 - accuracy: 0.996
1 - val_loss: 0.0823 - val_accuracy: 0.9801
Epoch 33/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0196 - accuracy: 0.996
9 - val_loss: 0.0669 - val_accuracy: 0.9855
Epoch 34/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0284 - accuracy: 0.996
5 - val_loss: 0.2562 - val_accuracy: 0.9510
Epoch 35/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0161 - accuracy: 0.996
9 - val_loss: 0.0800 - val_accuracy: 0.9874
Epoch 36/50
259/259 [==============================] - 48s 186ms/step - loss: 0.0128 - accuracy: 0.997
0 - val_loss: 0.2091 - val_accuracy: 0.9564
Epoch 37/50
259/259 [==============================] - 49s 187ms/step - loss: 0.0440 - accuracy: 0.997
1 - val_loss: 0.1369 - val_accuracy: 0.9811
Epoch 38/50
259/259 [==============================] - 47s 182ms/step - loss: 0.0347 - accuracy: 0.996
5 - val_loss: 0.4440 - val_accuracy: 0.9627
Epoch 39/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0178 - accuracy: 0.996
5 - val_loss: 0.1199 - val_accuracy: 0.9811
Epoch 40/50
259/259 [==============================] - 48s 184ms/step - loss: 0.0240 - accuracy: 0.995
```

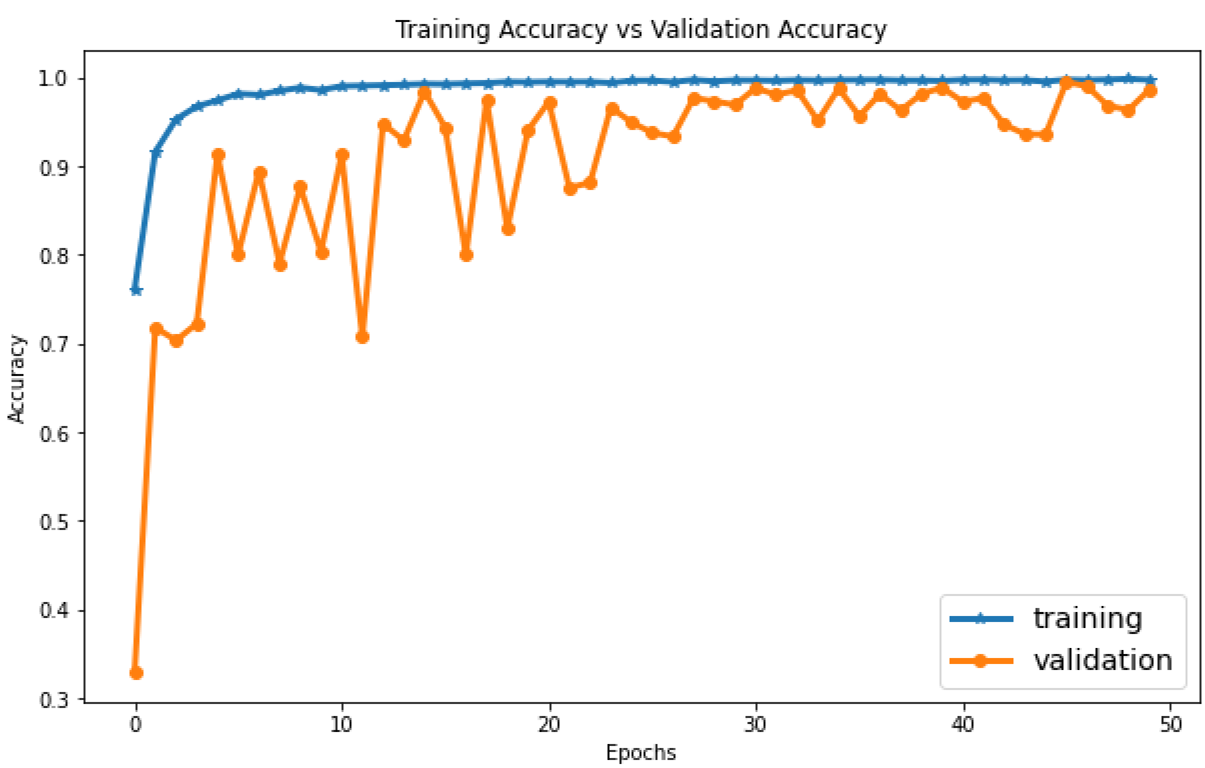
```
9 - val_loss: 0.0571 - val_accuracy: 0.9884
Epoch 41/50
259/259 [==============================] - 47s 183ms/step - loss: 0.0154 - accuracy: 0.997
