In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
ount("/content/drive", force_remount=True).
```

In [1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,Dropout,Globa
#   DataGenerator to read images and rescale images
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from collections import Counter
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
```

In [2]:
```
# Create EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=Tr
```

In [3]:
```
## Set Path Here before running the code
WORKING_DIRECTORY =  "C:/Users/T PRASOONA LEHARI/Desktop/FINAL_MP_DATASET"

##  Name of classes
CLASSES = ['Mild_Demented',
           'Moderate_Demented',
           'Non_Demented',
           'Very_Mild_Demented']
```

In [4]:
```
X, y = [], []

## Images rescaling
datagen = ImageDataGenerator(rescale=1.0/255.0)

#   Load images by resizing and shuffling randomly
train_dataset = datagen.flow_from_directory(WORKING_DIRECTORY, target_size=(150, 150),

### Seperate Dataset from  Data Genrator
X, y = train_dataset.next()
```

```
Found 1000 images belonging to 4 classes.
```

In [5]:
```
from sklearn.model_selection import train_test_split
from collections import Counter
import numpy as np

# Stratified split into training (80%) and rest (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, stratify=y, r

# Stratified split the rest into validation (50% of the remaining) and testing (50% of
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratif

# Print number of samples after train test split
print("Number of samples after splitting into Training, validation & test set\n")
print("Train     \t", sorted(Counter(np.argmax(y_train, axis=1)).items()))
print("Validation\t", sorted(Counter(np.argmax(y_val, axis=1)).items()))
print("Test      \t", sorted(Counter(np.argmax(y_test, axis=1)).items()))
```

```
Number of samples after splitting into Training, validation & test set

Train           [(0, 200), (1, 200), (2, 200), (3, 200)]
Validation      [(0, 25), (1, 25), (2, 25), (3, 25)]
Test            [(0, 25), (1, 25), (2, 25), (3, 25)]
```

In [6]:
```
print("Number of samples in each class:\t", sorted(Counter(np.argmax(y, axis=1)).items

#   class labels as per indices
print("Classes Names according to index:\t", train_dataset.class_indices)
```

```
Number of samples in each class:        [(0, 250), (1, 250), (2, 250), (3, 250)]
Classes Names according to index:       {'MildDemented': 0, 'ModerateDemented': 1,
'NonDemented': 2, 'VeryMildDemented': 3}
```

In [7]:
```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (800, 150, 150, 3)
y_train shape: (800, 4)
X_test shape: (100, 150, 150, 3)
y_test shape: (100, 4)
```

