practical-examination

November 22, 2024

1 Alzhimer Disease detection USING DEEP LEARNING MODALS

GROUP MEMBERS: * VIRAJ RAINA - 202101040042 * AKASH TOTRE - 202101090009

2 MODALS WE USED IN OUR PROJECT:

1) VGG16
2) INCEPTION-V3
3) RESENT-50
4) ALEXNET

3 IMPORT

[1]:
```
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
import random

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Input, Dropout, Flatten, Conv2D,␣
 ↪GlobalAveragePooling2D, concatenate
from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooling2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.utils import plot_model
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications import InceptionV3
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import load_model
from sklearn.utils import shuffle
```

```
%matplotlib inline
```

4 LOAD DATASET

[2]:
```
import os
os.environ['KAGGLE_USERNAME'] = "aniketsinghraju"
os.environ['KAGGLE_KEY'] = "5d04f4fb1bd8611556f0bb54607f35bc"

!kaggle datasets download -d lukechugh/best-alzheimer-mri-dataset-99-accuracy
```

```
Dataset URL: https://www.kaggle.com/datasets/lukechugh/best-alzheimer-mri-
dataset-99-accuracy
License(s): ODbL-1.0
Downloading best-alzheimer-mri-dataset-99-accuracy.zip to /content
 95% 68.0M/71.5M [00:00<00:00, 70.6MB/s]
100% 71.5M/71.5M [00:00<00:00, 76.7MB/s]
```

[5]:
```
!unzip best-alzheimer-mri-dataset-99-accuracy.zip
```

```
Archive:  best-alzheimer-mri-dataset-99-accuracy.zip
replace Combined Dataset/test/Mild Impairment/1 (10).jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename:
```

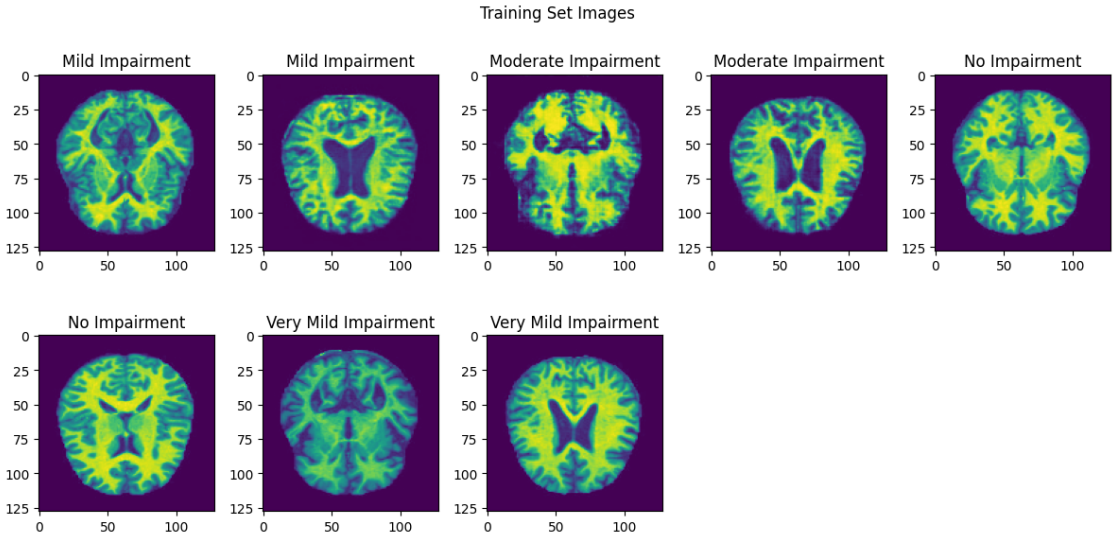
5 SET VARAIABLES

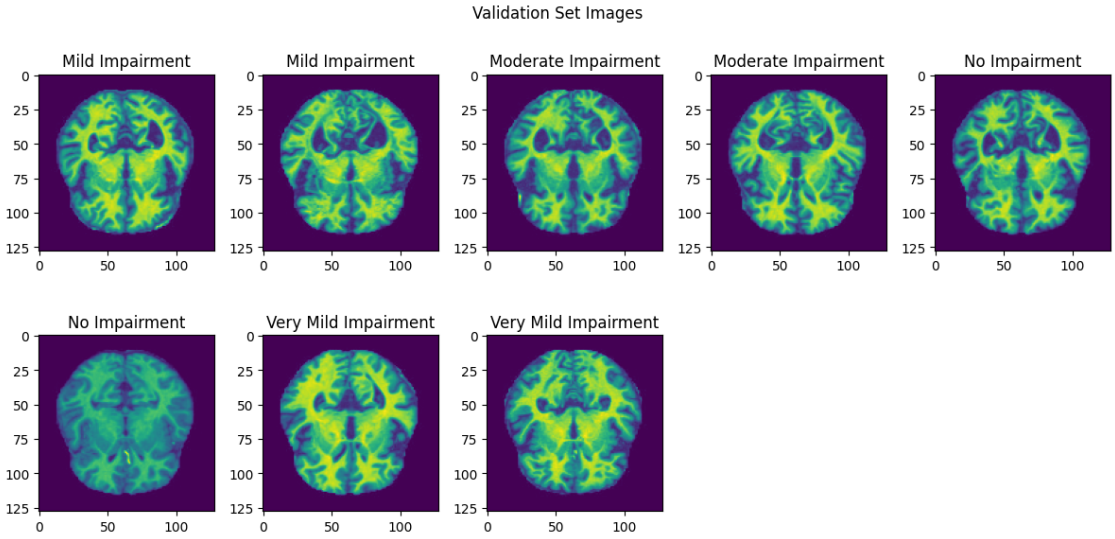
[4]:
```
img_size = (150, 150)
batch_size = 32

data_dir = "/content/Combined Dataset/train"

category_names = sorted(os.listdir(data_dir))

datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=img_size,
```

```
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True,
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    data_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False,
    subset='validation'
)

subset_images_train = []
subset_labels_train = []
class_counts_train = {cat: 0 for cat in category_names}
num_images_per_class = 2

for i in range(len(train_generator.filenames)):
    image_path = os.path.join(data_dir, train_generator.filenames[i])
    label = train_generator.labels[i]
    category_name = category_names[label]

    if class_counts_train[category_name] < num_images_per_class:
        img = plt.imread(image_path)
        subset_images_train.append(img)
        subset_labels_train.append(category_name)
        class_counts_train[category_name] += 1

subset_images_validation = []
subset_labels_validation = []
class_counts_validation = {cat: 0 for cat in category_names}

for i in range(len(validation_generator.filenames)):
    image_path = os.path.join(data_dir, validation_generator.filenames[i])
    label = validation_generator.labels[i]
    category_name = category_names[label]

    if class_counts_validation[category_name] < num_images_per_class:
        img = plt.imread(image_path)
        subset_images_validation.append(img)
        subset_labels_validation.append(category_name)
        class_counts_validation[category_name] += 1

plt.figure(figsize=(12, 6))
```

