```
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_
```

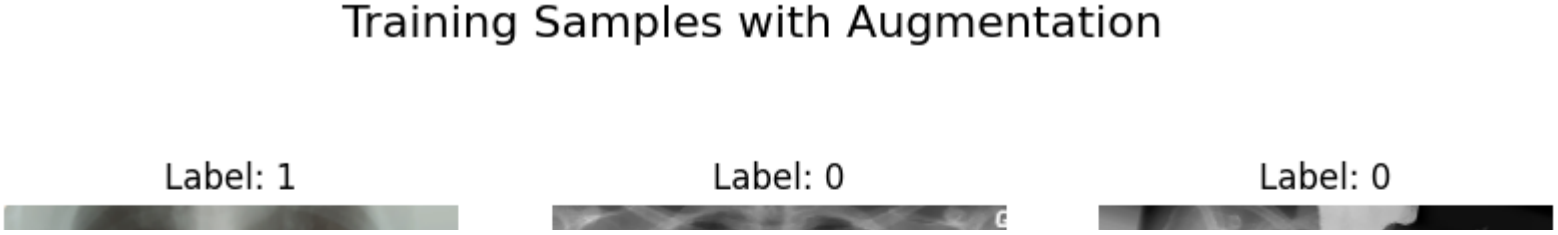
```
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.layers import Dropout, Dense, GlobalAveragePooling2D, Conv2D, MaxPooling2D, Flatten, BatchNo
from tensorflow.keras.models import Model, Sequential
from sklearn.metrics import classification_report, accuracy_score
from sklearn.utils import compute_class_weight
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
from imblearn.over_sampling import SMOTE



base_dir = "/content/drive/MyDrive/TB_Chest_Radiography_Database"


train_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=20,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    brightness_range=[0.8, 1.2],
    horizontal_flip=True,
    validation_split=0.2
```

```
                _ .
)

val_datagen = ImageDataGenerator(rescale=1.0 / 255.0, validation_split=0.2)


train_data = train_datagen.flow_from_directory(
    base_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode="binary",
    subset="training"
)

val_data = val_datagen.flow_from_directory(
    base_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode="binary",
    shuffle=False,
    subset="validation"
)


class_weights = compute_class_weight('balanced', classes=np.unique(train_data.classes), y=train_data.classes)
class_weights = {i: class_weights[i] for i in range(len(class_weights))}


def visualize_samples(data_generator, title):
    images, labels = next(data_generator)
    plt.figure(figsize=(10, 10))
    for i in range(9):
        plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])
        plt.title(f"Label: {int(labels[i])}")
        plt.axis("off")
    plt.suptitle(title, fontsize=16)
    plt.show()
```

```
    plt.show()

print("Visualizing some training samples after augmentation...")
visualize_samples(train_data, "Training Samples with Augmentation")

print("Visualizing some validation samples...")
visualize_samples(val_data, "Validation Samples")


train_classes = train_data.classes
val_classes = val_data.classes

def plot_class_distribution(classes, title):
    unique, counts = np.unique(classes, return_counts=True)
    plt.figure(figsize=(8, 5))
    sns.barplot(x=unique, y=counts, palette="viridis")
    plt.title(title, fontsize=14)
    plt.xlabel("Class", fontsize=12)
    plt.ylabel("Number of Samples", fontsize=12)
    plt.xticks(ticks=unique, labels=["Normal", "Tuberculosis"])
    plt.show()
    return unique, counts

print("Class distribution BEFORE handling imbalance:")
train_unique, train_counts = plot_class_distribution(train_classes, "Training Class Distribution (Before Balancing)
val_unique, val_counts = plot_class_distribution(val_classes, "Validation Class Distribution")


X_train = []
y_train = []
for _ in range(len(train_data)):
    images, labels = next(train_data)
    X_train.append(images)
    y_train.append(labels)

X_train = np.concatenate(X_train, axis=0).reshape(-1, 224 * 224 * 3)
y_train = np.concatenate(y_train, axis=0)
```

```
104
105
106
107 smote = SMOTE(random_state=42)
108 X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
109
110
111 X_train_res_images = X_train_res.reshape(-1, 224, 224, 3)
112
113
114 print("\nClass distribution AFTER handling imbalance with SMOTE:")
115 train_res_unique, train_res_counts = plot_class_distribution(y_train_res, "Training Class Distribution (After Balar
116
117
118 def visualize_smote_samples(X_res, y_res, title):
119     plt.figure(figsize=(10, 10))
120     for i in range(9):
121         plt.subplot(3, 3, i + 1)
122         plt.imshow(X_res[i])
123         plt.title(f"Label: {int(y_res[i])}")
124         plt.axis("off")
125     plt.suptitle(title, fontsize=16)
126     plt.show()
127
128 print("Visualizing some synthetic samples after SMOTE...")