In [10]:
```
from tensorflow.keras.applications import EfficientNetB0

# Create EfficientNetB0 model
model_efficientnet = EfficientNetB0(input_shape=(150, 150, 3), weights=None, classes=4

# Compile the model
model_efficientnet.compile(optimizer='adam', loss='categorical_crossentropy', metrics=

# Training with EarlyStopping
history_efficientnet = model_efficientnet.fit(X_train, y_train, epochs=100, validatior
```

```
Epoch 1/100
25/25 [==============================] - 255s 6s/step - loss: 3.6025 - accuracy: 0.33
75 - val_loss: 1.6054 - val_accuracy: 0.2500
Epoch 2/100
25/25 [==============================] - 105s 4s/step - loss: 1.7630 - accuracy: 0.43
75 - val_loss: 1.6944 - val_accuracy: 0.2500
Epoch 3/100
25/25 [==============================] - 101s 4s/step - loss: 1.4045 - accuracy: 0.56
25 - val_loss: 2.2929 - val_accuracy: 0.2500
Epoch 4/100
25/25 [==============================] - 102s 4s/step - loss: 0.8472 - accuracy: 0.66
75 - val_loss: 2.5059 - val_accuracy: 0.2500
Epoch 5/100
25/25 [==============================] - 105s 4s/step - loss: 0.5908 - accuracy: 0.80
00 - val_loss: 1.4889 - val_accuracy: 0.2500
Epoch 6/100
25/25 [==============================] - 108s 4s/step - loss: 0.4355 - accuracy: 0.84
75 - val_loss: 2.0776 - val_accuracy: 0.2500
Epoch 7/100
25/25 [==============================] - 110s 4s/step - loss: 0.2700 - accuracy: 0.90
75 - val_loss: 2.3658 - val_accuracy: 0.2500
Epoch 8/100
25/25 [==============================] - 107s 4s/step - loss: 0.3024 - accuracy: 0.88
63 - val_loss: 1.8146 - val_accuracy: 0.2500
Epoch 9/100
25/25 [==============================] - 103s 4s/step - loss: 0.2763 - accuracy: 0.91
75 - val_loss: 4.2797 - val_accuracy: 0.2500
Epoch 10/100
25/25 [==============================] - 102s 4s/step - loss: 0.2948 - accuracy: 0.90
13 - val_loss: 4.6288 - val_accuracy: 0.2500
Epoch 11/100
25/25 [==============================] - 105s 4s/step - loss: 0.1862 - accuracy: 0.94
00 - val_loss: 3.7134 - val_accuracy: 0.2500
Epoch 12/100
25/25 [==============================] - 102s 4s/step - loss: 0.1182 - accuracy: 0.96
00 - val_loss: 3.9432 - val_accuracy: 0.2500
Epoch 13/100
25/25 [==============================] - 103s 4s/step - loss: 0.0975 - accuracy: 0.96
00 - val_loss: 6.8504 - val_accuracy: 0.2500
Epoch 14/100
25/25 [==============================] - 102s 4s/step - loss: 0.1521 - accuracy: 0.96
13 - val_loss: 6.5677 - val_accuracy: 0.2500
Epoch 15/100
25/25 [==============================] - 102s 4s/step - loss: 0.1199 - accuracy: 0.95
88 - val_loss: 6.6761 - val_accuracy: 0.2500
Epoch 16/100
25/25 [==============================] - 104s 4s/step - loss: 0.1122 - accuracy: 0.96
50 - val_loss: 7.4007 - val_accuracy: 0.2500
Epoch 17/100
25/25 [==============================] - 104s 4s/step - loss: 0.4933 - accuracy: 0.92
75 - val_loss: 3.6293 - val_accuracy: 0.2500
Epoch 18/100
25/25 [==============================] - 102s 4s/step - loss: 0.2207 - accuracy: 0.93
50 - val_loss: 5.1064 - val_accuracy: 0.2500
Epoch 19/100
25/25 [==============================] - 103s 4s/step - loss: 0.1584 - accuracy: 0.94
25 - val_loss: 4.8188 - val_accuracy: 0.2500
Epoch 20/100
25/25 [==============================] - 102s 4s/step - loss: 0.3352 - accuracy: 0.93
62 - val_loss: 6.4989 - val_accuracy: 0.2500
```

```
Epoch 21/100
25/25 [==============================] - 101s 4s/step - loss: 0.0813 - accuracy: 0.96
88 - val_loss: 8.5131 - val_accuracy: 0.2500
Epoch 22/100
25/25 [==============================] - 101s 4s/step - loss: 0.0817 - accuracy: 0.97
37 - val_loss: 4.9693 - val_accuracy: 0.2500
Epoch 23/100
25/25 [==============================] - 102s 4s/step - loss: 0.0564 - accuracy: 0.97
87 - val_loss: 6.6418 - val_accuracy: 0.2500
