```python
from google.colab import drive
drive.mount('/content/drive')
```

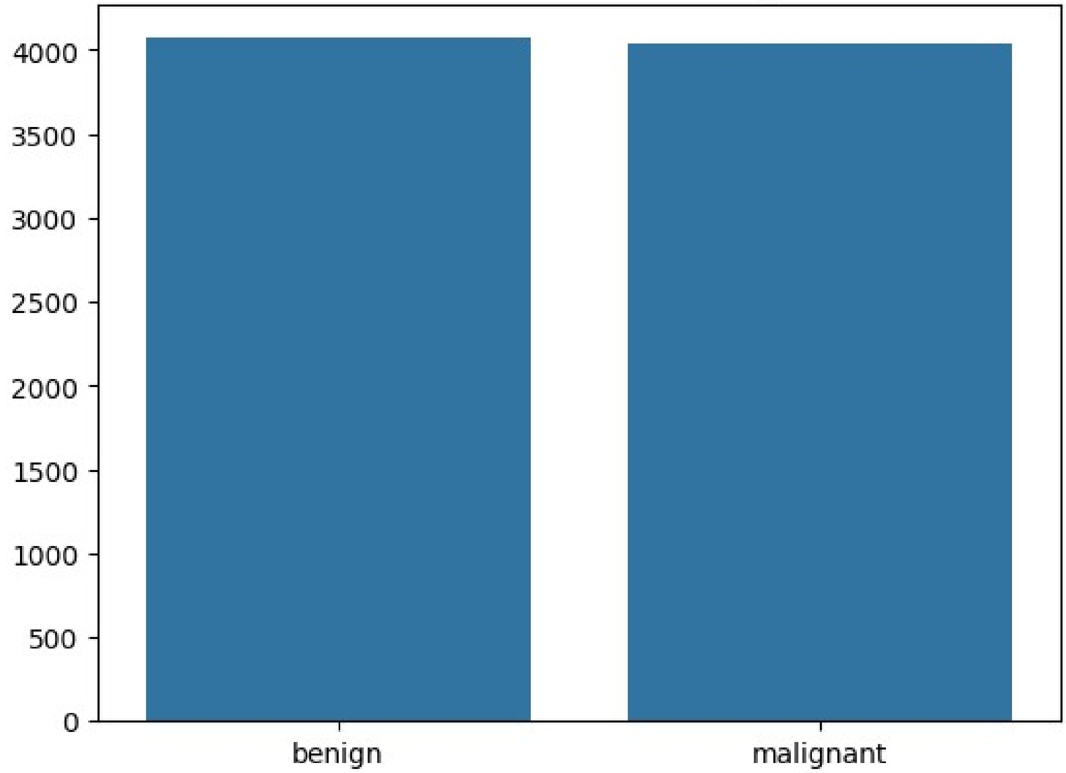
```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```python
from tensorflow.keras.applications import VGG16, EfficientNetB0
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import accuracy_score, f1_score,
classification_report, roc_curve, auc
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os
import zipfile
from sklearn.metrics import roc_auc_score

# Import necessary libraries
import os
import zipfile
from google.colab import drive

# Mount Google Drive to access the dataset
drive.mount('/content/drive', force_remount=True)

# Define path to the dataset in your Google Drive
dataset_path = "/content/drive/My
Drive/Neural_Network_Project/archive.zip"
extract_path = "/content/Neural_Network_Projects/extracted_dataset"

# Ensure the extract directory exists
os.makedirs(extract_path, exist_ok=True)

# Unzip the dataset
with zipfile.ZipFile(dataset_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("Dataset extracted successfully.")
```

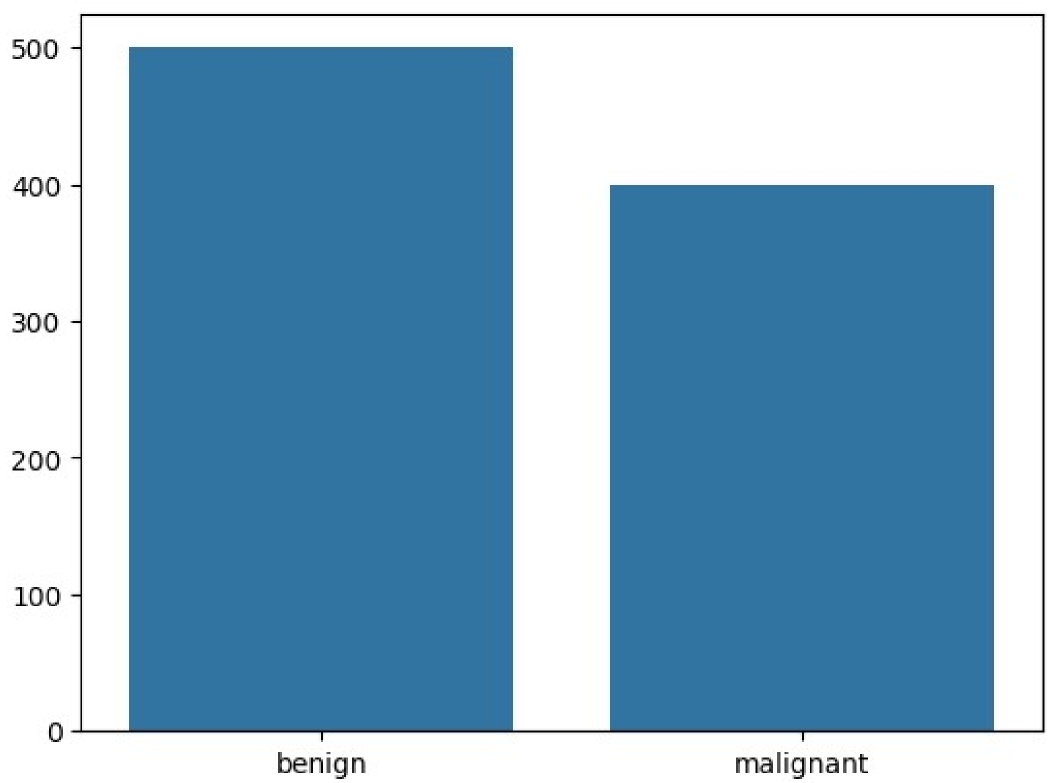
```
Mounted at /content/drive
Dataset extracted successfully.
```

```python
# Define train and validation directories
train_dir = os.path.join(extract_path, "ultrasound breast
classification/train")
val_dir = os.path.join(extract_path, "ultrasound breast
classification/val")

# EDA: Count and visualize class distributions
def count_images(folder):
    class_counts = {}
    for label in os.listdir(folder):
        class_counts[label] = len(os.listdir(os.path.join(folder,
label)))
    return class_counts

train_counts = count_images(train_dir)
val_counts = count_images(val_dir)

import seaborn as sns
import matplotlib.pyplot as plt

# Plot class distributions
sns.barplot(x=list(train_counts.keys()),
y=list(train_counts.values()))
plt.title("Class Distribution in Training Set")
plt.show()

sns.barplot(x=list(val_counts.keys()), y=list(val_counts.values()))
plt.title("Class Distribution in Validation Set")
plt.show()
```