2 - val_loss: 1.1128 - val_accuracy: 0.9724
Epoch 42/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0211 - accuracy: 0.997
3 - val_loss: 0.2118 - val_accuracy: 0.9767
Epoch 43/50
259/259 [==============================] - 48s 184ms/step - loss: 0.0462 - accuracy: 0.996
5 - val_loss: 0.2824 - val_accuracy: 0.9467
Epoch 44/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0354 - accuracy: 0.996
7 - val_loss: 0.6224 - val_accuracy: 0.9360
Epoch 45/50
259/259 [==============================] - 48s 184ms/step - loss: 0.0255 - accuracy: 0.995
2 - val_loss: 2.2292 - val_accuracy: 0.9355
Epoch 46/50
259/259 [==============================] - 48s 184ms/step - loss: 0.0415 - accuracy: 0.997
3 - val_loss: 0.0289 - val_accuracy: 0.9947
Epoch 47/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0225 - accuracy: 0.996
7 - val_loss: 0.0406 - val_accuracy: 0.9898
Epoch 48/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0143 - accuracy: 0.997
5 - val_loss: 0.1829 - val_accuracy: 0.9680
Epoch 49/50
259/259 [==============================] - 47s 183ms/step - loss: 0.0093 - accuracy: 0.998
1 - val_loss: 3.1791 - val_accuracy: 0.9632
Epoch 50/50
259/259 [==============================] - 48s 183ms/step - loss: 0.0167 - accuracy: 0.997
1 - val_loss: 1.8097 - val_accuracy: 0.9859
```

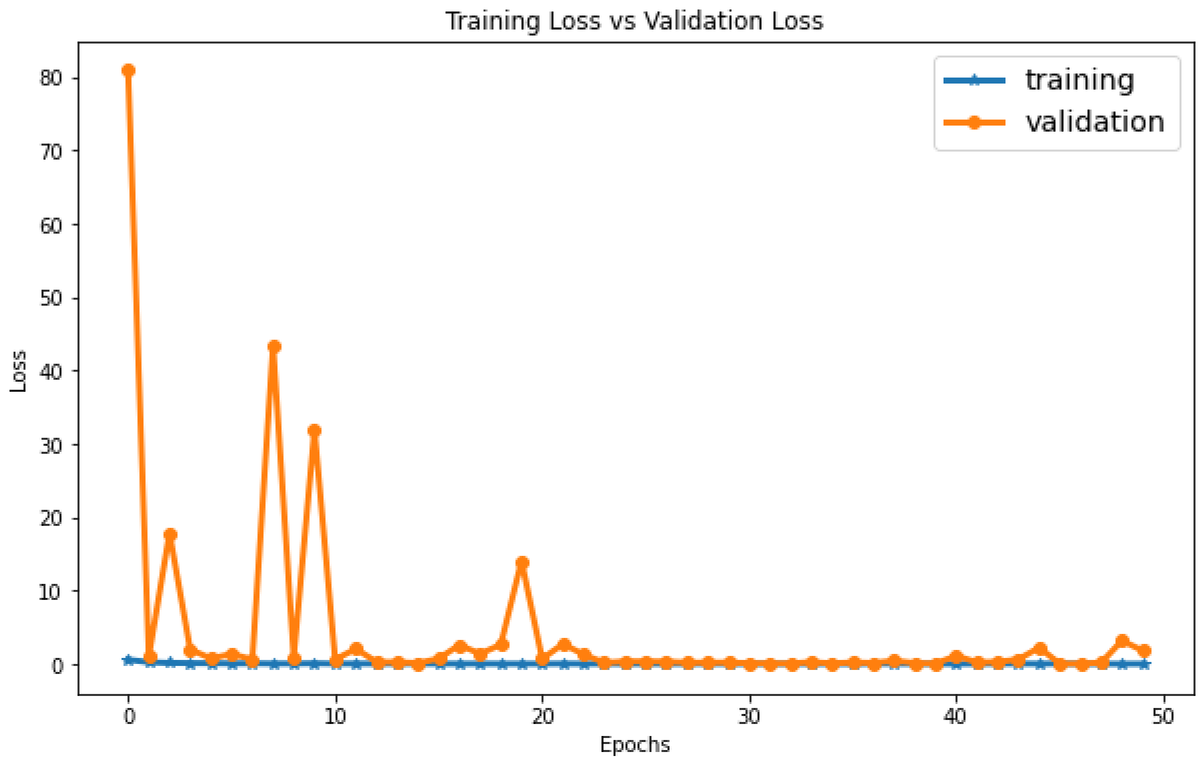
In [63]:

```
inc_model_s = 'inceptionv3_train.h5'
inc_model.save(inc_model_s)
```

In [70]:

```
visualize_training(inc_history)
```





In [65]:

```
accuracy = inc_history.history['accuracy']
loss = inc_history.history['loss']
val_accuracy = inc_history.history['val_accuracy']
val_loss = inc_history.history['val_loss']

print(f'Training Accuracy: {np.max(accuracy)}')
print(f'Training Loss: {np.min(loss)}')
print(f'Validation Accuracy: {np.max(val_accuracy)}')
print(f'Validation Loss: {np.min(val_loss)}')
```

```
Training Accuracy: 0.9981231689453125
Training Loss: 0.009340698830783367
Validation Accuracy: 0.9946679472923279
Validation Loss: 0.028861040249466896
```

Looks good!

Evaluating Models on Test Dataset

After using model.evaluate function I am looping through some samples from the testing dataset and vizualizing them and checking on accuracy of model

Conv2D Model 1 evaluation

In [71]:

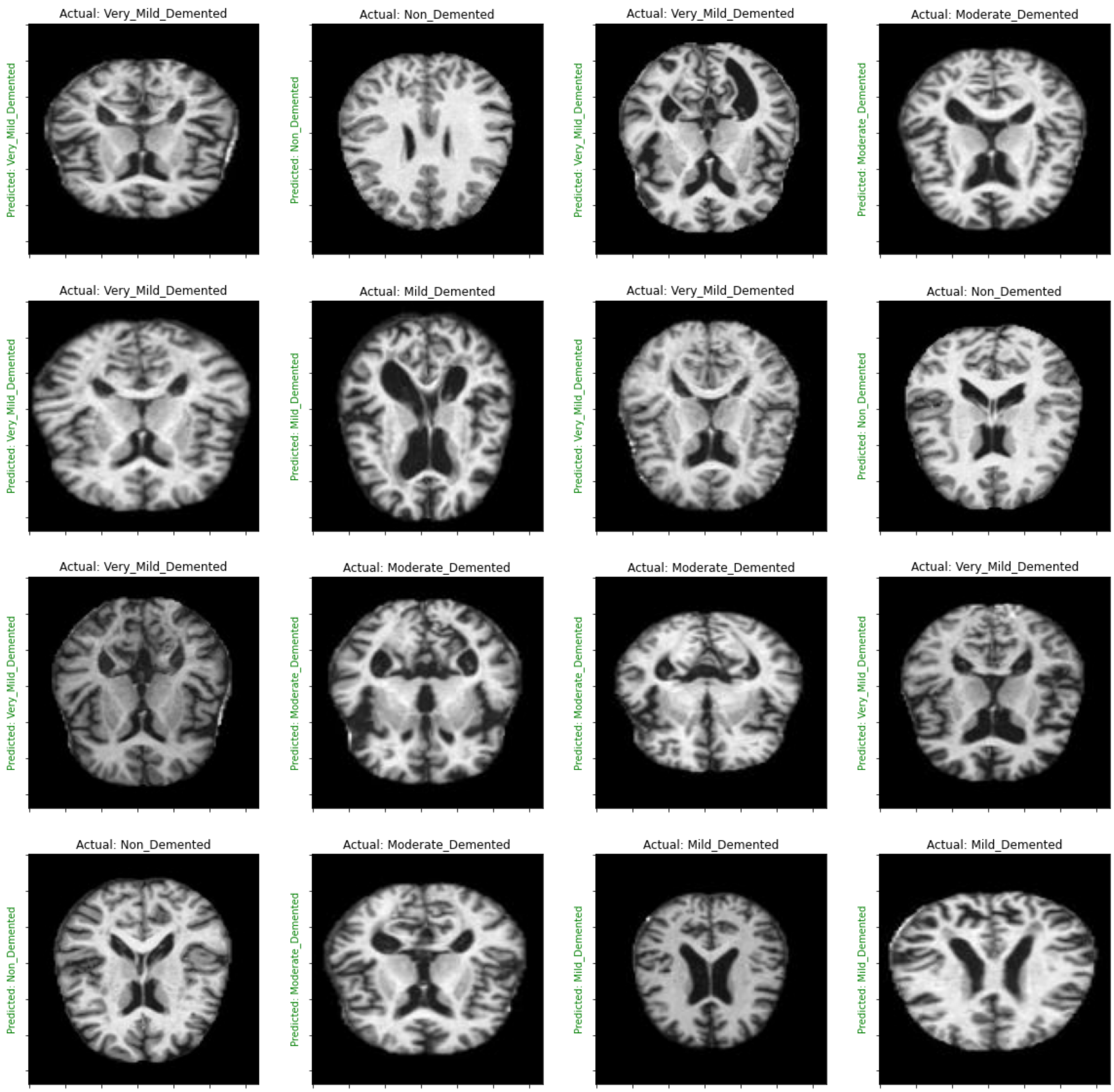
```
loss, accuracy = model_1.evaluate(test_data)
```