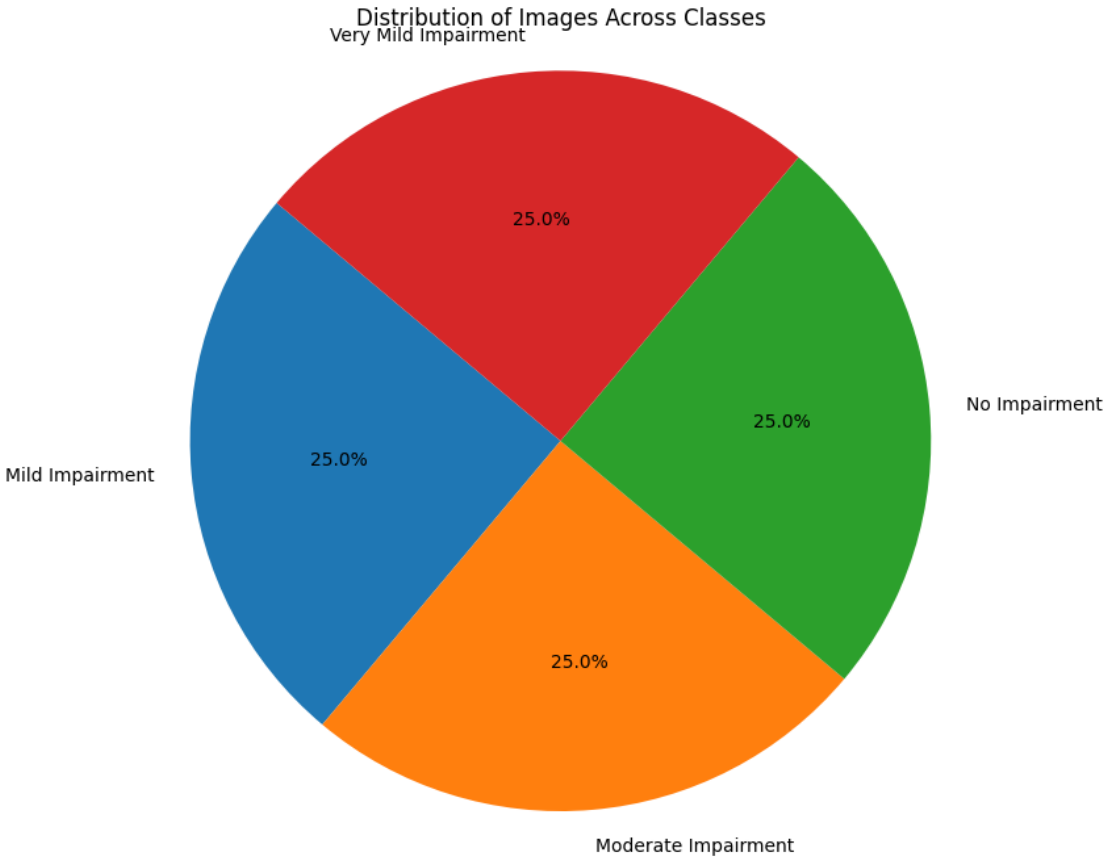
```
plt.suptitle("Training Set Images")
for i in range(len(subset_images_train)):
    plt.subplot(2,5,i+1)
    plt.imshow(subset_images_train[i])
    plt.title(subset_labels_train[i])
plt.tight_layout()

plt.figure(figsize=(12, 6))
plt.suptitle("Validation Set Images")
for i in range(len(subset_images_validation)):
    plt.subplot(2,5,i+1)
    plt.imshow(subset_images_validation[i])
    plt.title(subset_labels_validation[i])
plt.tight_layout()

plt.show()
```

```
Found 8192 images belonging to 4 classes.
Found 2048 images belonging to 4 classes.
```





[ ]:
```
class_names = list(train_generator.class_indices.keys())
class_counts = [train_generator.labels.tolist().count(class_index) for␣
 ↪class_index in range(len(class_names))]

plt.figure(figsize=(8, 8))
plt.pie(class_counts, labels=class_names, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Images Across Classes')

plt.axis('equal')

plt.show()
```



6 VGG16

[ ]:
```
img_size = (150, 150)
batch_size = 32
num_classes = len(train_generator.class_indices)

base_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(img_size[0], img_size[1], 3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

for layer in base_model.layers:
    layer.trainable = False
```

```
model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=40,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    verbose=1
)

model.save('alzheimer_vgg16.h5')
```

```
Model: "functional_1"
```

```
 Layer (type)                 Output Shape              ␣
 ↪Param #

 input_layer_1 (InputLayer)   (None, 150, 150, 3)       ␣
 ↪  0

 block1_conv1 (Conv2D)        (None, 150, 150, 64)      ␣
 ↪1,792

 block1_conv2 (Conv2D)        (None, 150, 150, 64)      ␣
 ↪36,928

 block1_pool (MaxPooling2D)   (None, 75, 75, 64)        ␣
 ↪  0

 block2_conv1 (Conv2D)        (None, 75, 75, 128)       ␣
 ↪73,856

 block2_conv2 (Conv2D)        (None, 75, 75, 128)       ␣
 ↪147,584

 block2_pool (MaxPooling2D)   (None, 37, 37, 128)       ␣
 ↪  0

 block3_conv1 (Conv2D)        (None, 37, 37, 256)       ␣
 ↪295,168
```

```
accuracy: 0.6401 - loss: 0.8512 - val_accuracy: 0.5752 - val_loss: 0.9650
Epoch 30/40
150/150             44s 293ms/step -
accuracy: 0.6702 - loss: 0.8032 - val_accuracy: 0.5601 - val_loss: 0.9491
Epoch 31/40
150/150             47s 304ms/step -
accuracy: 0.6399 - loss: 0.8382 - val_accuracy: 0.5640 - val_loss: 0.9391
Epoch 32/40
150/150             35s 237ms/step -
accuracy: 0.6696 - loss: 0.8120 - val_accuracy: 0.5640 - val_loss: 0.9510
Epoch 33/40
150/150             46s 299ms/step -
accuracy: 0.6621 - loss: 0.8090 - val_accuracy: 0.5674 - val_loss: 0.9509
Epoch 34/40
150/150             41s 274ms/step -
accuracy: 0.6535 - loss: 0.8200 - val_accuracy: 0.5684 - val_loss: 0.9595
Epoch 35/40
150/150             105s 322ms/step -
accuracy: 0.6589 - loss: 0.8056 - val_accuracy: 0.5806 - val_loss: 0.9356
Epoch 36/40
150/150             41s 277ms/step -
accuracy: 0.6587 - loss: 0.8121 - val_accuracy: 0.5850 - val_loss: 0.9118
Epoch 37/40
150/150             55s 358ms/step -
accuracy: 0.6622 - loss: 0.7919 - val_accuracy: 0.5732 - val_loss: 0.9260
Epoch 38/40
150/150             34s 225ms/step -
accuracy: 0.6684 - loss: 0.7906 - val_accuracy: 0.5957 - val_loss: 0.9067
Epoch 39/40
150/150             49s 311ms/step -
accuracy: 0.6855 - loss: 0.7598 - val_accuracy: 0.5820 - val_loss: 0.9193
Epoch 40/40
150/150             34s 229ms/step -
accuracy: 0.6696 - loss: 0.7834 - val_accuracy: 0.5811 - val_loss: 0.9022

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

7 RESNET50

[ ]:
```
img_size = (150, 150)
batch_size = 32
num_classes = len(train_generator.class_indices)
```

```
base_model = ResNet50(weights='imagenet', include_top=False,␣
 ↪input_shape=(img_size[0], img_size[1], 3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])


model.summary()

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=40,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    verbose=1
)
model.save('alzheimer_resnet50_40.h5')
```

```
Model: "functional_3"
```

```
 Layer (type)                Output Shape              Param #   Connected␣
 ↪to

 input_layer_3               (None, 150, 150, 3)             0   -          ␣
 ↪
 (InputLayer)                                                               ␣
 ↪

 conv1_pad (ZeroPadding2D)   (None, 156, 156, 3)             0   ␣
 ↪input_layer_3[0][0]

 conv1_conv (Conv2D)         (None, 75, 75, 64)          9,472   ␣
 ↪conv1_pad[0][0]
```

```
 global_average_pooling2d…  (None, 2048)                    0  ␣
 ↪conv5_block3_out[0][0]
 (GlobalAveragePooling2D)                                      ␣
 ↪

 dense_6 (Dense)            (None, 1024)              2,098,176  ␣
 ↪global_average_poolin…

 dropout_3 (Dropout)        (None, 1024)                    0  ␣
 ↪dense_6[0][0]

 dense_7 (Dense)            (None, 4)                   4,100  ␣
 ↪dropout_3[0][0]


 Total params: 25,689,988 (98.00 MB)

 Trainable params: 2,102,276 (8.02 MB)

 Non-trainable params: 23,587,712 (89.98 MB)

Epoch 1/40
150/150             61s 336ms/step -
accuracy: 0.2696 - loss: 1.5051 - val_accuracy: 0.3423 - val_loss: 1.3587
Epoch 2/40
  2/150             12s 87ms/step - accuracy:
0.1875 - loss: 1.5534

/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data;
interrupting training. Make sure that your dataset or generator can generate at
least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()`
function when building your dataset.
  self.gen.throw(typ, value, traceback)

150/150             36s 238ms/step -
accuracy: 0.2782 - loss: 1.4356 - val_accuracy: 0.3638 - val_loss: 1.3483
Epoch 3/40
150/150             98s 304ms/step -
accuracy: 0.3154 - loss: 1.3803 - val_accuracy: 0.3159 - val_loss: 1.3348
Epoch 4/40
150/150             42s 279ms/step -
accuracy: 0.3352 - loss: 1.3512 - val_accuracy: 0.3672 - val_loss: 1.3297
Epoch 5/40
150/150             46s 303ms/step -
accuracy: 0.3259 - loss: 1.3468 - val_accuracy: 0.3281 - val_loss: 1.3313
Epoch 6/40
```