Class Distribution in Training Set



Class Distribution in Validation Set



```python
# Data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False  # Ensure order matches for validation
)
```

```
Found 8113 images belonging to 2 classes.
Found 900 images belonging to 2 classes.
```

```python
# Model definitions
def create_cnn_model():
    model = Sequential([
        Conv2D(64, (3, 3), activation='relu', input_shape=(224, 224,
3)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    return model
```

```python
def create_vgg16_model():
    base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
    for layer in base_model.layers:
        layer.trainable = False
    model = Sequential([
        base_model,
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    return model

def create_efficientnet_model():
    base_model = EfficientNetB0(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
    for layer in base_model.layers:
        layer.trainable = False
    model = Sequential([
        base_model,
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    return model

# Compile models
def compile_model(model):
    model.compile(optimizer=Adam(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])

cnn_model = create_cnn_model()
vgg16_model = create_vgg16_model()
efficientnet_model = create_efficientnet_model()

compile_model(cnn_model)
compile_model(vgg16_model)
compile_model(efficientnet_model)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
# Training function
early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3)
model_checkpoint = ModelCheckpoint('/content/drive/My
Drive/Neural_Network_Project/best_model.keras', save_best_only=True)

def train_model(model, model_name):
    print(f"Training {model_name} model...")
    history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples //
train_generator.batch_size,
        epochs=20,
        validation_data=val_generator,
        validation_steps=val_generator.samples //
val_generator.batch_size,
        callbacks=[early_stop, reduce_lr, model_checkpoint]
    )
    return history

cnn_history = train_model(cnn_model, "CNN")
vgg16_history = train_model(vgg16_model, "VGG16")
efficientnet_history = train_model(efficientnet_model,
"EfficientNetB0")
```

```
Training CNN model...
Epoch 1/20

/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()

253/253 ━━━━━━━━━━━━━━━━━━━━ 142s 538ms/step - accuracy: 0.5945 -
loss: 0.6851 - val_accuracy: 0.7935 - val_loss: 0.4908 -
learning_rate: 1.0000e-04
Epoch 2/20
  1/253 ━━━━━━━━━━━━━━━━━━━━ 38s 151ms/step - accuracy: 0.7500 - loss:
0.5508

/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)

 253/253 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7500 - loss:
0.5508 - val_accuracy: 0.7500 - val_loss: 0.6289 - learning_rate:
1.0000e-04
Epoch 3/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 146s 561ms/step - accuracy: 0.7154 -
loss: 0.5573 - val_accuracy: 0.8170 - val_loss: 0.4797 -
learning_rate: 1.0000e-04
Epoch 4/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 9s 34ms/step - accuracy: 0.8125 - loss:
0.4504 - val_accuracy: 1.0000 - val_loss: 0.2768 - learning_rate:
1.0000e-04
Epoch 5/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 171s 468ms/step - accuracy: 0.7322 -
loss: 0.5295 - val_accuracy: 0.7467 - val_loss: 0.6756 -
learning_rate: 1.0000e-04
Epoch 6/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 0s 94us/step - accuracy: 0.6562 - loss:
0.6268 - val_accuracy: 0.7500 - val_loss: 0.6477 - learning_rate:
1.0000e-04
Epoch 7/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 119s 460ms/step - accuracy: 0.7574 -
loss: 0.5047 - val_accuracy: 0.8616 - val_loss: 0.4046 -
learning_rate: 1.0000e-04
Epoch 8/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 0s 110us/step - accuracy: 0.8438 - loss:
0.4608 - val_accuracy: 1.0000 - val_loss: 0.4979 - learning_rate:
5.0000e-05
Epoch 9/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 142s 462ms/step - accuracy: 0.7799 -
loss: 0.4632 - val_accuracy: 0.8594 - val_loss: 0.3870 -
learning_rate: 5.0000e-05
Training VGG16 model...
Epoch 1/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 130s 492ms/step - accuracy: 0.6829 -
loss: 0.6078 - val_accuracy: 0.8013 - val_loss: 0.3848 -
learning_rate: 1.0000e-04
Epoch 2/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8438 - loss:
0.3642 - val_accuracy: 1.0000 - val_loss: 0.1463 - learning_rate:
1.0000e-04
Epoch 3/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 128s 496ms/step - accuracy: 0.7873 -
loss: 0.4526 - val_accuracy: 0.7991 - val_loss: 0.3904 -
learning_rate: 1.0000e-04
Epoch 4/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 0s 209us/step - accuracy: 0.9062 - loss:
0.2980 - val_accuracy: 1.0000 - val_loss: 0.2093 - learning_rate:
```

```
1.0000e-04
Epoch 5/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 126s 482ms/step - accuracy: 0.8045 -
loss: 0.4192 - val_accuracy: 0.8136 - val_loss: 0.4082 -
learning_rate: 1.0000e-04
Epoch 6/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 0s 217us/step - accuracy: 0.9375 - loss:
0.2542 - val_accuracy: 1.0000 - val_loss: 0.2341 - learning_rate:
5.0000e-05
Epoch 7/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 126s 489ms/step - accuracy: 0.8262 -
loss: 0.3883 - val_accuracy: 0.8125 - val_loss: 0.3561 -
learning_rate: 5.0000e-05
Training EfficientNetB0 model...
Epoch 1/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 152s 507ms/step - accuracy: 0.4991 -
loss: 0.9665 - val_accuracy: 0.5580 - val_loss: 0.6931 -
learning_rate: 1.0000e-04
Epoch 2/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - accuracy: 0.4375 - loss:
0.6932 - val_accuracy: 0.0000e+00 - val_loss: 0.6934 - learning_rate:
1.0000e-04
Epoch 3/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 175s 455ms/step - accuracy: 0.5070 -
loss: 0.6955 - val_accuracy: 0.4420 - val_loss: 0.7138 -
learning_rate: 1.0000e-04
Epoch 4/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 0s 144us/step - accuracy: 0.5312 - loss:
0.6886 - val_accuracy: 1.0000 - val_loss: 0.5497 - learning_rate:
1.0000e-04
Epoch 5/20
253/253 ━━━━━━━━━━━━━━━━━━━━ 114s 442ms/step - accuracy: 0.5050 -
loss: 0.6953 - val_accuracy: 0.5580 - val_loss: 0.6931 -
learning_rate: 1.0000e-04
```