Epoch 24/100
25/25 [==============================] - 102s 4s/step - loss: 0.1827 - accuracy: 0.97
75 - val_loss: 3.5872 - val_accuracy: 0.2500
Epoch 25/100
25/25 [==============================] - 102s 4s/step - loss: 0.1746 - accuracy: 0.94
38 - val_loss: 1.1594 - val_accuracy: 0.4800
Epoch 26/100
25/25 [==============================] - 101s 4s/step - loss: 0.1060 - accuracy: 0.96
25 - val_loss: 1.3166 - val_accuracy: 0.5300
Epoch 27/100
25/25 [==============================] - 101s 4s/step - loss: 0.1201 - accuracy: 0.95
63 - val_loss: 1.5102 - val_accuracy: 0.4700
Epoch 28/100
25/25 [==============================] - 101s 4s/step - loss: 0.0468 - accuracy: 0.98
62 - val_loss: 1.3648 - val_accuracy: 0.5800
Epoch 29/100
25/25 [==============================] - 101s 4s/step - loss: 0.0425 - accuracy: 0.98
50 - val_loss: 1.2343 - val_accuracy: 0.6700
Epoch 30/100
25/25 [==============================] - 145s 6s/step - loss: 0.0539 - accuracy: 0.98
37 - val_loss: 0.8412 - val_accuracy: 0.7700
Epoch 31/100
25/25 [==============================] - 151s 6s/step - loss: 0.0391 - accuracy: 0.98
62 - val_loss: 0.8385 - val_accuracy: 0.7600
Epoch 32/100
25/25 [==============================] - 150s 6s/step - loss: 0.0378 - accuracy: 0.98
87 - val_loss: 0.8503 - val_accuracy: 0.7600
Epoch 33/100
25/25 [==============================] - 152s 6s/step - loss: 0.0776 - accuracy: 0.98
00 - val_loss: 0.3530 - val_accuracy: 0.8800
Epoch 34/100
25/25 [==============================] - 151s 6s/step - loss: 0.0642 - accuracy: 0.97
75 - val_loss: 0.2184 - val_accuracy: 0.9100
Epoch 35/100
25/25 [==============================] - 152s 6s/step - loss: 0.0372 - accuracy: 0.98
37 - val_loss: 0.2083 - val_accuracy: 0.9300
Epoch 36/100
25/25 [==============================] - 151s 6s/step - loss: 0.0227 - accuracy: 0.99
25 - val_loss: 0.2337 - val_accuracy: 0.9100
Epoch 37/100
25/25 [==============================] - 151s 6s/step - loss: 0.0319 - accuracy: 0.98
75 - val_loss: 0.2496 - val_accuracy: 0.9300
Epoch 38/100
25/25 [==============================] - 151s 6s/step - loss: 0.0183 - accuracy: 0.99
25 - val_loss: 0.2788 - val_accuracy: 0.9400
Epoch 39/100
25/25 [==============================] - 152s 6s/step - loss: 0.0462 - accuracy: 0.99
00 - val_loss: 0.2382 - val_accuracy: 0.9300
Epoch 40/100
25/25 [==============================] - 152s 6s/step - loss: 0.0273 - accuracy: 0.99
00 - val_loss: 0.2876 - val_accuracy: 0.9200
```

```
Epoch 41/100
25/25 [==============================] - 152s 6s/step - loss: 0.0222 - accuracy: 0.99
50 - val_loss: 0.1403 - val_accuracy: 0.9500
Epoch 42/100
25/25 [==============================] - 151s 6s/step - loss: 0.0330 - accuracy: 0.99
25 - val_loss: 0.4418 - val_accuracy: 0.8900
Epoch 43/100
25/25 [==============================] - 152s 6s/step - loss: 0.0283 - accuracy: 0.99
00 - val_loss: 0.1967 - val_accuracy: 0.9300
Epoch 44/100
25/25 [==============================] - 152s 6s/step - loss: 0.0421 - accuracy: 0.98
50 - val_loss: 0.2820 - val_accuracy: 0.9300
Epoch 45/100
25/25 [==============================] - 151s 6s/step - loss: 0.2566 - accuracy: 0.92
25 - val_loss: 38.1932 - val_accuracy: 0.2500
Epoch 46/100
25/25 [==============================] - 152s 6s/step - loss: 0.4205 - accuracy: 0.84
87 - val_loss: 24.0730 - val_accuracy: 0.3800
Epoch 47/100
25/25 [==============================] - 152s 6s/step - loss: 0.1593 - accuracy: 0.94
63 - val_loss: 10.4500 - val_accuracy: 0.6300
Epoch 48/100
25/25 [==============================] - 152s 6s/step - loss: 0.0505 - accuracy: 0.98
12 - val_loss: 1.6804 - val_accuracy: 0.7200
Epoch 49/100
25/25 [==============================] - 153s 6s/step - loss: 0.0725 - accuracy: 0.97