```
150/150             42s 283ms/step -
accuracy: 0.4213 - loss: 1.2278 - val_accuracy: 0.4360 - val_loss: 1.2339
Epoch 39/40
150/150             101s 304ms/step -
accuracy: 0.4261 - loss: 1.2375 - val_accuracy: 0.4312 - val_loss: 1.2527
Epoch 40/40
150/150             34s 226ms/step -
accuracy: 0.4243 - loss: 1.2229 - val_accuracy: 0.4165 - val_loss: 1.2379
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

8 INCEPTION

[ ]:
```
img_size = (150, 150)
batch_size = 32
num_classes = len(train_generator.class_indices)

base_model = InceptionV3(weights='imagenet', include_top=False,␣
 ↪input_shape=(img_size[0], img_size[1], 3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=40,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    verbose=1
```

```
)

model.save('alzheimer_inception_40.h5')
```

```
Model: "functional_5"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_5 (InputLayer)	(None, 150, 150, 3)	0	-
conv2d_94 (Conv2D)	(None, 74, 74, 32)	864	input_layer_5[0][0]
batch_normalization_94 (BatchNormalization)	(None, 74, 74, 32)	96	conv2d_94[0][0]
activation_94 (Activation)	(None, 74, 74, 32)	0	batch_normalization_9…
conv2d_95 (Conv2D)	(None, 72, 72, 32)	9,216	activation_94[0][0]
batch_normalization_95 (BatchNormalization)	(None, 72, 72, 32)	96	conv2d_95[0][0]
activation_95 (Activation)	(None, 72, 72, 32)	0	batch_normalization_9…
conv2d_96 (Conv2D)	(None, 72, 72, 64)	18,432	activation_95[0][0]
batch_normalization_96	(None, 72, 72, 64)	192	conv2d_96[0][0]

```
↪concatenate_3[0][0],                                                  ␣
                                                                       ␣
↪activation_187[0][0]

global_average_pooling2d…   (None, 2048)                    0          ␣
↪mixed10[0][0]
(GlobalAveragePooling2D)                                               ␣
↪

dense_10 (Dense)            (None, 1024)                2,098,176      ␣
↪global_average_poolin…

dropout_5 (Dropout)         (None, 1024)                    0          ␣
↪dense_10[0][0]

dense_11 (Dense)            (None, 4)                       4,100      ␣
↪dropout_5[0][0]


 Total params: 23,905,060 (91.19 MB)

 Trainable params: 2,102,276 (8.02 MB)

 Non-trainable params: 21,802,784 (83.17 MB)

Epoch 1/40
256/256             80s 266ms/step -
accuracy: 0.3855 - loss: 1.6475 - val_accuracy: 0.5557 - val_loss: 1.0102
Epoch 2/40
256/256             6s 23ms/step -
accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 3/40
256/256             68s 251ms/step -
accuracy: 0.5231 - loss: 1.0702 - val_accuracy: 0.5645 - val_loss: 0.9407
Epoch 4/40
256/256             0s 127us/step -
accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 5/40
256/256             66s 254ms/step -
accuracy: 0.5603 - loss: 1.0164 - val_accuracy: 0.5991 - val_loss: 0.9186
Epoch 6/40
256/256             8s 31ms/step -
accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 7/40
```

```
256/256             84s 252ms/step -
accuracy: 0.6672 - loss: 0.7562 - val_accuracy: 0.6060 - val_loss: 0.8477
Epoch 40/40
256/256             0s 115us/step -
accuracy: 0.0000e+00 - loss: 0.0000e+00

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

9 ALEXNET

[ ]:
```
img_size = (150, 150)
batch_size = 32
input_layer = Input(shape=(img_size[0], img_size[1], 3))
x = Conv2D(96, (11, 11), strides=(4, 4), activation='relu')(input_layer)
x = MaxPooling2D((3, 3), strides=(2, 2))(x)
x = Conv2D(256, (5, 5), activation='relu')(x)
x = MaxPooling2D((3, 3), strides=(2, 2))(x)
x = Conv2D(384, (3, 3), activation='relu')(x)
x = Conv2D(384, (3, 3), activation='relu')(x)
x = Conv2D(256, (3, 3), padding='same', activation='relu')(x)
x = MaxPooling2D((2, 2), strides=(2, 2))(x)
x = Flatten()(x)
x = Dense(4096, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(4096, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(train_generator.num_classes, activation='softmax')(x)

model = Model(inputs=input_layer, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=40,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    verbose=1
)
```

```
model.save('alzheimer_alexnet.h5')
```

```
Model: "functional_6"
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_layer_6 (InputLayer)   (None, 150, 150, 3)       0
_________________________________________________________________
conv2d_188 (Conv2D)          (None, 35, 35, 96)        34,944
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 17, 17, 96)        0
_________________________________________________________________
conv2d_189 (Conv2D)          (None, 13, 13, 256)       614,656
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
conv2d_190 (Conv2D)          (None, 4, 4, 384)         885,120
_________________________________________________________________
conv2d_191 (Conv2D)          (None, 2, 2, 384)         1,327,488
_________________________________________________________________
conv2d_192 (Conv2D)          (None, 2, 2, 256)         884,992
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 1, 1, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense_12 (Dense)             (None, 4096)              1,052,672
_________________________________________________________________
dropout_6 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_13 (Dense)             (None, 4096)              16,781,312
```

```
256/256             9s 35ms/step -
accuracy: 0.0000e+00 - loss: 0.0000e+00
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

10 SAVE THE MODELS

[7]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[8]:
```
!cp alzheimer_resnet50_40.h5 /content/drive/MyDrive/
```

11 EVAL

[ ]:
```
def evaluate_model(model_name, model_filename):
    custom_model = load_model(model_filename)

    test_generator = datagen.flow_from_directory(
        "/content/Combined Dataset/test",
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

    test_loss, test_accuracy = custom_model.evaluate(test_generator, verbose=1)
    print(f"{model_name} - Test Accuracy: {test_accuracy * 100:.2f}%")
    print(f"{model_name} - Test Loss: {test_loss:.4f}")

    predictions = custom_model.predict(test_generator)
    predicted_labels = np.argmax(predictions, axis=1)

    true_labels = test_generator.classes

    class_report = classification_report(true_labels, predicted_labels,␣
 ↪target_names=test_generator.class_indices.keys())
    print(f"{model_name} - Classification Report:\n", class_report)

    confusion_mtx = confusion_matrix(true_labels, predicted_labels)