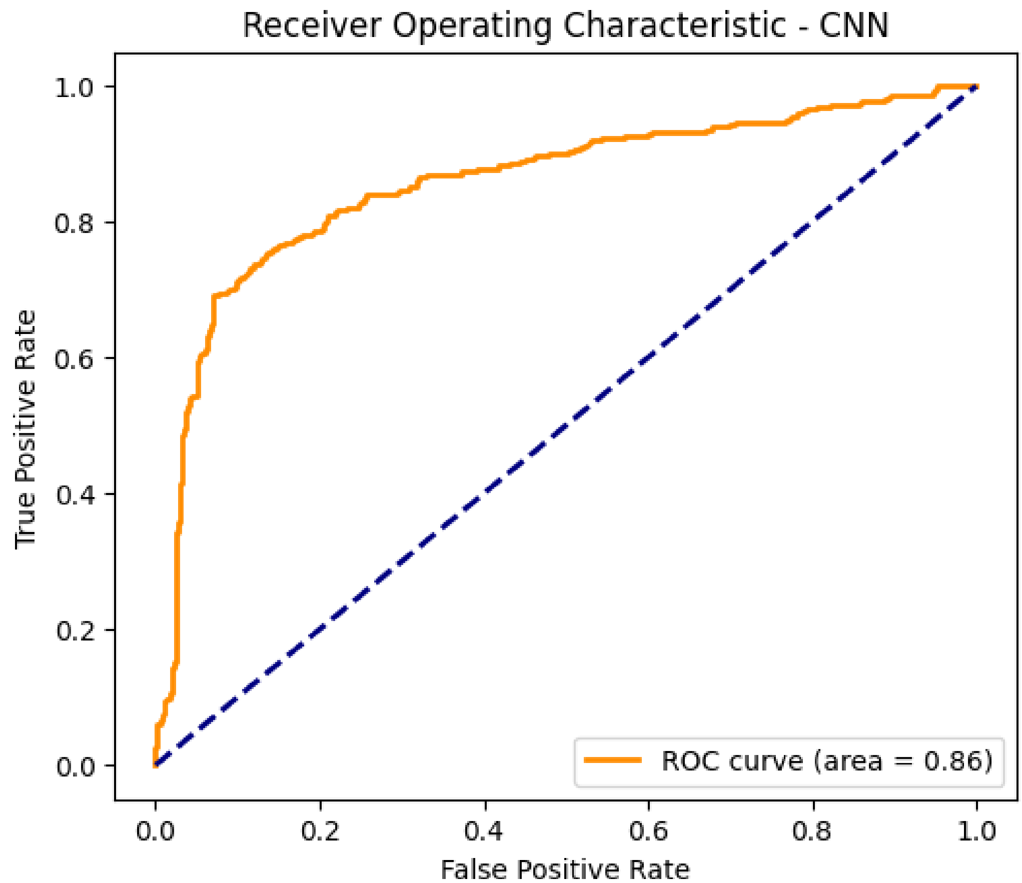
```python
# Evaluate models
cnn_val_acc = cnn_model.evaluate(val_generator)[1]
vgg16_val_acc = vgg16_model.evaluate(val_generator)[1]
efficientnet_val_acc = efficientnet_model.evaluate(val_generator)[1]

print(f"CNN Validation Accuracy: {cnn_val_acc * 100:.2f}%")
print(f"VGG16 Validation Accuracy: {vgg16_val_acc * 100:.2f}%")
print(f"EfficientNetB0 Validation Accuracy: {efficientnet_val_acc *
100:.2f}%")
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - accuracy: 0.8770 - loss:
0.3327
29/29 ━━━━━━━━━━━━━━━━━━━━ 4s 130ms/step - accuracy: 0.8349 - loss:
0.3328
29/29 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - accuracy: 0.8542 - loss:
```

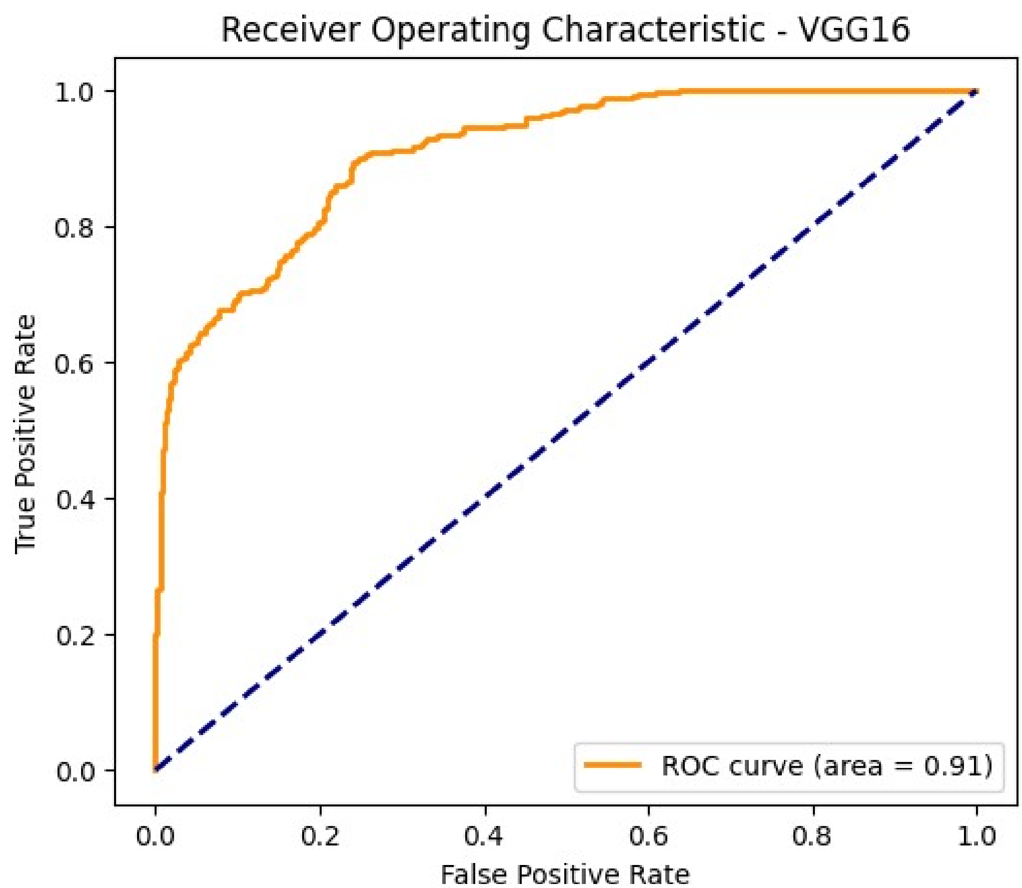
```
0.6930
CNN Validation Accuracy: 80.78%
VGG16 Validation Accuracy: 80.22%
EfficientNetB0 Validation Accuracy: 55.56%
```

```
# Ensure shuffle is False in val_generator
val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)

# Plot ROC curve
from sklearn.metrics import roc_curve, auc

def plot_roc_curve(model, model_name, val_generator):
    y_true = val_generator.classes  # True labels
    y_pred = model.predict(val_generator, verbose=1).ravel()  #
Predicted probabilities
    fpr, tpr, _ = roc_curve(y_true, y_pred)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6, 5))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve
(area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver Operating Characteristic - {model_name}')
    plt.legend(loc='lower right')
    plt.show()

# Call this function for each model
plot_roc_curve(cnn_model, "CNN", val_generator)
plot_roc_curve(vgg16_model, "VGG16", val_generator)
plot_roc_curve(efficientnet_model, "EfficientNetB0", val_generator)
```