62 - val_loss: 0.2889 - val_accuracy: 0.9000
Epoch 50/100
25/25 [==============================] - 151s 6s/step - loss: 0.0663 - accuracy: 0.98
12 - val_loss: 0.6249 - val_accuracy: 0.8200
Epoch 51/100
25/25 [==============================] - 152s 6s/step - loss: 0.0622 - accuracy: 0.98
25 - val_loss: 0.5557 - val_accuracy: 0.8400
Epoch 52/100
25/25 [==============================] - 151s 6s/step - loss: 0.0565 - accuracy: 0.98
00 - val_loss: 0.3060 - val_accuracy: 0.9100
Epoch 53/100
25/25 [==============================] - 151s 6s/step - loss: 0.0249 - accuracy: 0.99
12 - val_loss: 0.2574 - val_accuracy: 0.9300
Epoch 54/100
25/25 [==============================] - 151s 6s/step - loss: 0.0291 - accuracy: 0.98
75 - val_loss: 0.3442 - val_accuracy: 0.9000
Epoch 55/100
25/25 [==============================] - 152s 6s/step - loss: 0.0370 - accuracy: 0.99
12 - val_loss: 0.4761 - val_accuracy: 0.8800
Epoch 56/100
25/25 [==============================] - 151s 6s/step - loss: 0.0482 - accuracy: 0.98
75 - val_loss: 0.6211 - val_accuracy: 0.8600
Epoch 57/100
25/25 [==============================] - 152s 6s/step - loss: 0.0216 - accuracy: 0.99
25 - val_loss: 0.3517 - val_accuracy: 0.8700
Epoch 58/100
25/25 [==============================] - 151s 6s/step - loss: 0.0288 - accuracy: 0.99
50 - val_loss: 0.3377 - val_accuracy: 0.8800
Epoch 59/100
25/25 [==============================] - 151s 6s/step - loss: 0.0274 - accuracy: 0.99
37 - val_loss: 0.3338 - val_accuracy: 0.9200
Epoch 60/100
25/25 [==============================] - 152s 6s/step - loss: 0.0607 - accuracy: 0.97
87 - val_loss: 0.2479 - val_accuracy: 0.9100
```

```
Epoch 61/100
25/25 [==============================] - 156s 6s/step - loss: 0.1105 - accuracy: 0.97
25 - val_loss: 518.4313 - val_accuracy: 0.3000
Epoch 62/100
25/25 [==============================] - 158s 6s/step - loss: 0.0478 - accuracy: 0.98
12 - val_loss: 184.2026 - val_accuracy: 0.3500
Epoch 63/100
25/25 [==============================] - 152s 6s/step - loss: 0.0397 - accuracy: 0.98
37 - val_loss: 13.6298 - val_accuracy: 0.5600
Epoch 64/100
25/25 [==============================] - 152s 6s/step - loss: 0.0257 - accuracy: 0.99
12 - val_loss: 6.4045 - val_accuracy: 0.5600
Epoch 65/100
25/25 [==============================] - 152s 6s/step - loss: 0.0417 - accuracy: 0.98
50 - val_loss: 1.1004 - val_accuracy: 0.8100
Epoch 66/100
25/25 [==============================] - 152s 6s/step - loss: 0.0234 - accuracy: 0.98
87 - val_loss: 0.8128 - val_accuracy: 0.8800
Epoch 67/100
25/25 [==============================] - 152s 6s/step - loss: 0.0502 - accuracy: 0.98
62 - val_loss: 0.6485 - val_accuracy: 0.8200
Epoch 68/100
25/25 [==============================] - 152s 6s/step - loss: 0.0330 - accuracy: 0.98
62 - val_loss: 0.7050 - val_accuracy: 0.8500
Epoch 69/100
25/25 [==============================] - 152s 6s/step - loss: 0.0298 - accuracy: 0.99
25 - val_loss: 0.3717 - val_accuracy: 0.9200
Epoch 70/100
25/25 [==============================] - 151s 6s/step - loss: 0.0416 - accuracy: 0.98
62 - val_loss: 0.2410 - val_accuracy: 0.9400
Epoch 71/100
25/25 [==============================] - 152s 6s/step - loss: 0.0370 - accuracy: 0.98
75 - val_loss: 0.0976 - val_accuracy: 0.9700
Epoch 72/100
25/25 [==============================] - 152s 6s/step - loss: 0.0282 - accuracy: 0.99
12 - val_loss: 0.0547 - val_accuracy: 0.9800
Epoch 73/100
25/25 [==============================] - 151s 6s/step - loss: 0.0197 - accuracy: 0.98
87 - val_loss: 0.0627 - val_accuracy: 0.9600
Epoch 74/100
25/25 [==============================] - 151s 6s/step - loss: 0.0140 - accuracy: 0.99
37 - val_loss: 0.0654 - val_accuracy: 0.9600
Epoch 75/100