    plt.figure(figsize=(8, 6))
```

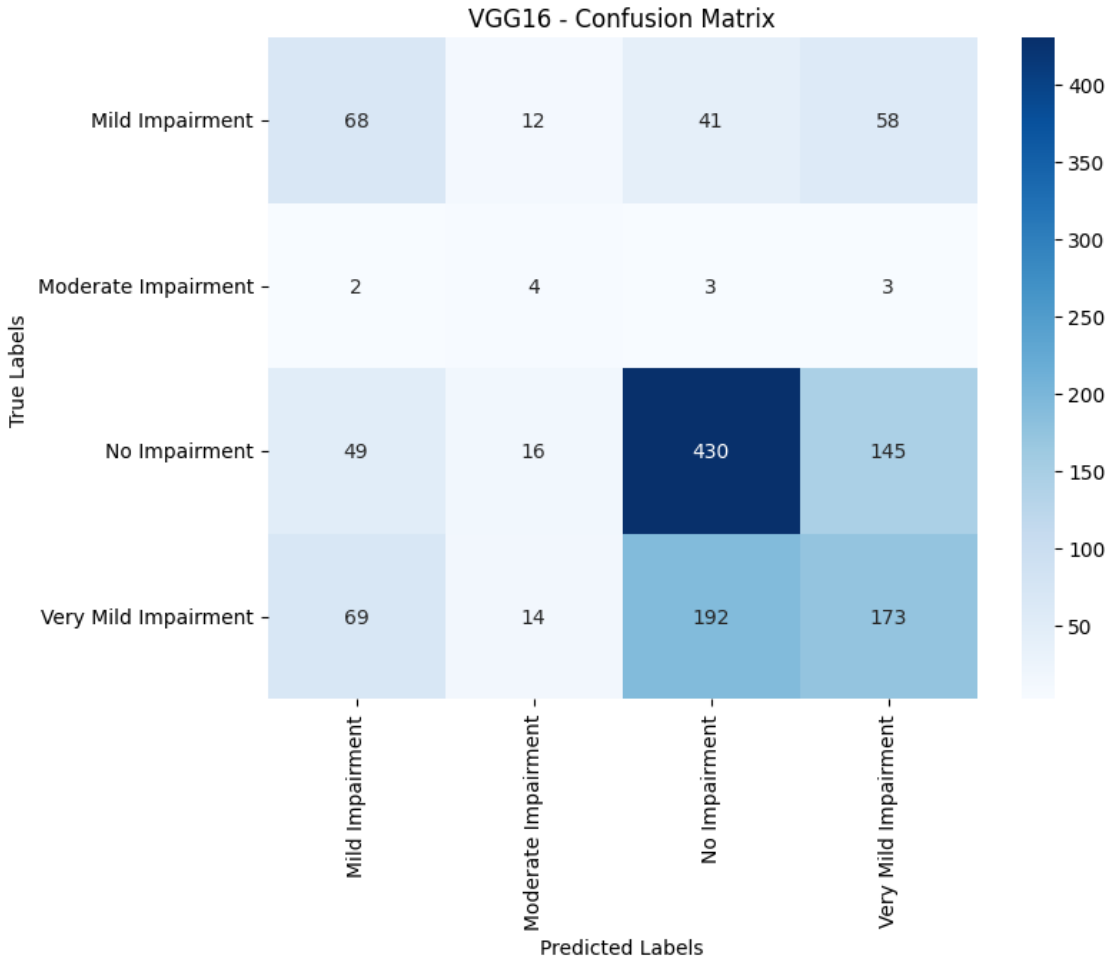
```
    sns.heatmap(confusion_mtx, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=test_generator.class_indices.keys(), yticklabels=test_generator.
 ↪class_indices.keys())
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f"{model_name} - Confusion Matrix")
    plt.show()

evaluate_model("VGG16", "alzheimer_vgg16_40.h5")
evaluate_model("ResNet50", "alzheimer_resnet50_40.h5")
evaluate_model("InceptionV3", "alzheimer_inception_40.h5")
evaluate_model("AlexNet", "alzheimer_alexnet_40.h5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
```

```
Found 1279 images belonging to 4 classes.
```

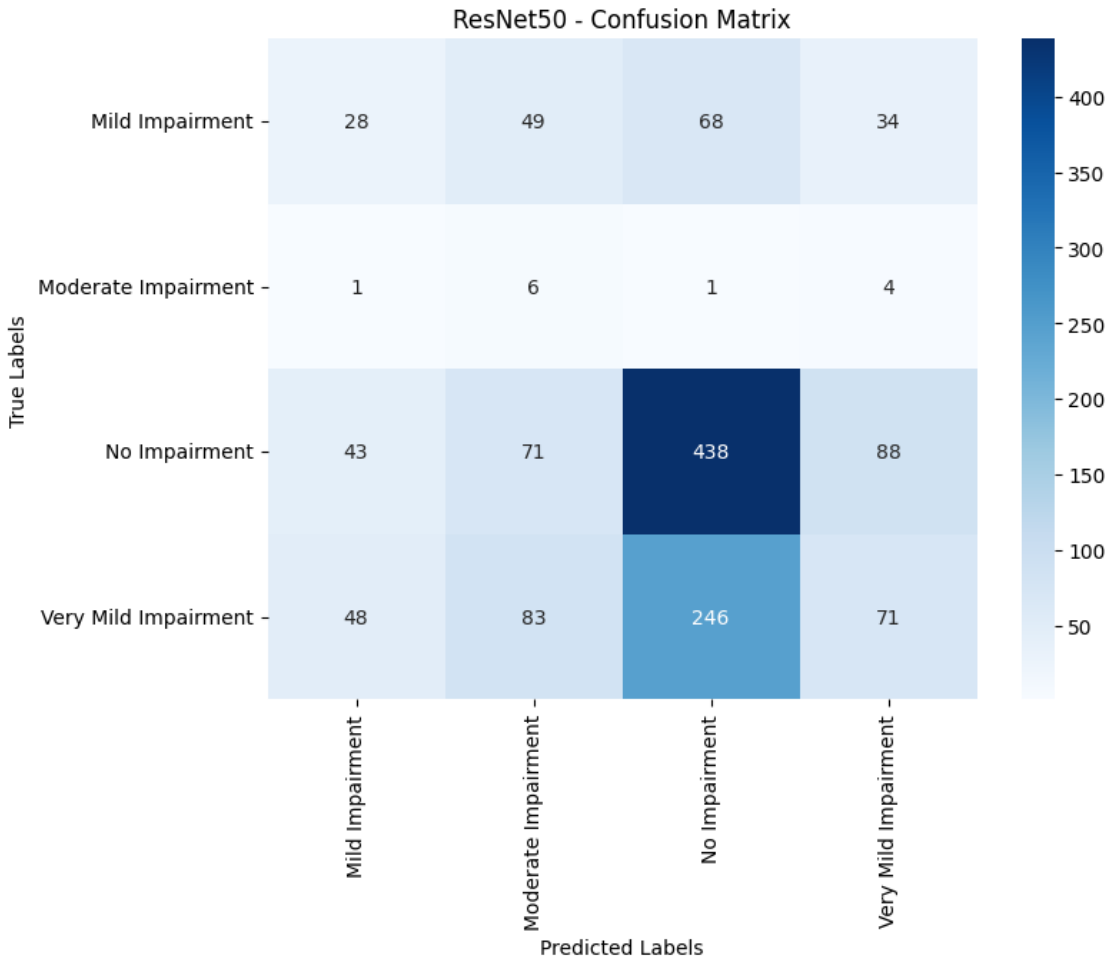
```
/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
40/40            10s 235ms/step -
accuracy: 0.5097 - loss: 1.0142
VGG16 - Test Accuracy: 54.42%
VGG16 - Test Loss: 0.9648
40/40            7s 163ms/step
VGG16 - Classification Report:
                    precision    recall  f1-score   support

     Mild Impairment       0.36      0.38      0.37       179
 Moderate Impairment       0.09      0.33      0.14        12
       No Impairment       0.65      0.67      0.66       640
Very Mild Impairment       0.46      0.39      0.42       448