129 visualize_smote_samples(X_train_res_images, y_train_res, "Synthetic Samples After SMOTE")
130
```

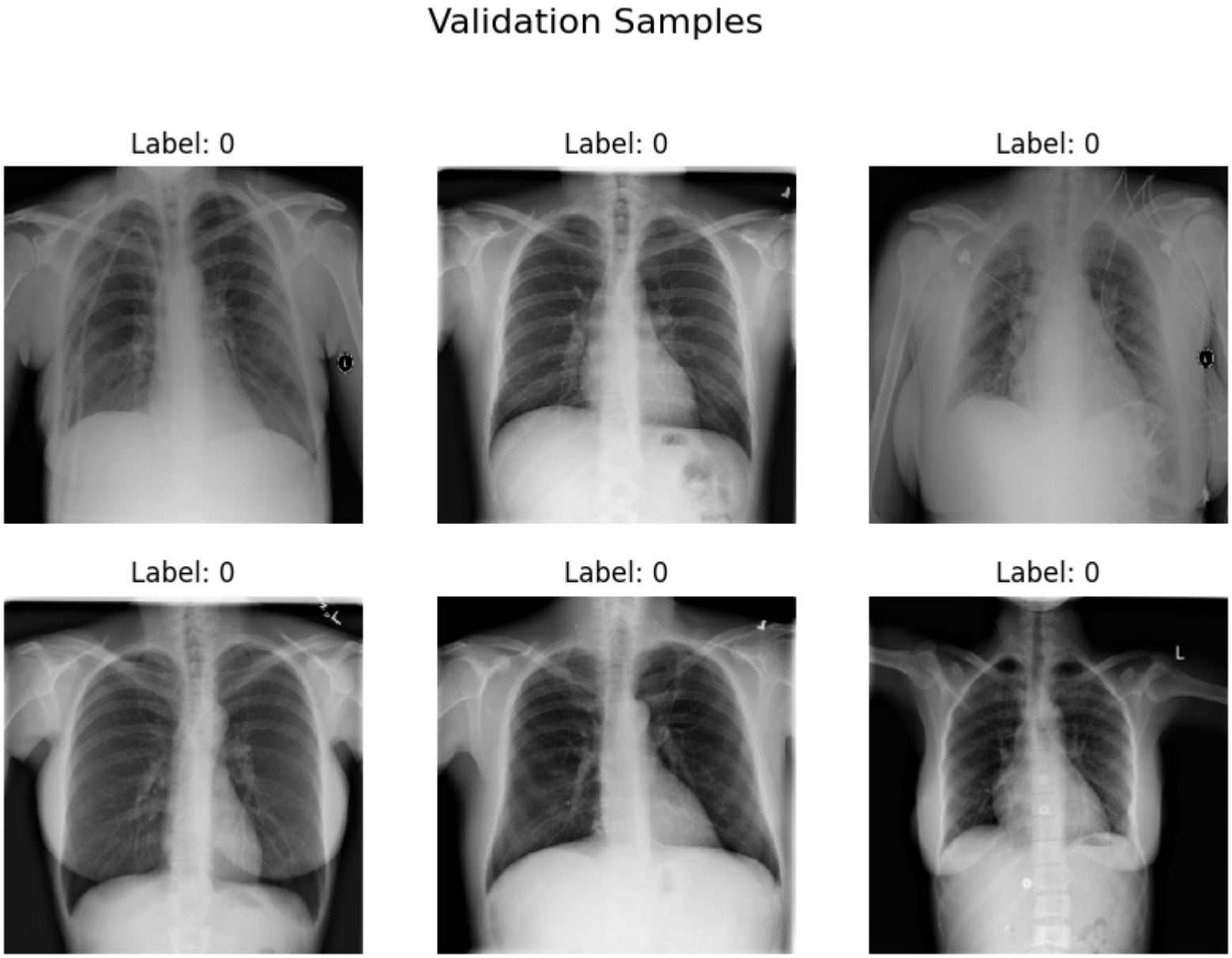
```
Found 3360 images belonging to 2 classes.