25/25 [==============================] - 152s 6s/step - loss: 0.0339 - accuracy: 0.99
25 - val_loss: 0.1079 - val_accuracy: 0.9700
Epoch 76/100
25/25 [==============================] - 152s 6s/step - loss: 0.0417 - accuracy: 0.99
12 - val_loss: 0.2740 - val_accuracy: 0.9300
Epoch 77/100
25/25 [==============================] - 155s 6s/step - loss: 0.0183 - accuracy: 0.99
25 - val_loss: 0.1385 - val_accuracy: 0.9500
Epoch 78/100
25/25 [==============================] - 151s 6s/step - loss: 0.0121 - accuracy: 0.99
50 - val_loss: 0.1296 - val_accuracy: 0.9400
Epoch 79/100
25/25 [==============================] - 152s 6s/step - loss: 0.0196 - accuracy: 0.99
62 - val_loss: 0.1215 - val_accuracy: 0.9400
Epoch 80/100
25/25 [==============================] - 152s 6s/step - loss: 0.0100 - accuracy: 0.99
62 - val_loss: 0.1243 - val_accuracy: 0.9600
```

```
Epoch 81/100
25/25 [==============================] - 152s 6s/step - loss: 0.0204 - accuracy: 0.99
62 - val_loss: 0.1470 - val_accuracy: 0.9400
Epoch 82/100
25/25 [==============================] - 151s 6s/step - loss: 0.0033 - accuracy: 1.00
00 - val_loss: 0.1665 - val_accuracy: 0.9500
Epoch 83/100
25/25 [==============================] - 151s 6s/step - loss: 0.0080 - accuracy: 0.99
62 - val_loss: 0.2545 - val_accuracy: 0.9200
Epoch 84/100
25/25 [==============================] - 152s 6s/step - loss: 0.0045 - accuracy: 0.99
87 - val_loss: 0.2450 - val_accuracy: 0.9400
Epoch 85/100
25/25 [==============================] - 150s 6s/step - loss: 0.0140 - accuracy: 0.99
50 - val_loss: 0.2177 - val_accuracy: 0.9400
Epoch 86/100
25/25 [==============================] - 152s 6s/step - loss: 0.0043 - accuracy: 0.99
87 - val_loss: 0.2828 - val_accuracy: 0.9200
Epoch 87/100
25/25 [==============================] - 151s 6s/step - loss: 0.0041 - accuracy: 0.99
75 - val_loss: 0.2737 - val_accuracy: 0.9100
Epoch 88/100
25/25 [==============================] - 154s 6s/step - loss: 0.0295 - accuracy: 0.98
62 - val_loss: 0.2257 - val_accuracy: 0.9300
Epoch 89/100
25/25 [==============================] - 199s 8s/step - loss: 0.0222 - accuracy: 0.99
62 - val_loss: 0.1680 - val_accuracy: 0.9400
Epoch 90/100
25/25 [==============================] - 154s 6s/step - loss: 0.0011 - accuracy: 1.00
00 - val_loss: 0.1457 - val_accuracy: 0.9600
Epoch 91/100
25/25 [==============================] - 153s 6s/step - loss: 0.0125 - accuracy: 0.99
75 - val_loss: 0.1173 - val_accuracy: 0.9700
Epoch 92/100
25/25 [==============================] - 153s 6s/step - loss: 0.0040 - accuracy: 1.00
00 - val_loss: 0.1417 - val_accuracy: 0.9600
Epoch 93/100
25/25 [==============================] - 154s 6s/step - loss: 0.0065 - accuracy: 0.99
87 - val_loss: 0.1905 - val_accuracy: 0.9500
Epoch 94/100
25/25 [==============================] - 153s 6s/step - loss: 0.0127 - accuracy: 0.99
50 - val_loss: 0.2365 - val_accuracy: 0.9200
Epoch 95/100
25/25 [==============================] - 153s 6s/step - loss: 0.0091 - accuracy: 0.99
62 - val_loss: 0.2739 - val_accuracy: 0.9100
Epoch 96/100
25/25 [==============================] - 152s 6s/step - loss: 0.0115 - accuracy: 0.99
62 - val_loss: 0.1569 - val_accuracy: 0.9400
Epoch 97/100
25/25 [==============================] - 151s 6s/step - loss: 0.0056 - accuracy: 0.99
62 - val_loss: 0.2158 - val_accuracy: 0.9500
Epoch 98/100
25/25 [==============================] - 152s 6s/step - loss: 0.0049 - accuracy: 0.99
87 - val_loss: 0.2031 - val_accuracy: 0.9400
Epoch 99/100
25/25 [==============================] - 152s 6s/step - loss: 0.0049 - accuracy: 0.99
87 - val_loss: 0.2359 - val_accuracy: 0.9400
Epoch 100/100
25/25 [==============================] - 152s 6s/step - loss: 0.0856 - accuracy: 0.99
50 - val_loss: 0.2317 - val_accuracy: 0.9300
```