            accuracy                           0.53      1279
           macro avg       0.39      0.44      0.40      1279
        weighted avg       0.53      0.53      0.53      1279
```



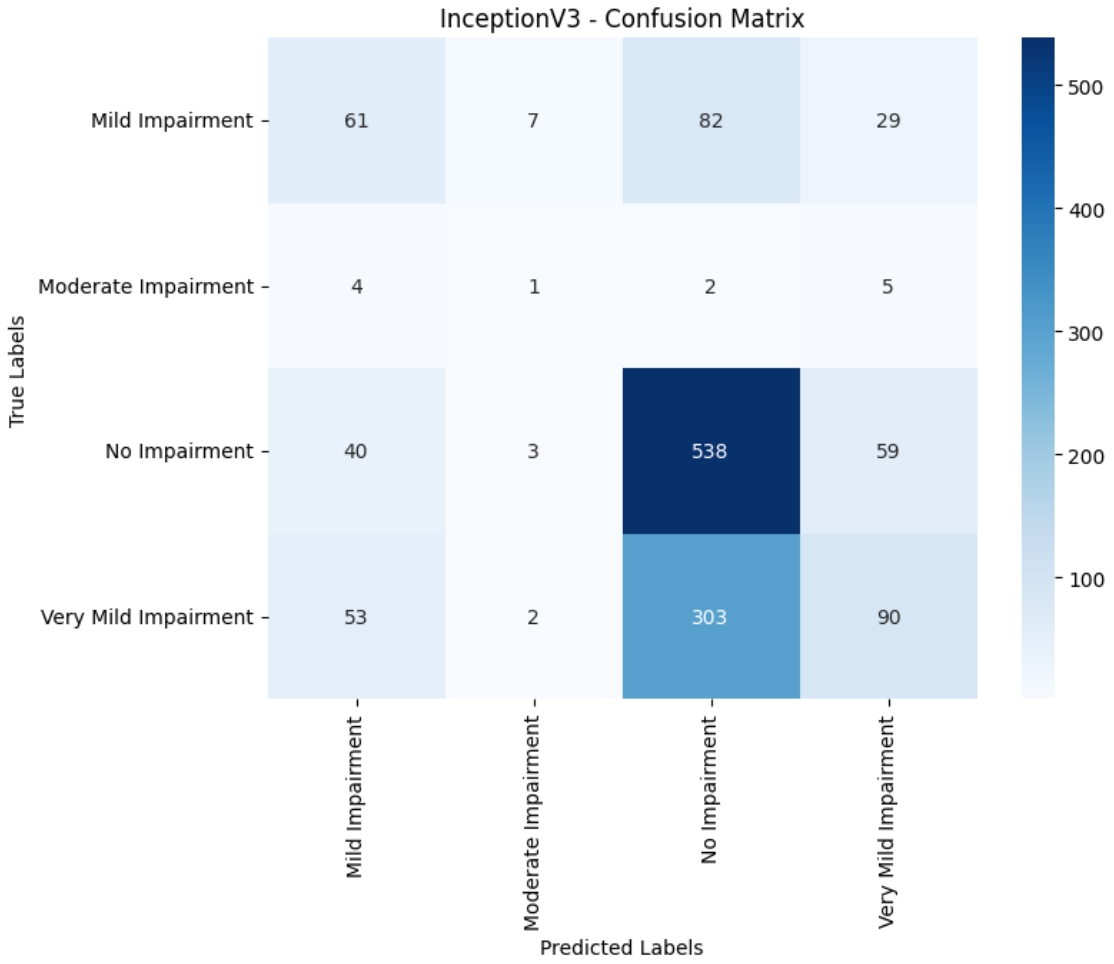

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.

Found 1279 images belonging to 4 classes.

/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()

40/40            13s 210ms/step -
accuracy: 0.4138 - loss: 1.2228
ResNet50 - Test Accuracy: 42.46%
ResNet50 - Test Loss: 1.1840
40/40            14s 231ms/step
```

```
ResNet50 - Classification Report:
                      precision    recall  f1-score   support

     Mild Impairment       0.23      0.16      0.19       179
 Moderate Impairment       0.03      0.50      0.05        12
       No Impairment       0.58      0.68      0.63       640
Very Mild Impairment       0.36      0.16      0.22       448

            accuracy                           0.42      1279
           macro avg       0.30      0.37      0.27      1279
        weighted avg       0.45      0.42      0.42      1279
```



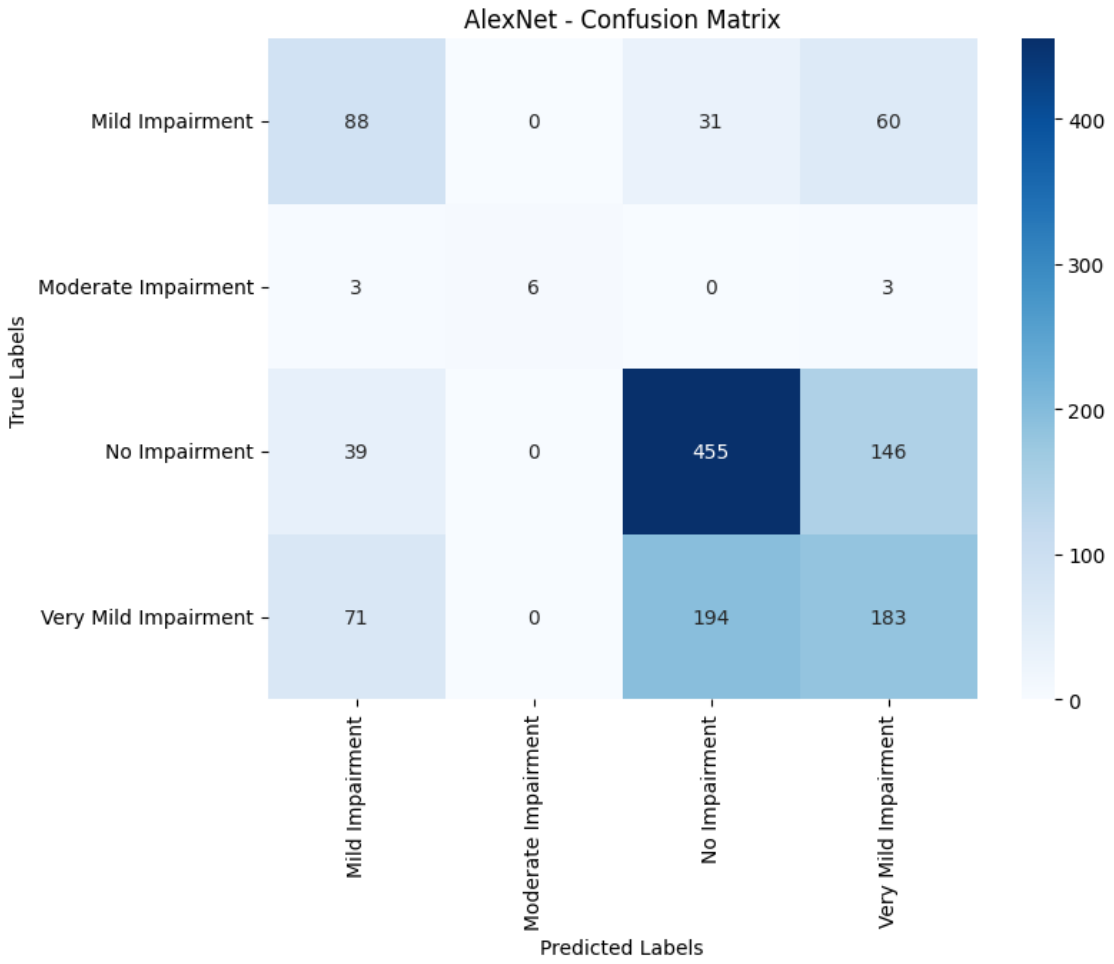

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.

Found 1279 images belonging to 4 classes.
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
40/40            16s 229ms/step -
accuracy: 0.5677 - loss: 0.9693
InceptionV3 - Test Accuracy: 54.73%
InceptionV3 - Test Loss: 0.9370
40/40            15s 290ms/step
InceptionV3 - Classification Report:
                     precision    recall  f1-score   support

     Mild Impairment       0.39      0.34      0.36       179
 Moderate Impairment       0.08      0.08      0.08        12
       No Impairment       0.58      0.84      0.69       640
Very Mild Impairment       0.49      0.20      0.29       448

            accuracy                           0.54      1279
           macro avg       0.38      0.37      0.35      1279
        weighted avg       0.52      0.54      0.50      1279
```




```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.

Found 1279 images belonging to 4 classes.

/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()

40/40            10s 217ms/step -
accuracy: 0.5924 - loss: 0.8624
AlexNet - Test Accuracy: 58.17%
AlexNet - Test Loss: 0.8575
40/40            7s 173ms/step
```

```
AlexNet - Classification Report:
                      precision    recall  f1-score   support

     Mild Impairment       0.44      0.49      0.46       179
 Moderate Impairment       1.00      0.50      0.67        12
       No Impairment       0.67      0.71      0.69       640
Very Mild Impairment       0.47      0.41      0.44       448