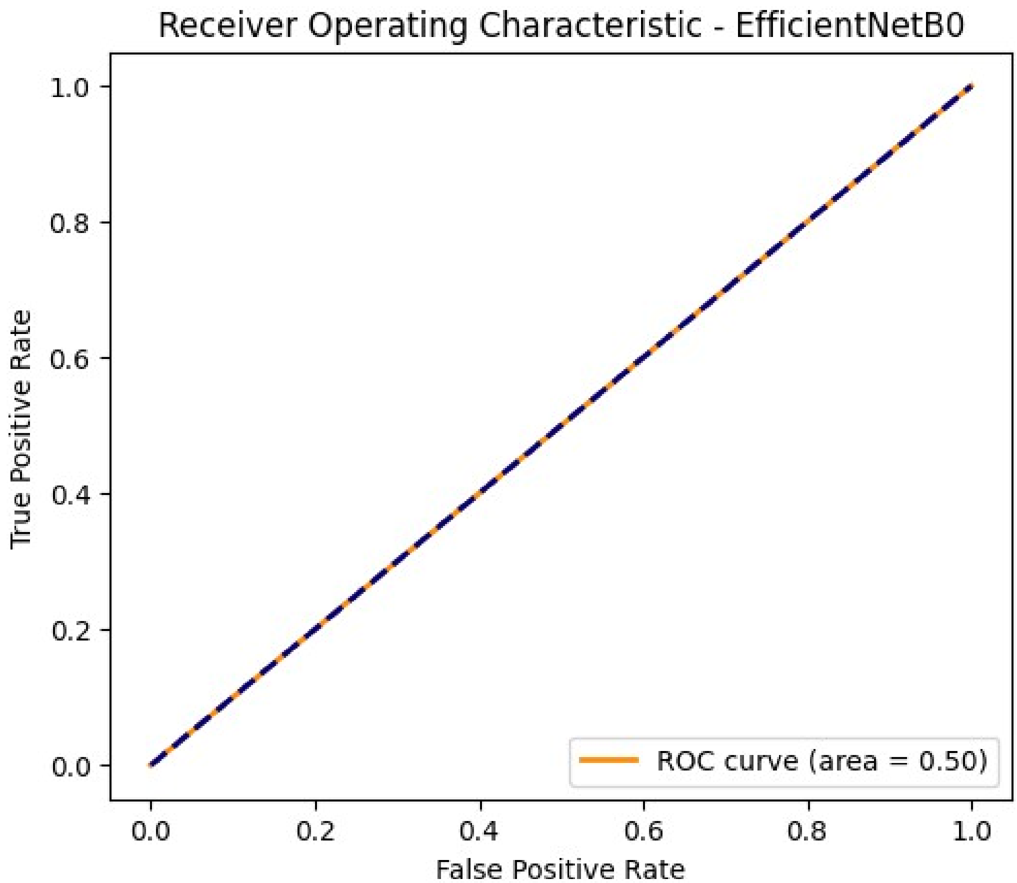
```
Found 900 images belonging to 2 classes.
29/29 ━━━━━━━━━━━━━━━━━━━━ 2s 74ms/step
```




```
29/29 ━━━━━━━━━━━━━━━━━━━━ 5s 147ms/step
```