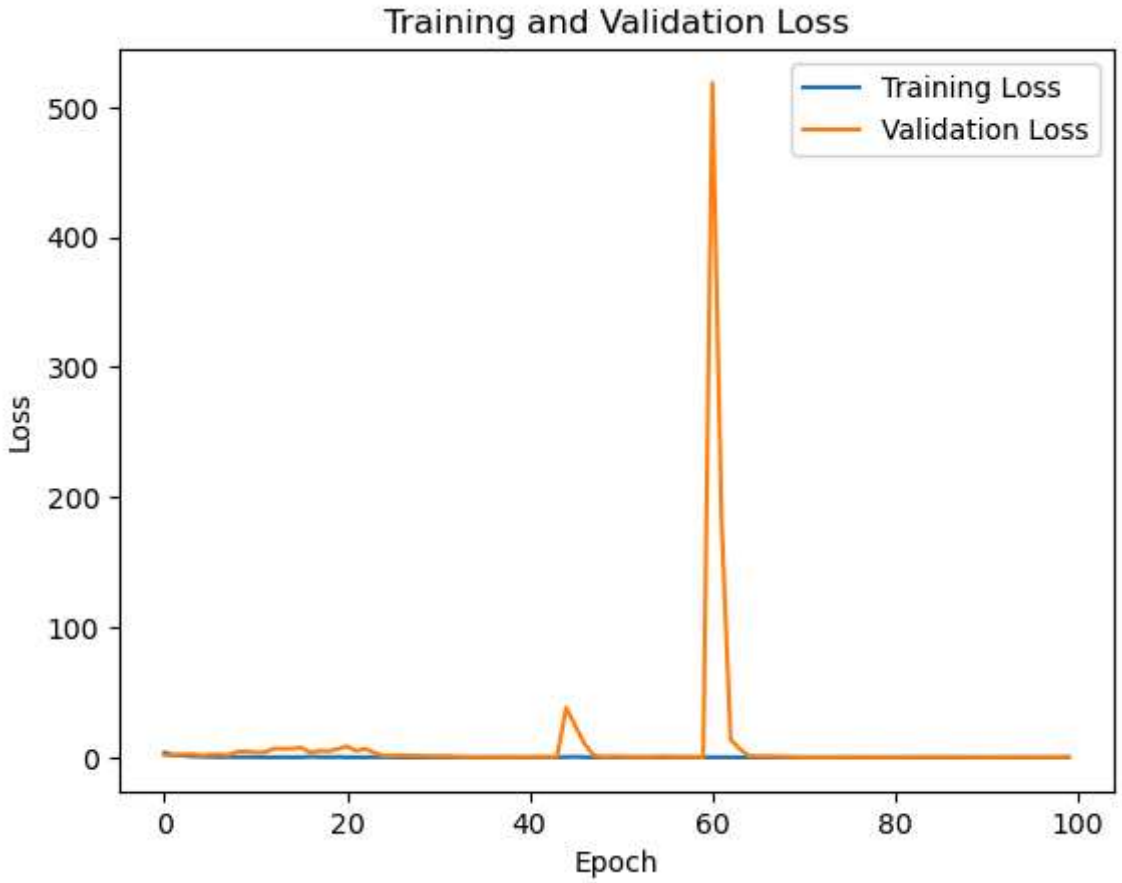
In [11]:
```
# Evaluate the model on the test set
test_loss_efficientnet, test_accuracy_efficientnet = model_efficientnet.evaluate(X_tes
```

```
4/4 [==============================] - 5s 1s/step - loss: 0.2730 - accuracy: 0.9000
```