            accuracy                           0.57      1279
           macro avg       0.64      0.53      0.56      1279
        weighted avg       0.57      0.57      0.57      1279
```

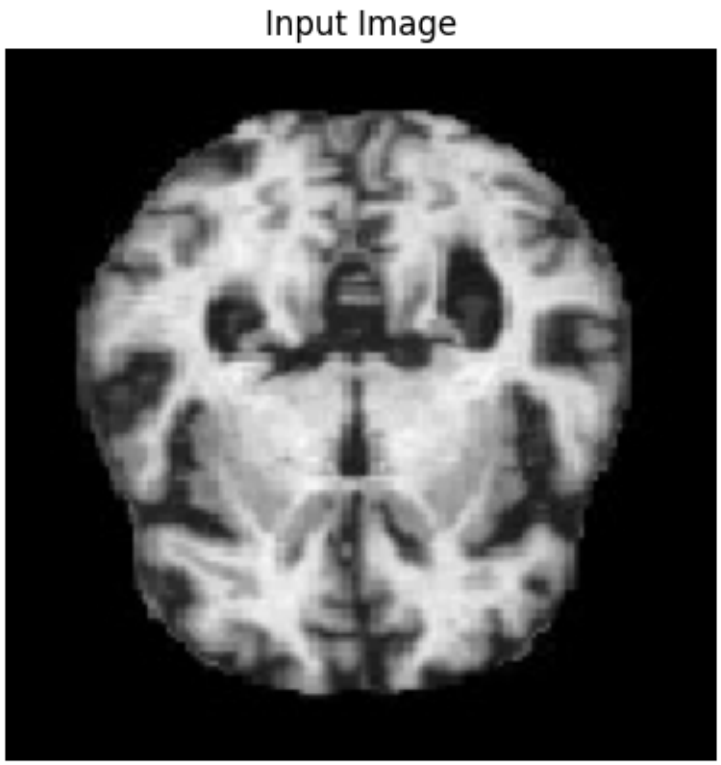


[10]:
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
# Load the models
model_vgg16 = load_model('alzheimer_vgg16_40.h5')
model_inception = load_model('alzheimer_inception_40.h5')
model_alexnet = load_model('alzheimer_alexnet_40.h5')
model_resnet50 = load_model('alzheimer_resnet50_40.h5')  # Load ResNet50 model

# Define the image size (make sure it matches the models' expected input size)
img_size = (150, 150)

# Function to make a prediction and display results for multiple models
def predict_image(image_path, models, class_indices):
    # Load the image
    img = load_img(image_path, target_size=img_size)

    # Display the input image
    plt.imshow(img)
    plt.title("Input Image")
    plt.axis('off')
    plt.show()

    # Convert image to array and preprocess
    img_array = img_to_array(img) / 255.0  # Normalize the image
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    # Initialize a dictionary to store predictions
    predictions = {}

    # Predict with each model
    for model_name, model in models.items():
        pred = model.predict(img_array)
        predicted_class = np.argmax(pred, axis=1)[0]  # Get the predicted class␣
↪index
        class_labels = {v: k for k, v in class_indices.items()}  # Reverse the␣
↪class indices
        predicted_label = class_labels[predicted_class]  # Get the predicted␣
↪class label
        predictions[model_name] = {'label': predicted_label, 'confidence':␣
↪pred[0]}

    # Print out the predictions for each model
    for model_name, result in predictions.items():
        print(f"{model_name} Prediction: {result['label']}")
        print(f"Confidence Scores: {result['confidence']}\n")

    return predictions
```

```
# Path to the image you want to test
image_path = "/content/Combined Dataset/test/Mild Impairment/1 (10).jpg"

# Class indices (replace with your actual class mapping from the generator)
class_indices = {'Mild Impairment': 0, 'Moderate Impairment': 1, 'No␣
 ↪Impairment': 2, 'Very Mild Impairment': 3}  # Example

# List of models
models = {
    'VGG16': model_vgg16,
    'InceptionV3': model_inception,
    'AlexNet': model_alexnet,
    'ResNet50': model_resnet50  # Add ResNet50 to the list
}

# Make predictions
predictions = predict_image(image_path, models, class_indices)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
```




```
1/1             2s 2s/step
1/1             6s 6s/step
1/1             1s 1s/step
1/1             5s 5s/step
VGG16 Prediction: Mild Impairment
Confidence Scores: [0.690032   0.00656325 0.15104868 0.15235607]

InceptionV3 Prediction: Mild Impairment
Confidence Scores: [6.2403983e-01 9.7702621e-05 6.5274172e-02 3.1058836e-01]

AlexNet Prediction: Mild Impairment
Confidence Scores: [9.0663189e-01 1.3889793e-04 1.0717951e-02 8.2511283e-02]

ResNet50 Prediction: Moderate Impairment
Confidence Scores: [0.29382876 0.32801533 0.13538507 0.2427708 ]
```

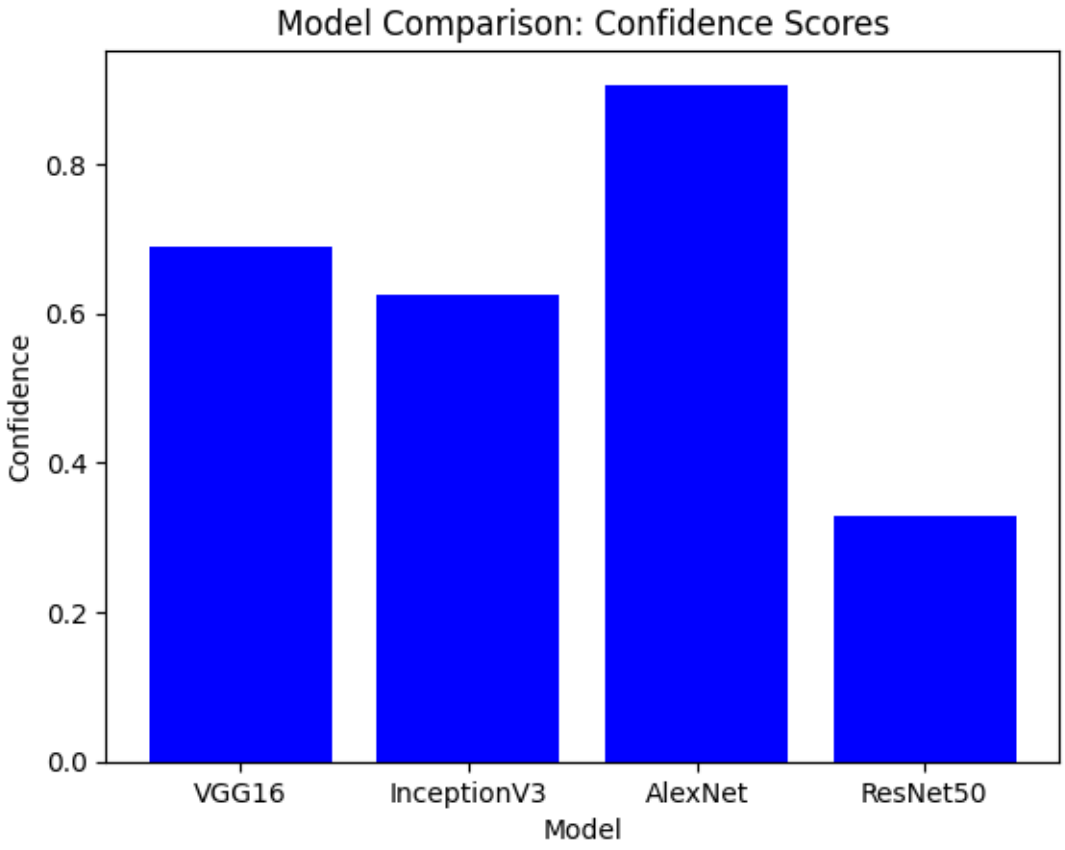
[11]:
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
# Load models (as before)
model_vgg16 = load_model('alzheimer_vgg16_40.h5')
model_inception = load_model('alzheimer_inception_40.h5')
model_alexnet = load_model('alzheimer_alexnet_40.h5')
model_resnet50 = load_model('alzheimer_resnet50_40.h5')

# Define the image size
img_size = (150, 150)

# Function to make a prediction and display results
def compare_models(image_path, models, class_indices):
    # Load the image
    img = load_img(image_path, target_size=img_size)
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    # Initialize a dictionary to store predictions
    predictions = {}

    for model_name, model in models.items():
        pred = model.predict(img_array)
        predicted_class = np.argmax(pred, axis=1)[0]
        class_labels = {v: k for k, v in class_indices.items()}
        predicted_label = class_labels[predicted_class]
        predictions[model_name] = pred[0]  # Store the confidence scores

    # Plotting the confidence scores for each model
    model_names = list(predictions.keys())
    confidence_scores = [np.max(predictions[model_name]) for model_name in␣
 ↪model_names]

    plt.bar(model_names, confidence_scores, color='blue')
    plt.title("Model Comparison: Confidence Scores")
    plt.ylabel("Confidence")
    plt.xlabel("Model")
    plt.show()

# Path to the image you want to test
image_path1 = "/content/Combined Dataset/test/Mild Impairment/1 (10).jpg"
image_path2 = "/content/Combined Dataset/test/Moderate Impairment/13 (2).jpg"
image_path3 = "/content/Combined Dataset/test/No Impairment/1 (16).jpg"
image_path4 = "/content/Combined Dataset/test/Very Mild Impairment/1 (10).jpg"
class_indices = {'Mild Impairment': 0, 'Moderate Impairment': 1, 'No␣
 ↪Impairment': 2, 'Very Mild Impairment': 3}