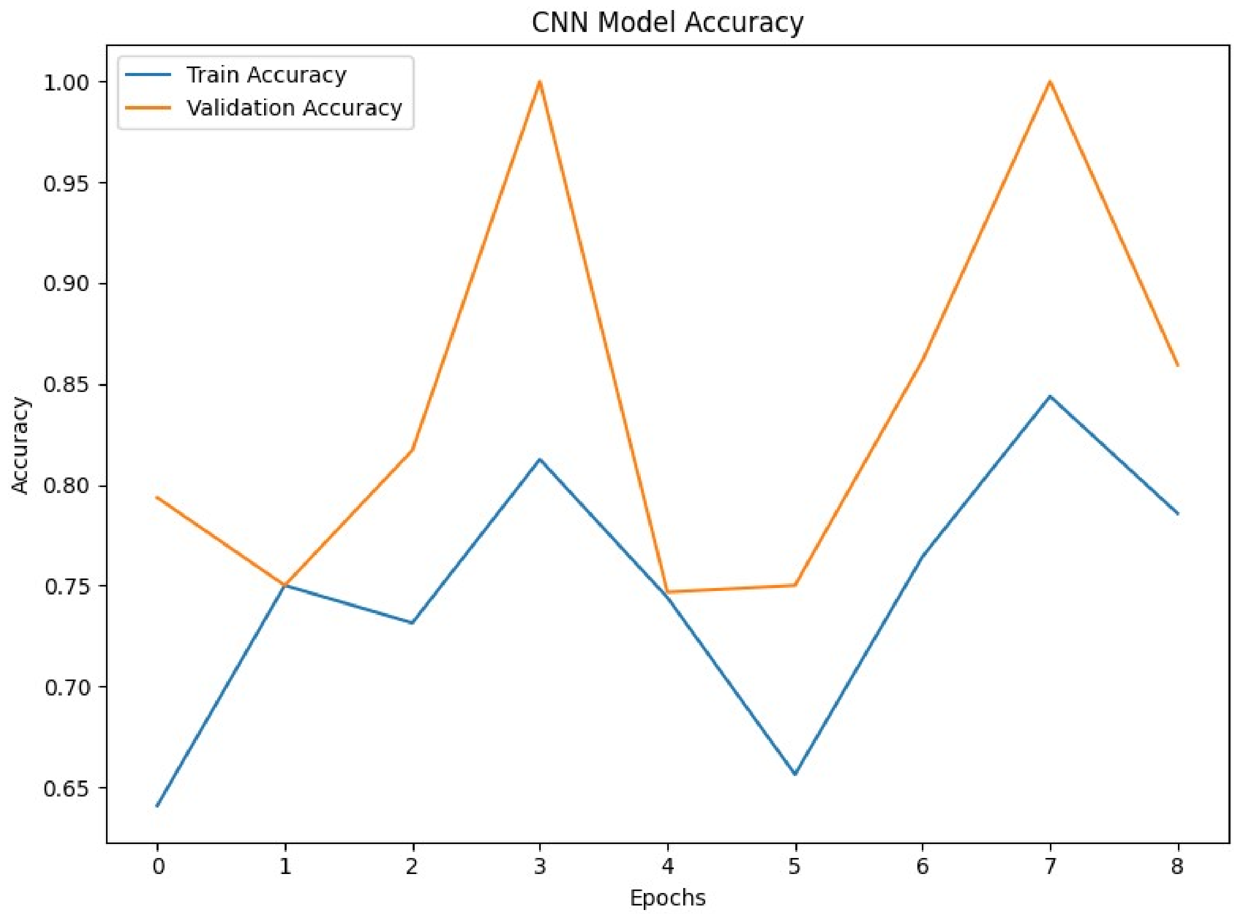


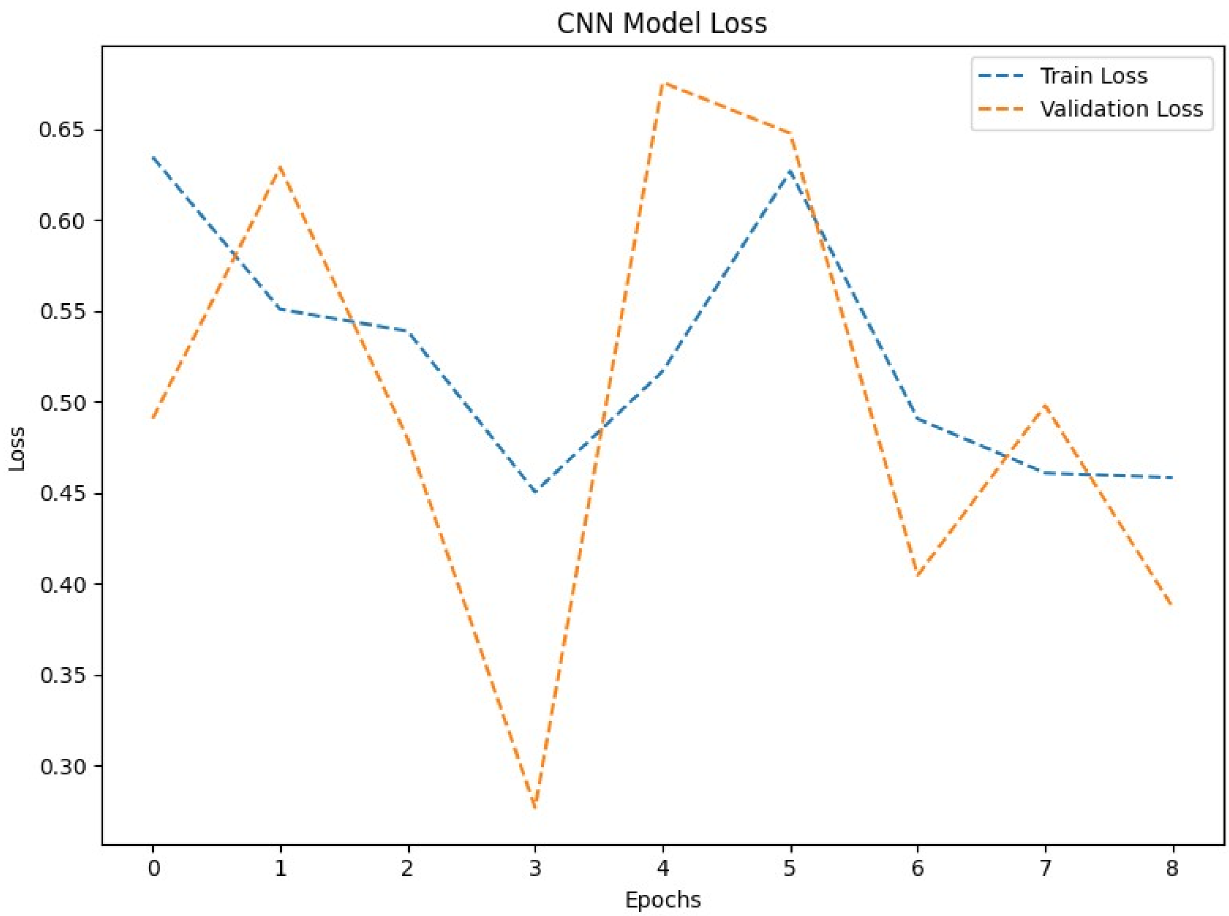

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 13s 246ms/step
```




```
# Clear any previous figure to avoid overriding issues
plt.clf()

# CNN Model Accuracy Plot
plt.figure(figsize=(8, 6))
plt.plot(cnn_history.history['accuracy'], label='Train Accuracy')
plt.plot(cnn_history.history['val_accuracy'], label='Validation
Accuracy')
plt.title('CNN Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()

# CNN Model Loss Plot
plt.figure(figsize=(8, 6))
plt.plot(cnn_history.history['loss'], label='Train Loss',
linestyle='dashed')
plt.plot(cnn_history.history['val_loss'], label='Validation Loss',
linestyle='dashed')
plt.title('CNN Model Loss')
```

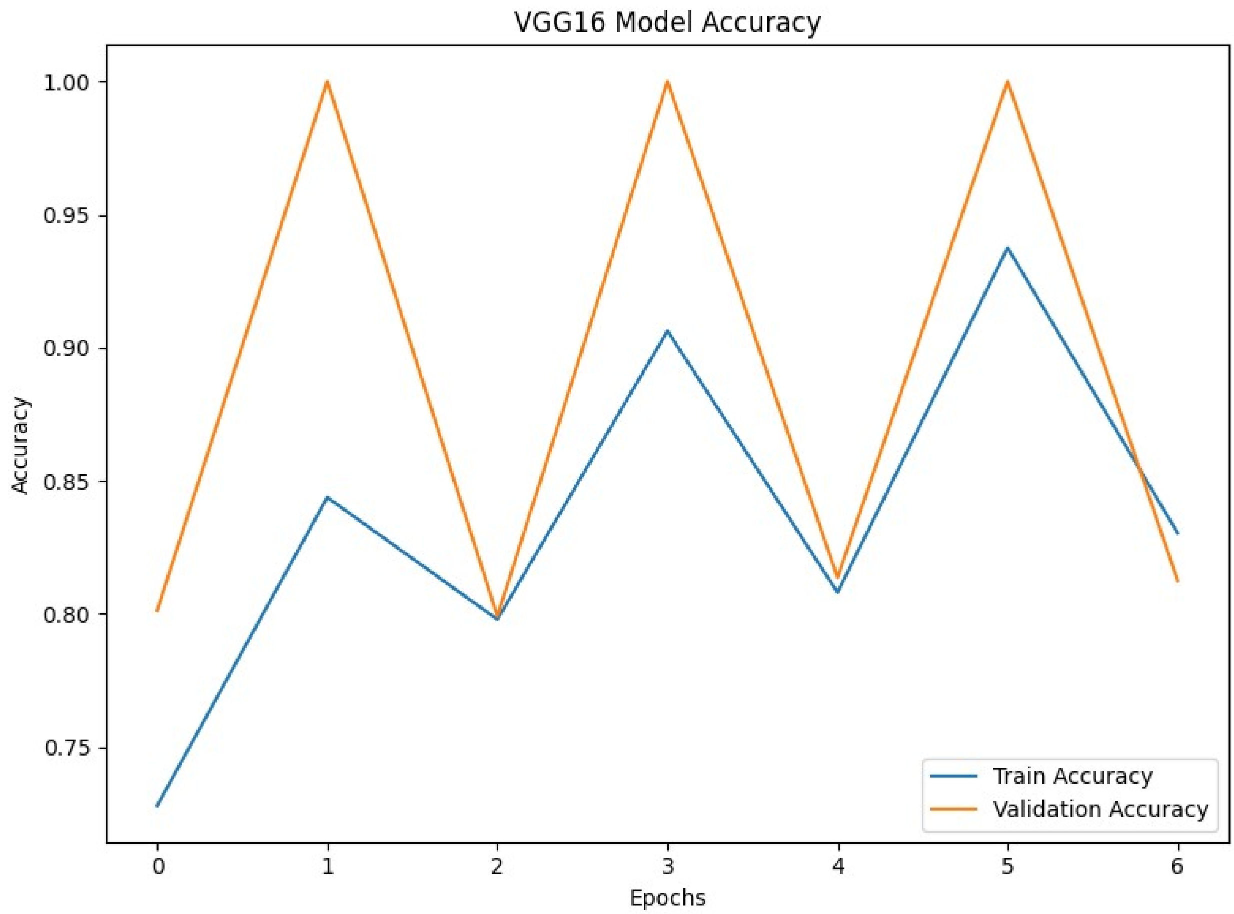
```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()