Found 840 images belonging to 2 classes.
Visualizing some training samples after augmentation...
```



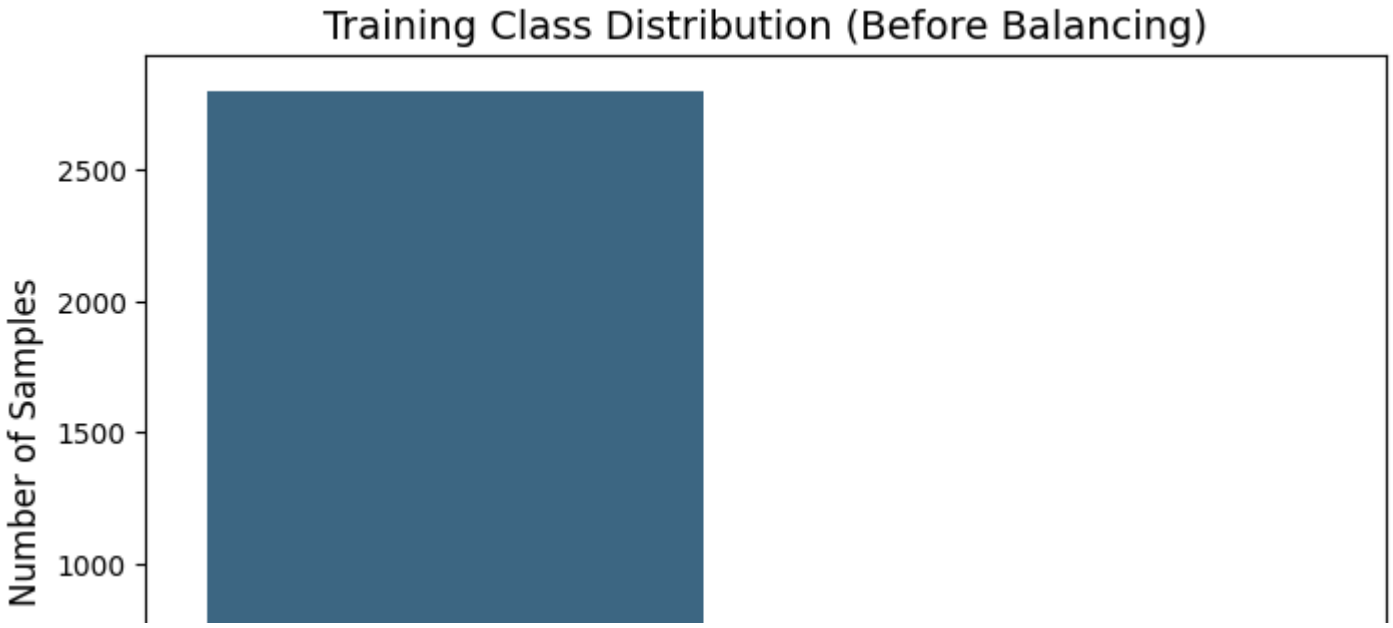
Label: 0 Label: 1 Label: 1

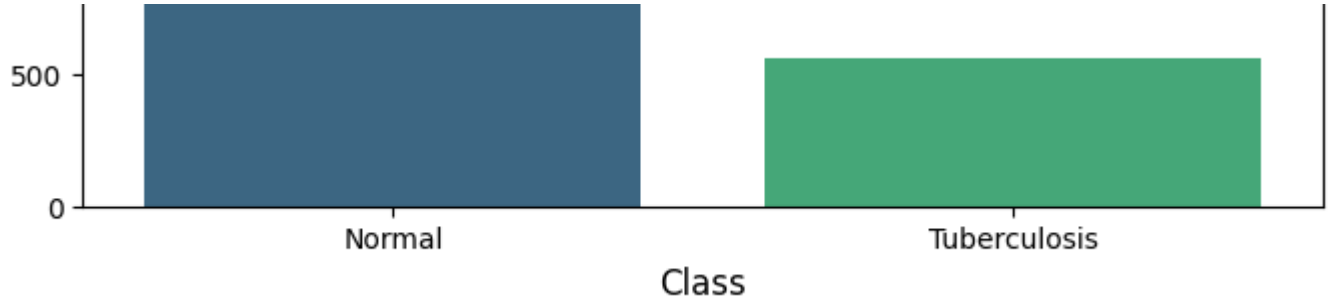
Label: 0 Label: 0 Label: 0

```
Visualizing some validation samples...