# List of models
models = {
```

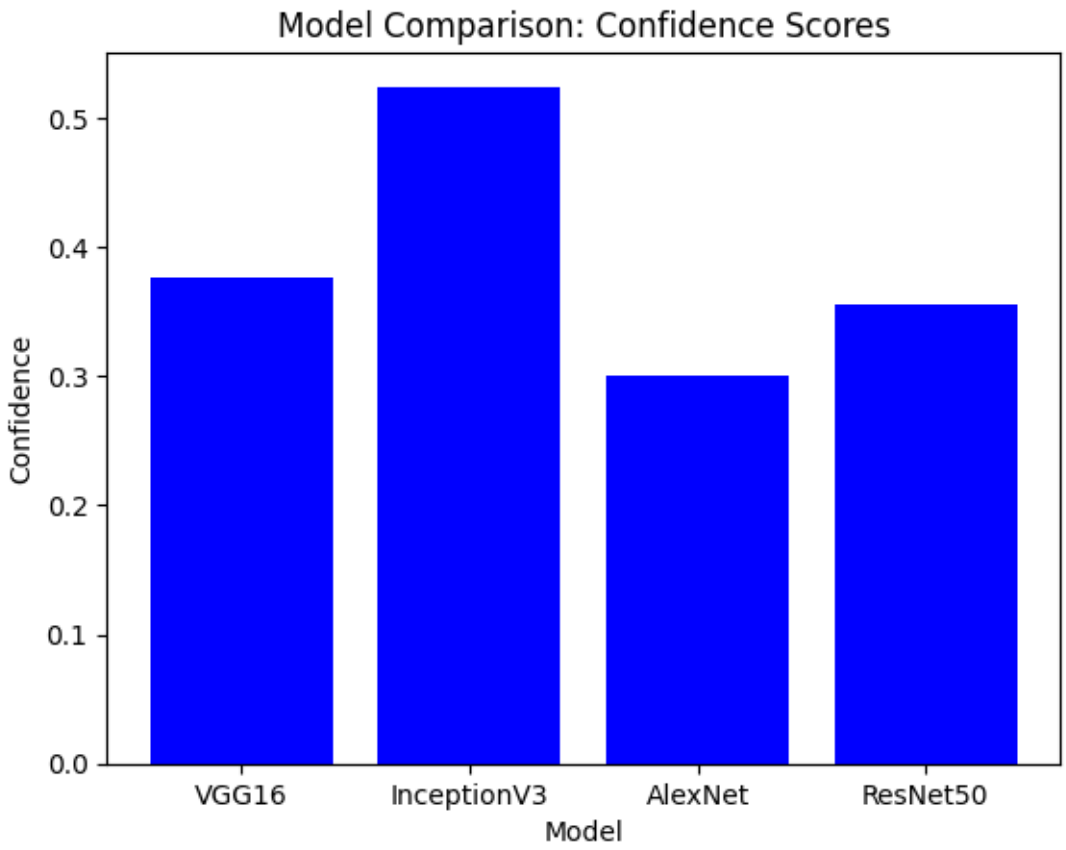
```
    'VGG16': model_vgg16,
    'InceptionV3': model_inception,
    'AlexNet': model_alexnet,
    'ResNet50': model_resnet50
}

# Compare models
compare_models(image_path1, models, class_indices)
compare_models(image_path2, models, class_indices)
compare_models(image_path3, models, class_indices)
compare_models(image_path4, models, class_indices)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:tensorflow:5 out of the last 5 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7ba6d82104c0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.

1/1            0s 380ms/step

WARNING:tensorflow:6 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7ba6d82125f0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.

1/1            3s 3s/step
```

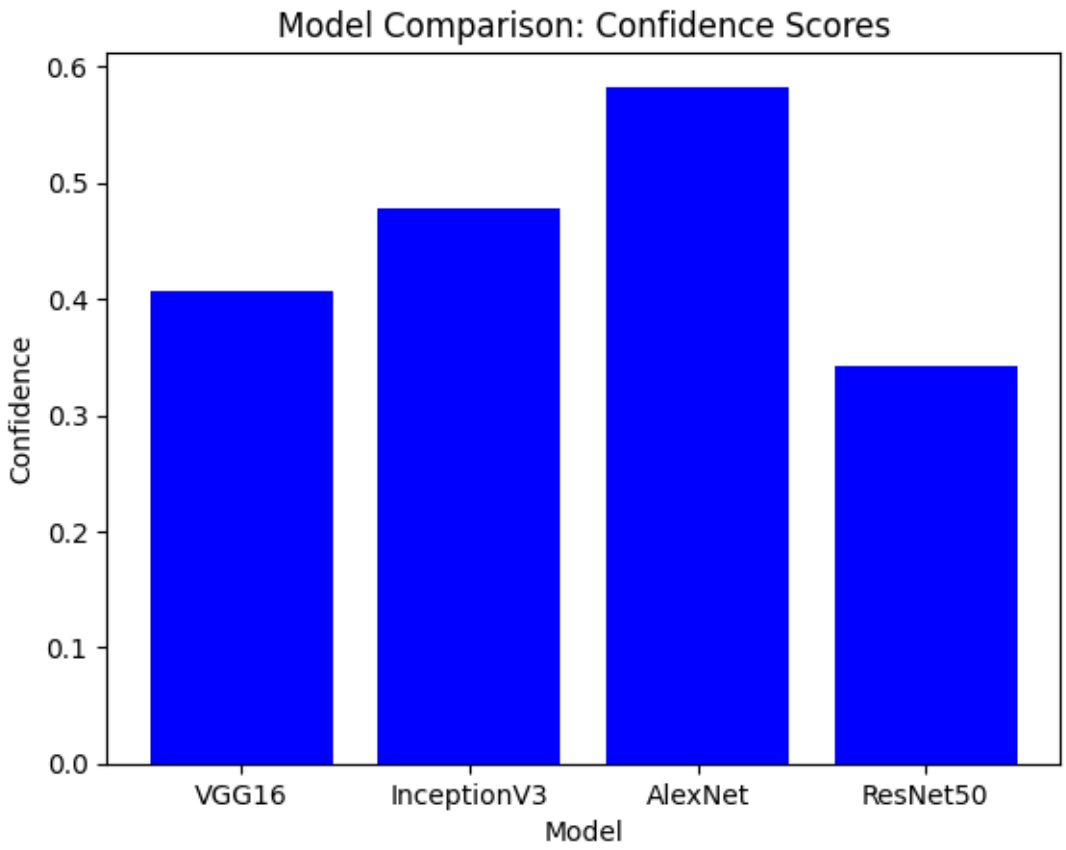
```
1/1             0s 278ms/step
1/1             2s 2s/step
```



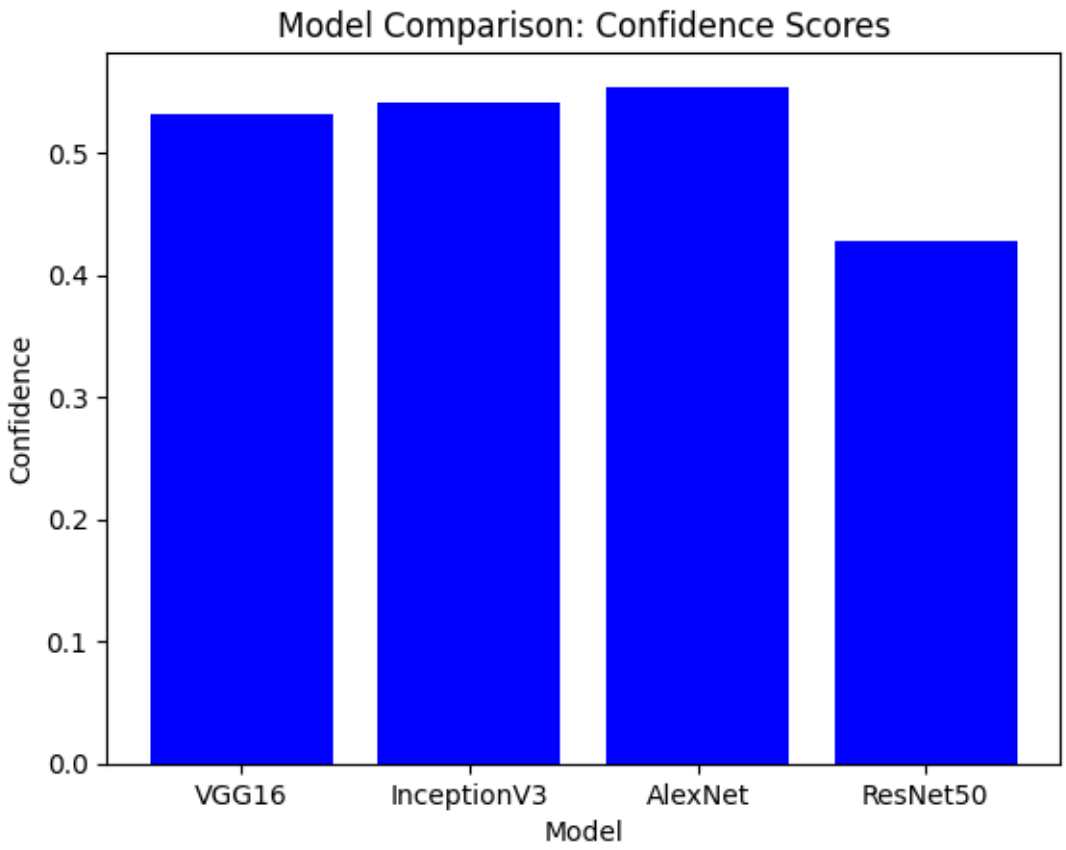