<Figure size 640x480 with 0 Axes>
```

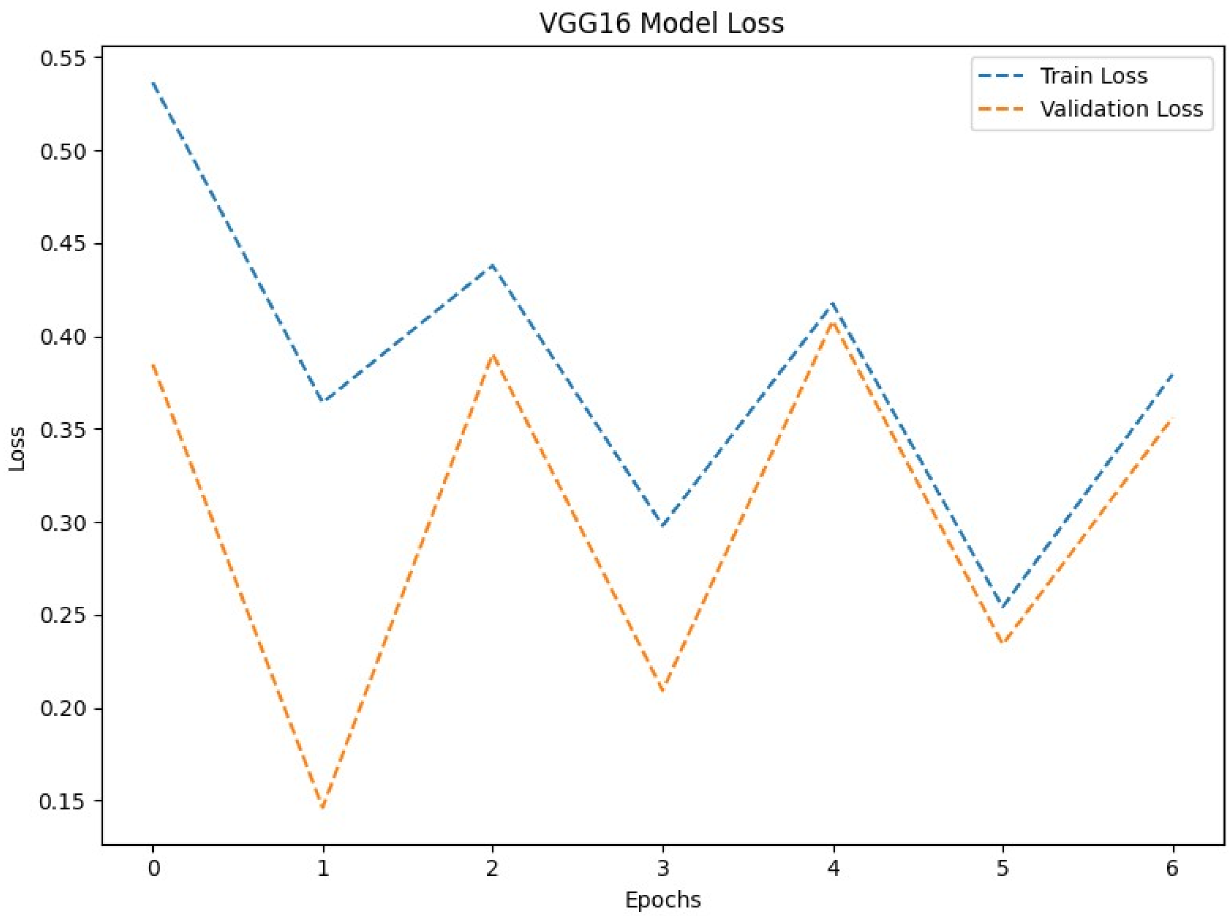





```
# VGG16 Model Accuracy Plot
plt.figure(figsize=(8, 6))
plt.plot(vgg16_history.history['accuracy'], label='Train Accuracy')
plt.plot(vgg16_history.history['val_accuracy'], label='Validation
Accuracy')
plt.title('VGG16 Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()

# VGG16 Model Loss Plot
plt.figure(figsize=(8, 6))
plt.plot(vgg16_history.history['loss'], label='Train Loss',
linestyle='dashed')
plt.plot(vgg16_history.history['val_loss'], label='Validation Loss',
linestyle='dashed')
plt.title('VGG16 Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