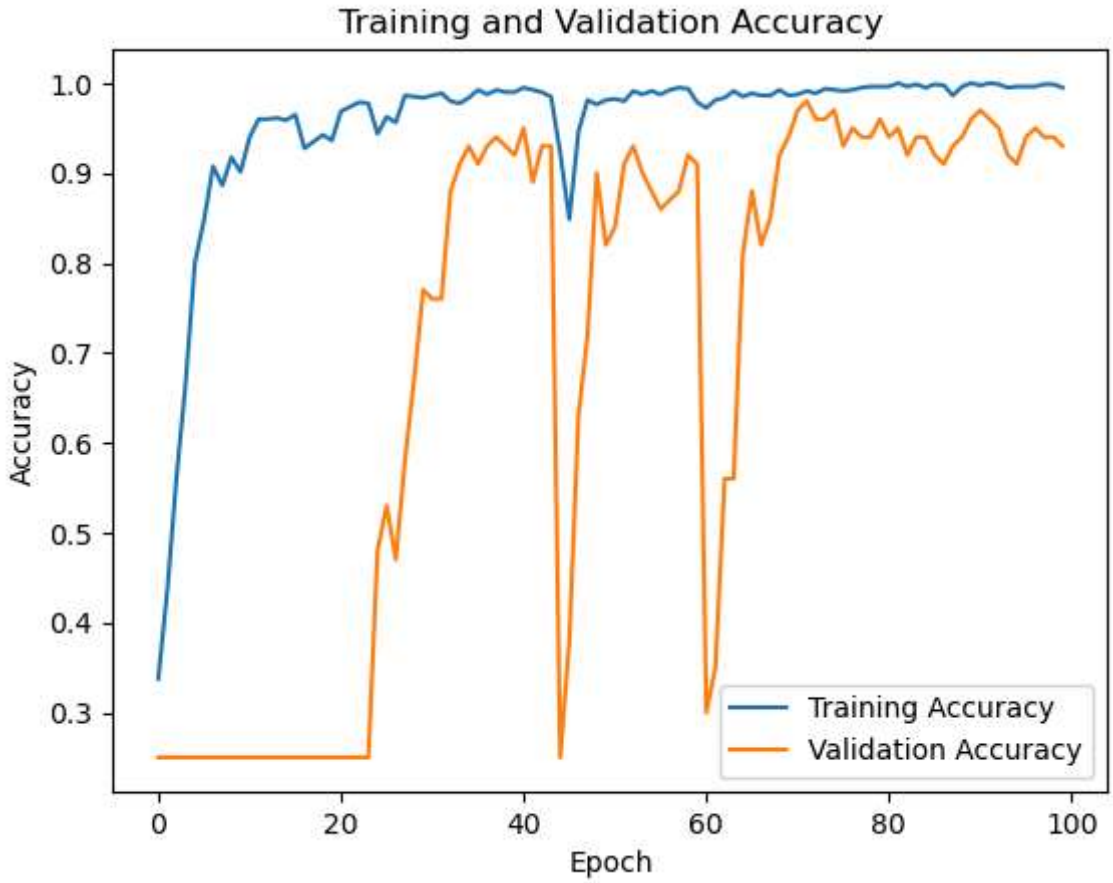
In [13]:
```
print(f"Efficient-Net Test Accuracy: {test_accuracy_efficientnet}, Test Loss: {test_lo
```

```
Efficient-Net Test Accuracy: 0.8999999761581421, Test Loss: 0.27296340465545654
```