```
33/33 [==============================] - 2s 56ms/step - loss: 0.2864 - accuracy: 0.8884
```

In [72]:

```
plt.figure(figsize=(20, 20))
for images, labels in test_data.take(1):
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        predictions = model_1.predict(tf.expand_dims(images[i], 0))
        score = tf.nn.softmax(predictions[0])
        if(classes[labels[i]]==classes[np.argmax(score)]):
            plt.title("Actual: "+classes[labels[i]])
```

```
            plt.ylabel("Predicted: "+classes[np.argmax(score)],fontdict={'color':'green'})

        else:
            plt.title("Actual: "+classes[labels[i]])
            plt.ylabel("Predicted: "+classes[np.argmax(score)],fontdict={'color':'red'})
        plt.gca().axes.yaxis.set_ticklabels([])
        plt.gca().axes.xaxis.set_ticklabels([])
```



Conv2D Model 2 Evaluation

In [73]:

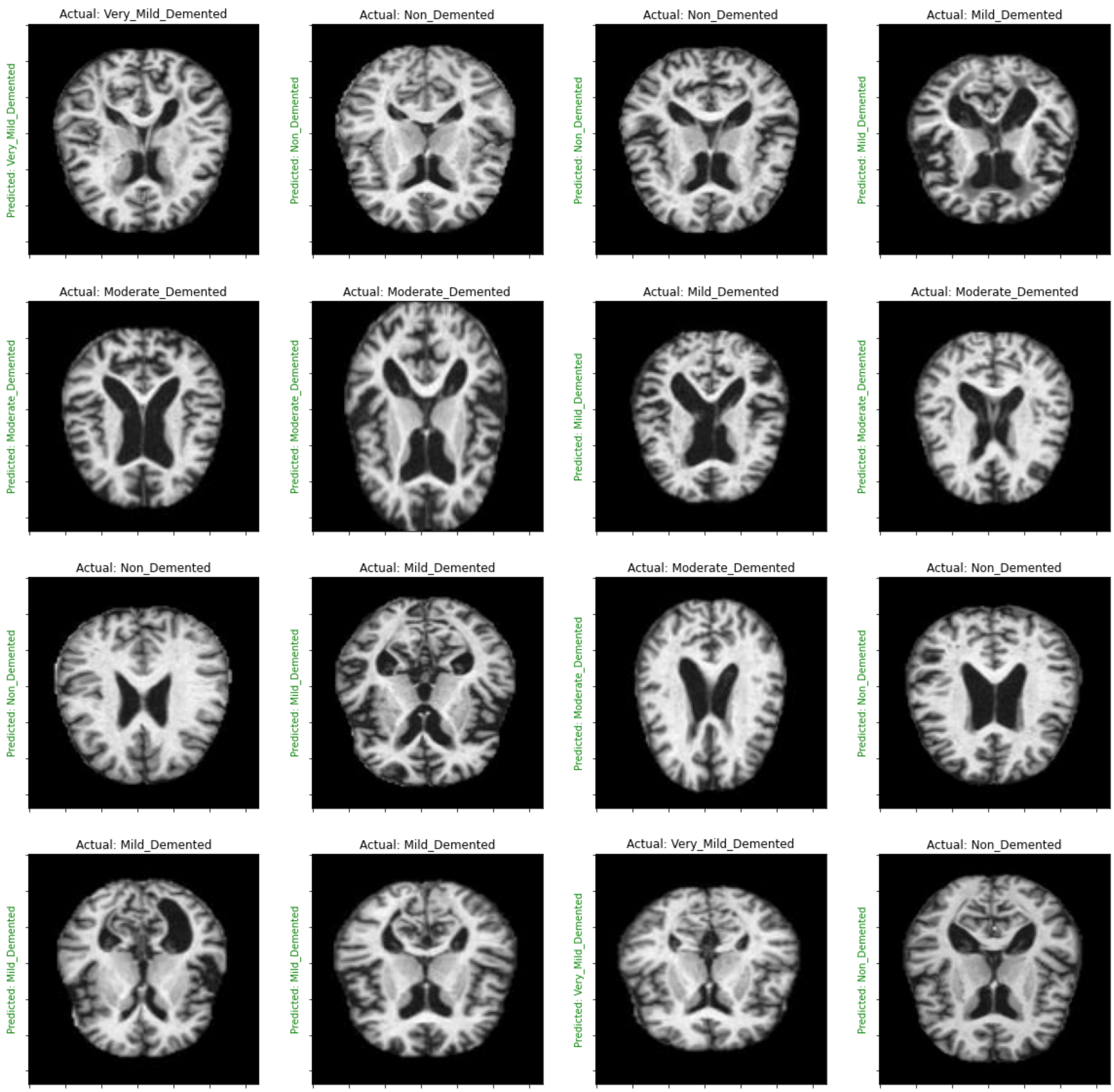
```
loss, accuracy = model_2.evaluate(test_data)
```