```
1/1             0s 17ms/step
1/1             0s 21ms/step
1/1             0s 19ms/step
1/1             0s 20ms/step
```




```
1/1             0s 19ms/step
1/1             0s 25ms/step
1/1             0s 18ms/step
1/1             0s 20ms/step
```




```
1/1             0s 18ms/step
1/1             0s 23ms/step
1/1             0s 17ms/step
1/1             0s 20ms/step
```



[12]:
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Load models (as before)
model_vgg16 = load_model('alzheimer_vgg16_40.h5')
model_inception = load_model('alzheimer_inception_40.h5')
model_alexnet = load_model('alzheimer_alexnet_40.h5')
model_resnet50 = load_model('alzheimer_resnet50_40.h5')

# Define the image size
img_size = (150, 150)

# Function to compare models and accumulate confidence scores
def compare_models(image_paths, models, class_indices):
    # Initialize a dictionary to store cumulative confidence scores for each␣
 ↪model
    model_confidence_scores = {model_name: [] for model_name in models.keys()}
```

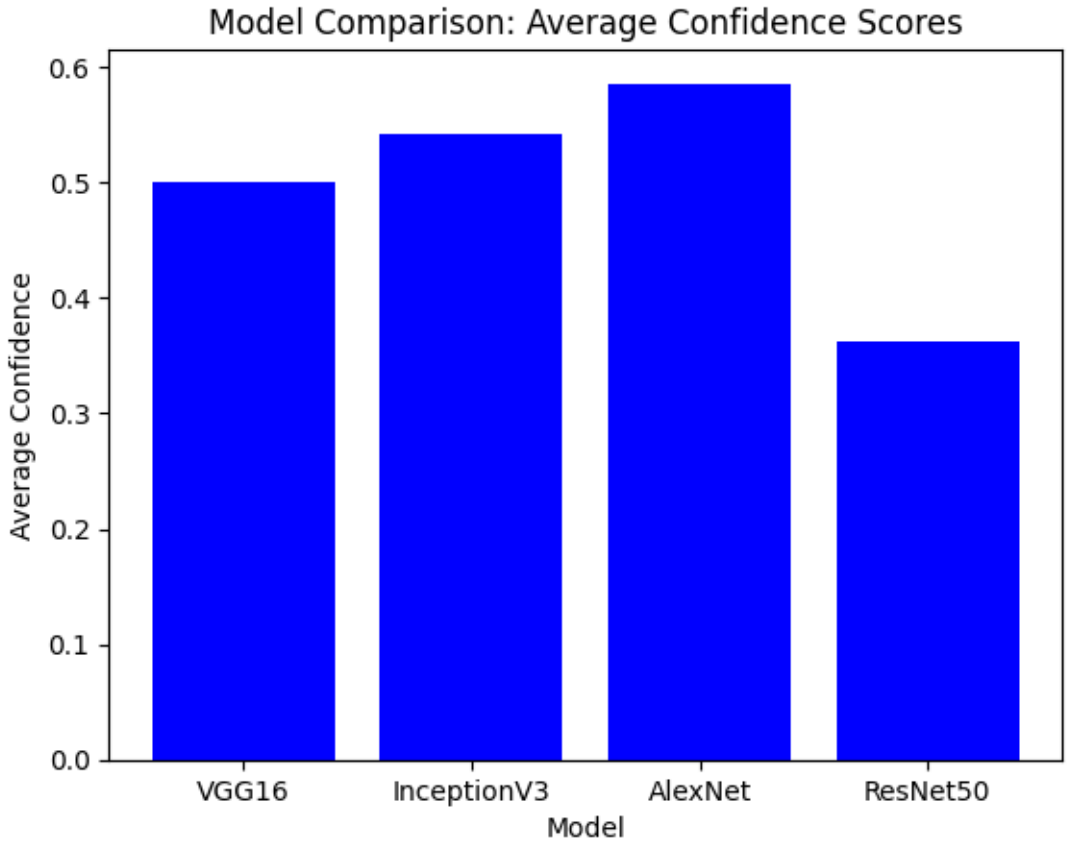
```
    for image_path in image_paths:
        # Load the image
        img = load_img(image_path, target_size=img_size)
        img_array = img_to_array(img) / 255.0
        img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

        # Initialize a dictionary to store predictions for the current image
        predictions = {}

        for model_name, model in models.items():
            pred = model.predict(img_array)
            predicted_class = np.argmax(pred, axis=1)[0]
            class_labels = {v: k for k, v in class_indices.items()}
            predicted_label = class_labels[predicted_class]
            predictions[model_name] = pred[0]  # Store the confidence scores

        # Store the maximum confidence score for each model for this image
        for model_name, score in predictions.items():
            model_confidence_scores[model_name].append(np.max(score))

    # Calculate average confidence scores for each model
    avg_confidence_scores = {model_name: np.mean(scores) for model_name, scores␣
 ↪in model_confidence_scores.items()}

    return avg_confidence_scores

# Path to the images you want to test
image_paths = [
    "/content/Combined Dataset/test/Mild Impairment/1 (10).jpg",
    "/content/Combined Dataset/test/Moderate Impairment/13 (2).jpg",
    "/content/Combined Dataset/test/No Impairment/1 (16).jpg",
    "/content/Combined Dataset/test/Very Mild Impairment/1 (10).jpg"
]

class_indices = {'Mild Impairment': 0, 'Moderate Impairment': 1, 'No␣
 ↪Impairment': 2, 'Very Mild Impairment': 3}
# List of models
models = {
    'VGG16': model_vgg16,
    'InceptionV3': model_inception,
    'AlexNet': model_alexnet,
    'ResNet50': model_resnet50
}

# Compare models and get the average confidence scores
avg_confidence_scores = compare_models(image_paths, models, class_indices)
```

```python
# Print out the average confidence scores for each model
print("Average Confidence Scores for Each Model:")
for model_name, score in avg_confidence_scores.items():
    print(f"{model_name}: {score:.4f}")

# Plot the average confidence scores for each model
model_names = list(avg_confidence_scores.keys())
confidence_scores = list(avg_confidence_scores.values())

plt.bar(model_names, confidence_scores, color='blue')
plt.title("Model Comparison: Average Confidence Scores")
plt.ylabel("Average Confidence")
plt.xlabel("Model")
plt.show()

# Identify the best model based on the highest average confidence score
best_model = max(avg_confidence_scores, key=avg_confidence_scores.get)
print(f"\nBest Model: {best_model}")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
```

```
1/1             1s 586ms/step
1/1             4s 4s/step
1/1             0s 280ms/step
1/1             2s 2s/step
1/1             0s 17ms/step
1/1             0s 21ms/step
1/1             0s 20ms/step
1/1             0s 20ms/step
1/1             0s 16ms/step
1/1             0s 19ms/step
1/1             0s 18ms/step
1/1             0s 19ms/step
1/1             0s 16ms/step
1/1             0s 21ms/step
1/1             0s 15ms/step
1/1             0s 19ms/step
```

```
Average Confidence Scores for Each Model:
VGG16: 0.5010
InceptionV3: 0.5419
AlexNet: 0.5859
ResNet50: 0.3630
```




```
Best Model: AlexNet
```