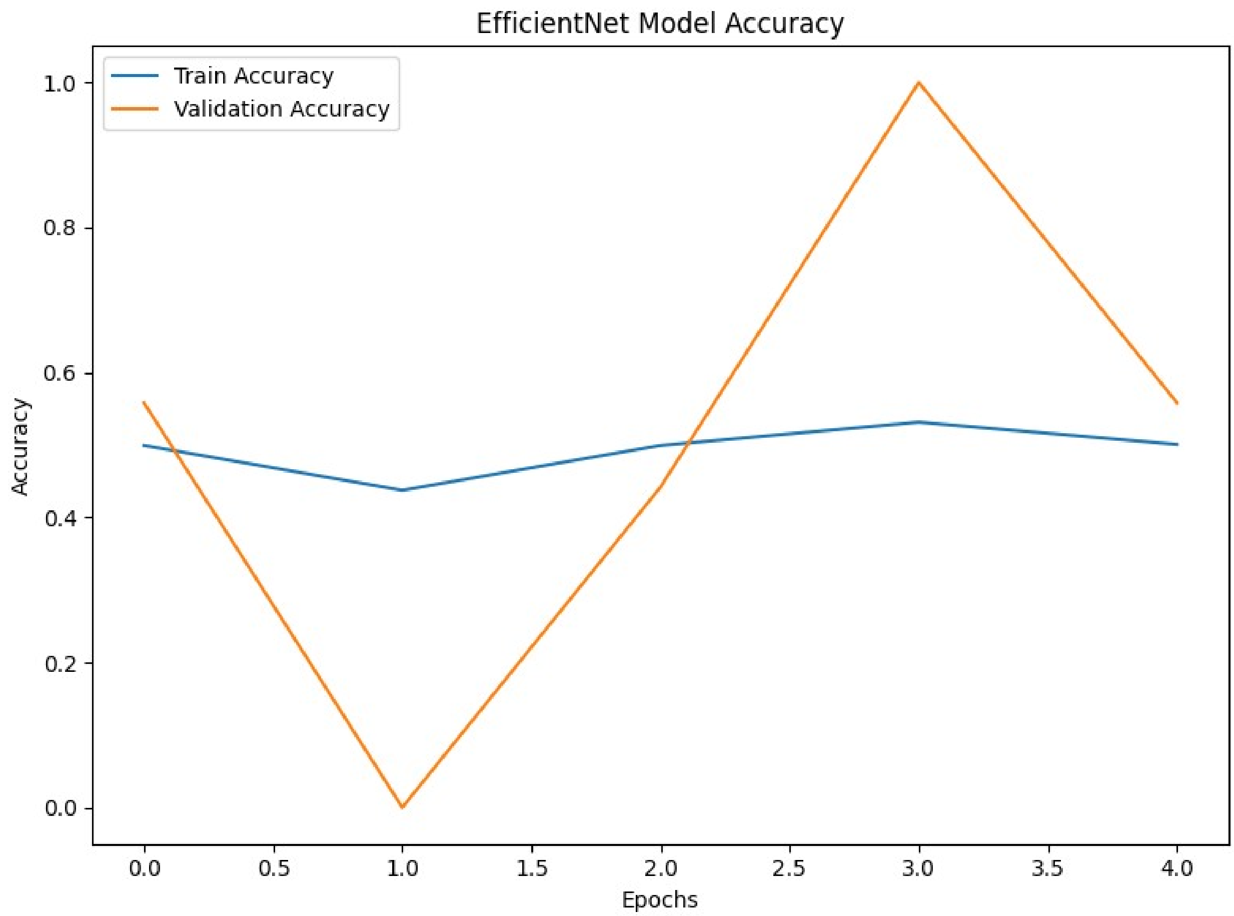
```
plt.legend()
plt.tight_layout()
plt.show()
```

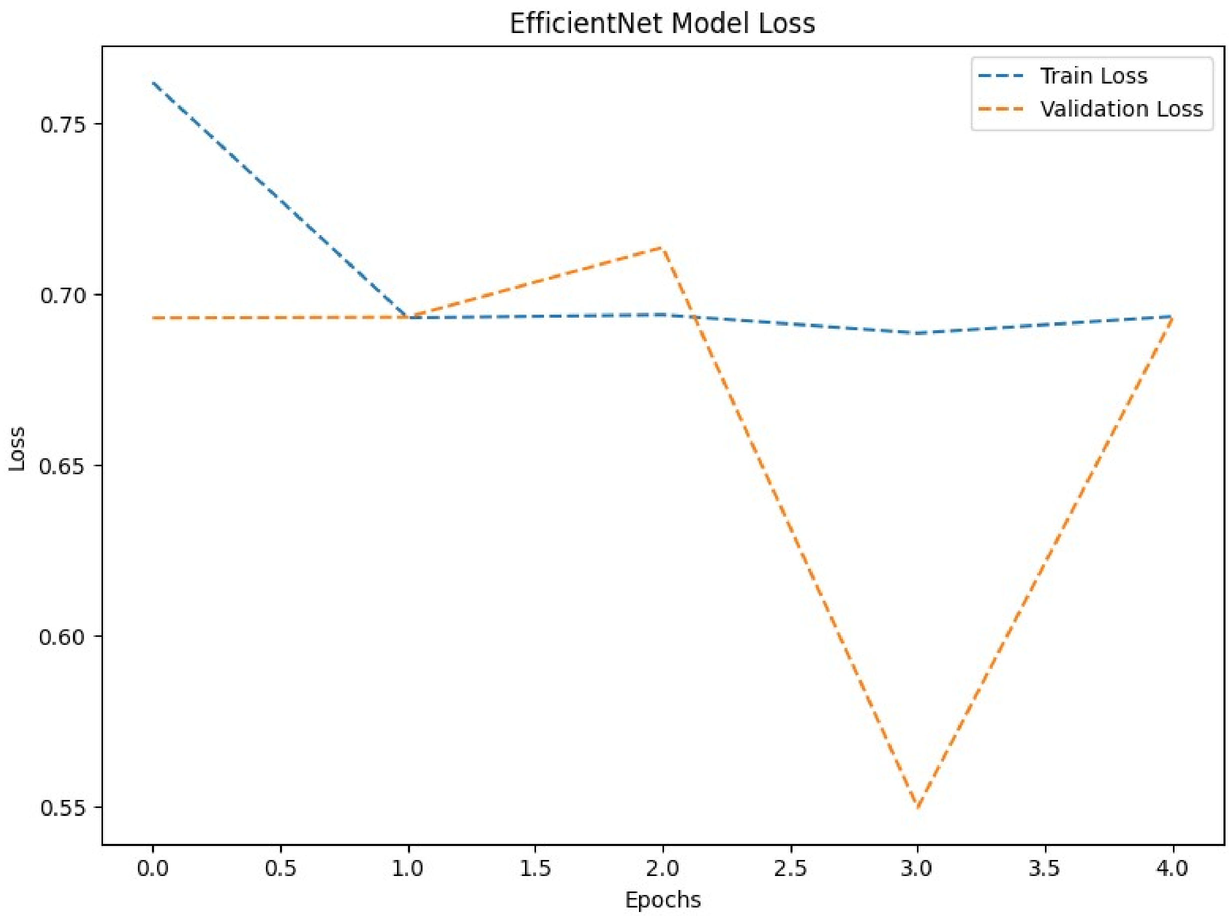





```
# EfficientNet Model Accuracy Plot
plt.figure(figsize=(8, 6))
plt.plot(efficientnet_history.history['accuracy'], label='Train
Accuracy')
plt.plot(efficientnet_history.history['val_accuracy'],
label='Validation Accuracy')
plt.title('EfficientNet Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()

# EfficientNet Model Loss Plot
plt.figure(figsize=(8, 6))
plt.plot(efficientnet_history.history['loss'], label='Train Loss',
linestyle='dashed')
plt.plot(efficientnet_history.history['val_loss'], label='Validation
Loss', linestyle='dashed')
plt.title('EfficientNet Model Loss')
plt.xlabel('Epochs')
```