```
33/33 [==============================] - 1s 30ms/step - loss: 0.0313 - accuracy: 0.9918
```

In [113…

```
plt.figure(figsize=(20, 20))
for images, labels in test_data.take(1):
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
```

```
        predictions = model_2.predict(tf.expand_dims(images[i], 0))
        score = tf.nn.softmax(predictions[0])
        if(classes[labels[i]]==classes[np.argmax(score)]):
            plt.title("Actual: "+classes[labels[i]])
            plt.ylabel("Predicted: "+classes[np.argmax(score)],fontdict={'color':'green'})

        else:
            plt.title("Actual: "+classes[labels[i]])
            plt.ylabel("Predicted: "+classes[np.argmax(score)],fontdict={'color':'red'})
        plt.gca().axes.yaxis.set_ticklabels([])
        plt.gca().axes.xaxis.set_ticklabels([])
```



VGG19 Model evaluation

In [98]:

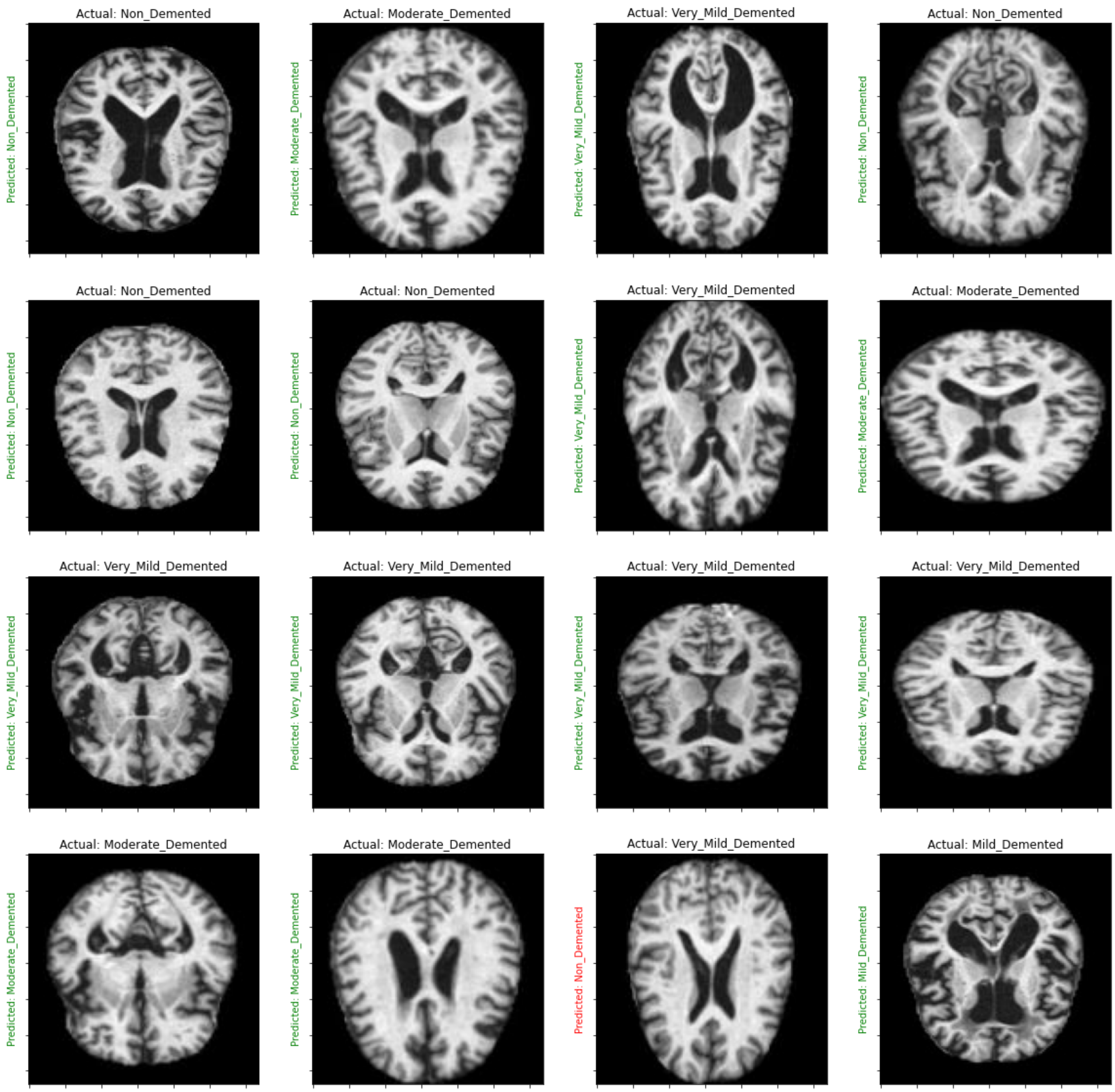
```
loss, accuracy = vgg_model.evaluate(test_data)
```

```
33/33 [==============================] - 4s 122ms/step - loss: 0.1648 - accuracy: 0.9483
```

In [131…

```
plt.figure(figsize=(20, 20))
```

```
for images, labels in test_data.take(1):
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        predictions = vgg_model.predict(tf.expand_dims(images[i], 0))
        score = tf.nn.softmax(predictions[0])
        if(classes[labels[i]]==classes[np.argmax(score)]):
            plt.title("Actual: "+classes[labels[i]])
            plt.ylabel("Predicted: "+classes[np.argmax(score)],fontdict={'color':'green'})