In [14]:
```
# Plot training and validation loss
plt.plot(history_efficientnet.history['loss'], label='Training Loss')
plt.plot(history_efficientnet.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```




In [15]:
```
# Plot training and validation accuracy
plt.plot(history_efficientnet.history['accuracy'], label='Training Accuracy')
plt.plot(history_efficientnet.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```



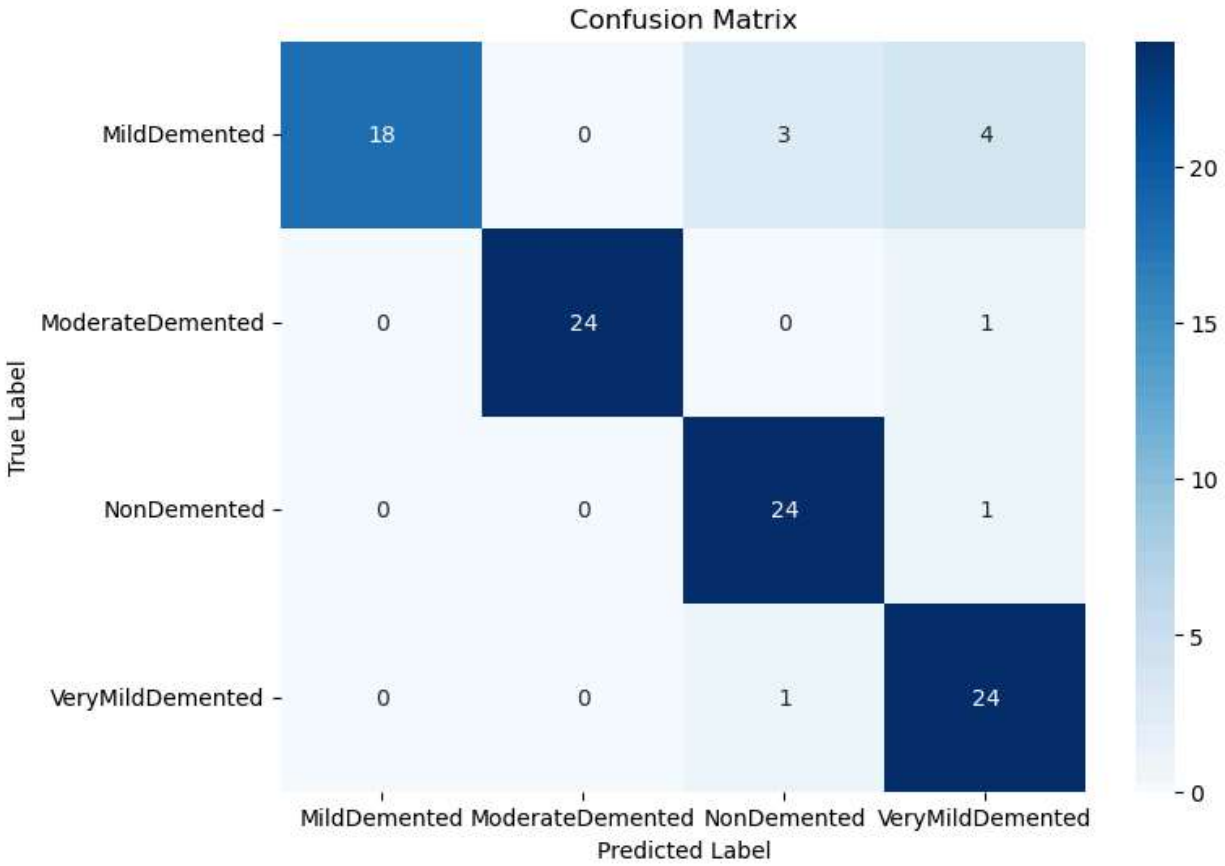

In [16]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
# Generate predictions for test set
y_pred = model_efficientnet.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Generate confusion matrix
conf_matrix = confusion_matrix(y_true_classes, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['MildDemented
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Print classification report
from sklearn.metrics import classification_report
print(classification_report(y_true_classes, y_pred_classes))
```

```
4/4 [==============================] - 30s 1s/step
```




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.72 | 0.84 | 25 |
| 1 | 1.00 | 0.96 | 0.98 | 25 |
| 2 | 0.86 | 0.96 | 0.91 | 25 |
| 3 | 0.80 | 0.96 | 0.87 | 25 |
| accuracy | | | 0.90 | 100 |
| macro avg | 0.91 | 0.90 | 0.90 | 100 |
| weighted avg | 0.91 | 0.90 | 0.90 | 100 |

In [18]:
```
from tensorflow.keras.preprocessing import image
import numpy as np

def classify_user_image(image_path):
    img = image.load_img(image_path, target_size=(150, 150))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.  # rescale image
    prediction = model_efficientnet.predict(img_array)
    class_labels = ['MildDemented', 'ModerateDemented', 'NonDemented', 'VeryMildDement
    predicted_class = class_labels[np.argmax(prediction)]
    return predicted_class

# Example usage:
user_image_path = 'C:/Users/T PRASOONA LEHARI/Desktop/FINAL_MP_DATASET/MildDemented/mi
predicted_class = classify_user_image(user_image_path)
print(f"Predicted Class: {predicted_class}")#MildDemented
```

```
1/1 [==============================] - 0s 292ms/step
Predicted Class: MildDemented
```