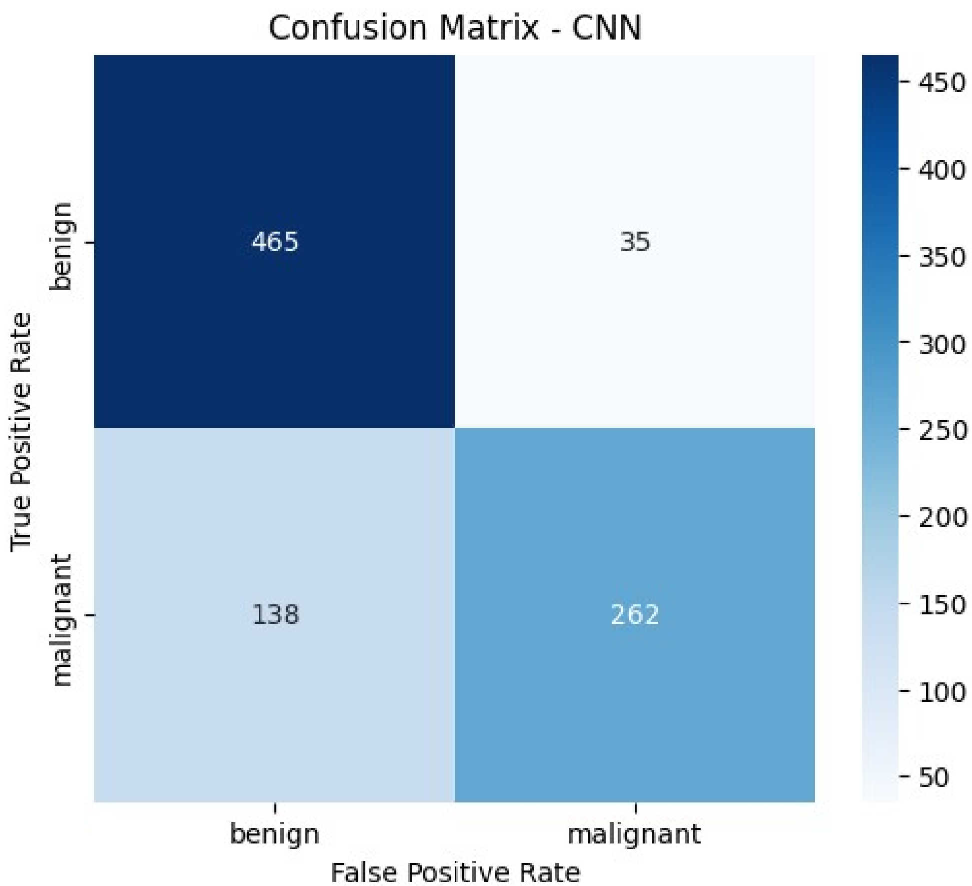
```
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```



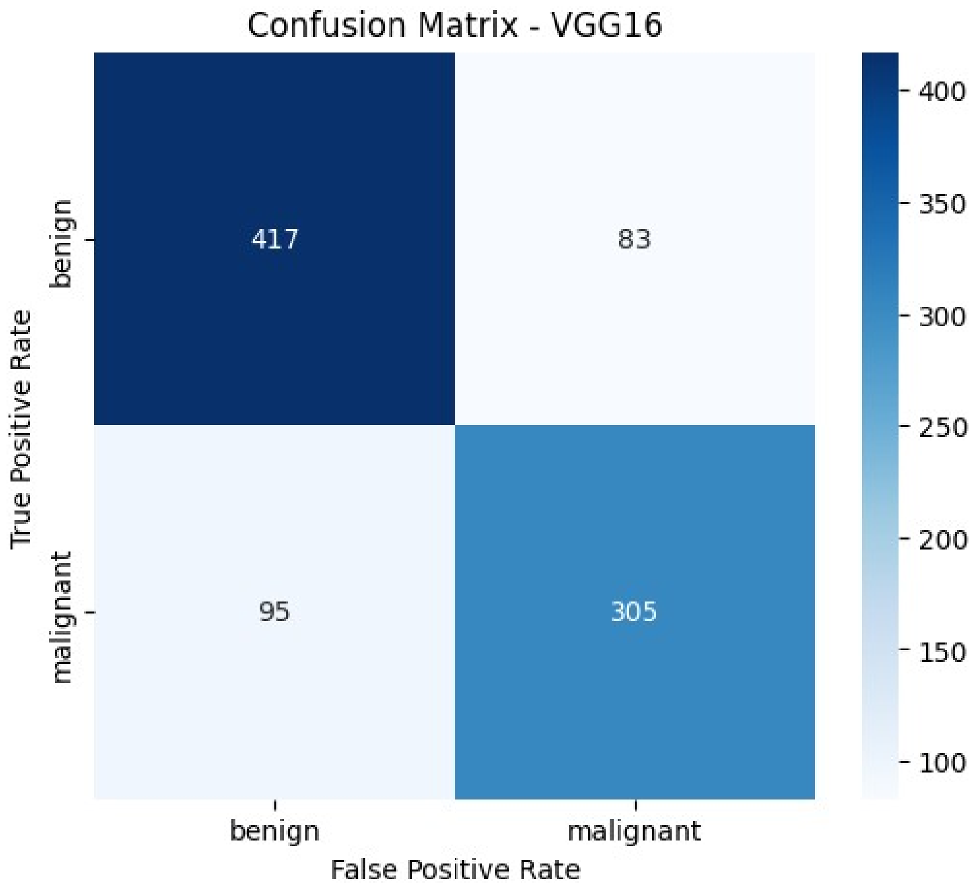



```python
# Confusion Matrix
def plot_confusion_matrix(model, model_name):
    y_true = val_generator.classes  # True labels
    y_pred = model.predict(val_generator, verbose=1).ravel()  #
Predicted probabilities
    y_pred_binary = (y_pred > 0.5).astype(int)  # Convert
probabilities to binary

    cm = confusion_matrix(y_true, y_pred_binary)

    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=val_generator.class_indices.keys(),
                yticklabels=val_generator.class_indices.keys())
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.show()

plot_confusion_matrix(cnn_model, "CNN")
plot_confusion_matrix(vgg16_model, "VGG16")
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step
```




```
29/29 ━━━━━━━━━━━━━━━━━━━━ 4s 123ms/step
```




```python
# Models list and their names
models = [cnn_model, vgg16_model, efficientnet_model]
model_names = ["CNN", "VGG16", "EfficientNetB0"]

# Calculate validation accuracies for all models
val_accuracies = []

for model, name in zip(models, model_names):
    # Evaluate the model on the validation data
    loss, accuracy = model.evaluate(val_generator, verbose=0)
    val_accuracies.append(accuracy)


# Identify the best model
best_model_index = np.argmax(val_accuracies)
best_model = models[best_model_index]
best_model_name = model_names[best_model_index]

print(f"The best-performing model is {best_model_name} with a
validation accuracy of {val_accuracies[best_model_index]:.2f}")

# Save the best model in .h5 format
```

```python
best_model.save(f"best_model_{best_model_name}.h5")
print(f"The best model ({best_model_name}) has been saved as 'best_model_{best_model_name}.h5'")
```

```
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')`.

The best-performing model is CNN with a validation accuracy of 0.81
The best model (CNN) has been saved as 'best_model_CNN.h5'
```