        else:
            plt.title("Actual: "+classes[labels[i]])
            plt.ylabel("Predicted: "+classes[np.argmax(score)],fontdict={'color':'red'})
        plt.gca().axes.yaxis.set_ticklabels([])
        plt.gca().axes.xaxis.set_ticklabels([])
```



Inception Model evaluation

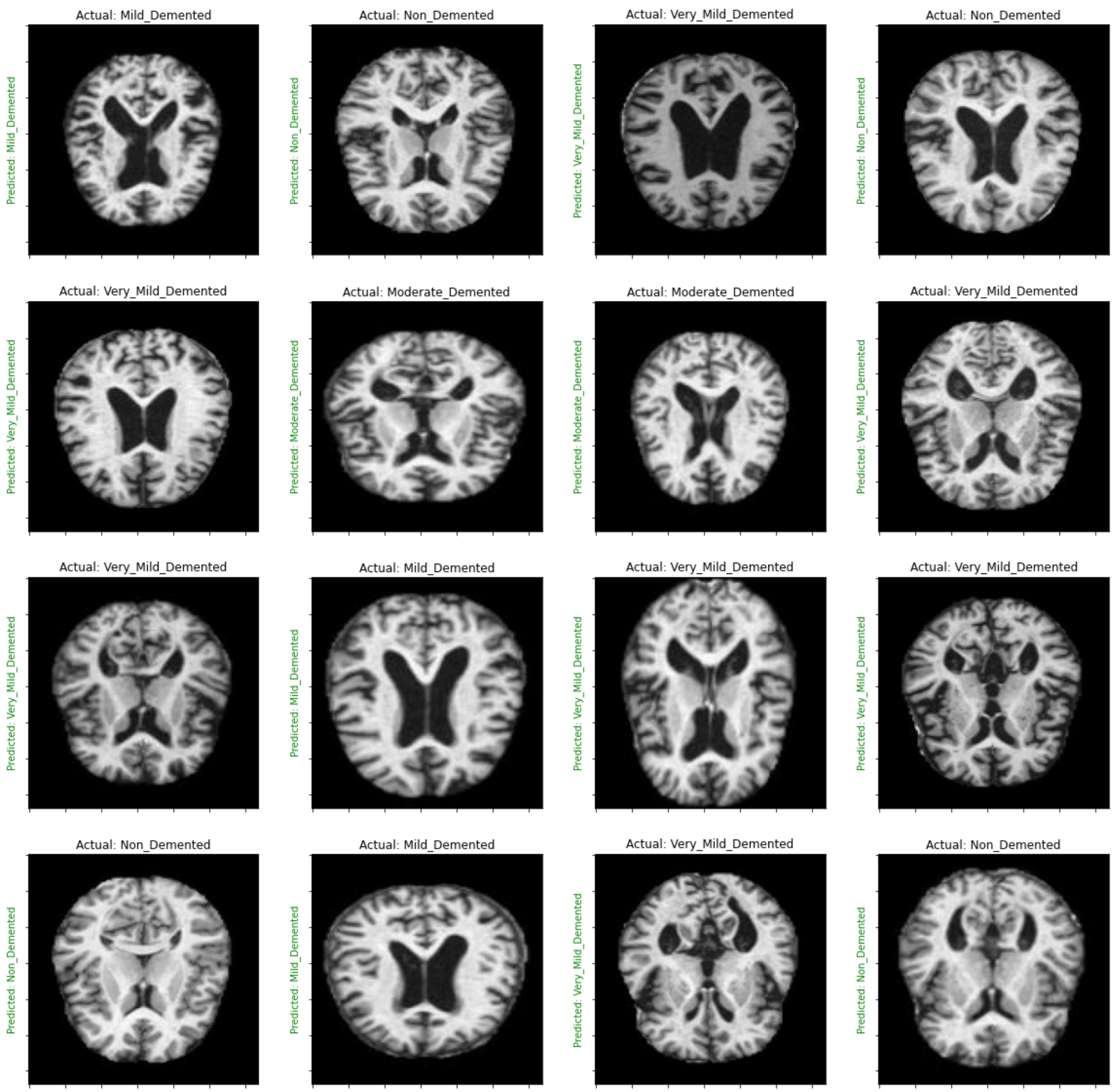
In [77]:

```
loss, accuracy = inc_model.evaluate(test_data)
```

```
33/33 [==============================] - 3s 71ms/step - loss: 1.8475 - accuracy: 0.9865
```

In [78]:

```python
plt.figure(figsize=(20, 20))
for images, labels in test_data.take(1):
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        predictions = inc_model.predict(tf.expand_dims(images[i], 0))
        score = tf.nn.softmax(predictions[0])
        if(classes[labels[i]]==classes[np.argmax(score)]):
            plt.title("Actual: "+classes[labels[i]])
            plt.ylabel("Predicted: "+classes[np.argmax(score)],fontdict={'color':'green'})

        else:
            plt.title("Actual: "+classes[labels[i]])
            plt.ylabel("Predicted: "+classes[np.argmax(score)],fontdict={'color':'red'})
        plt.gca().axes.yaxis.set_ticklabels([])
        plt.gca().axes.xaxis.set_ticklabels([])
```



In [ ]:

```
In [ ]:
```