```

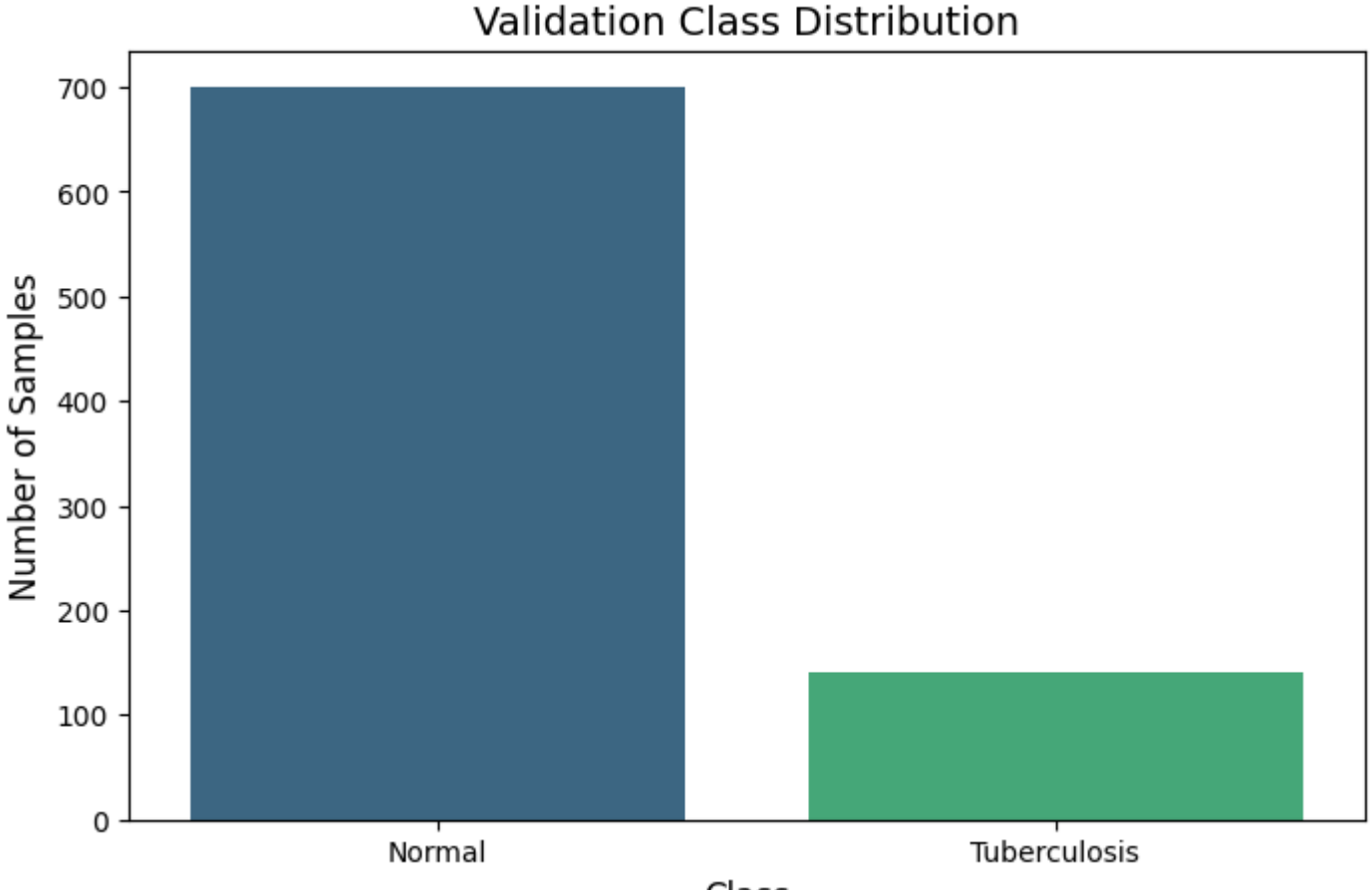



```
Class distribution BEFORE handling imbalance:
<ipython-input-3-7349cddcffdc>:82: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable tc

  sns.barplot(x=unique, y=counts, palette="viridis")
```



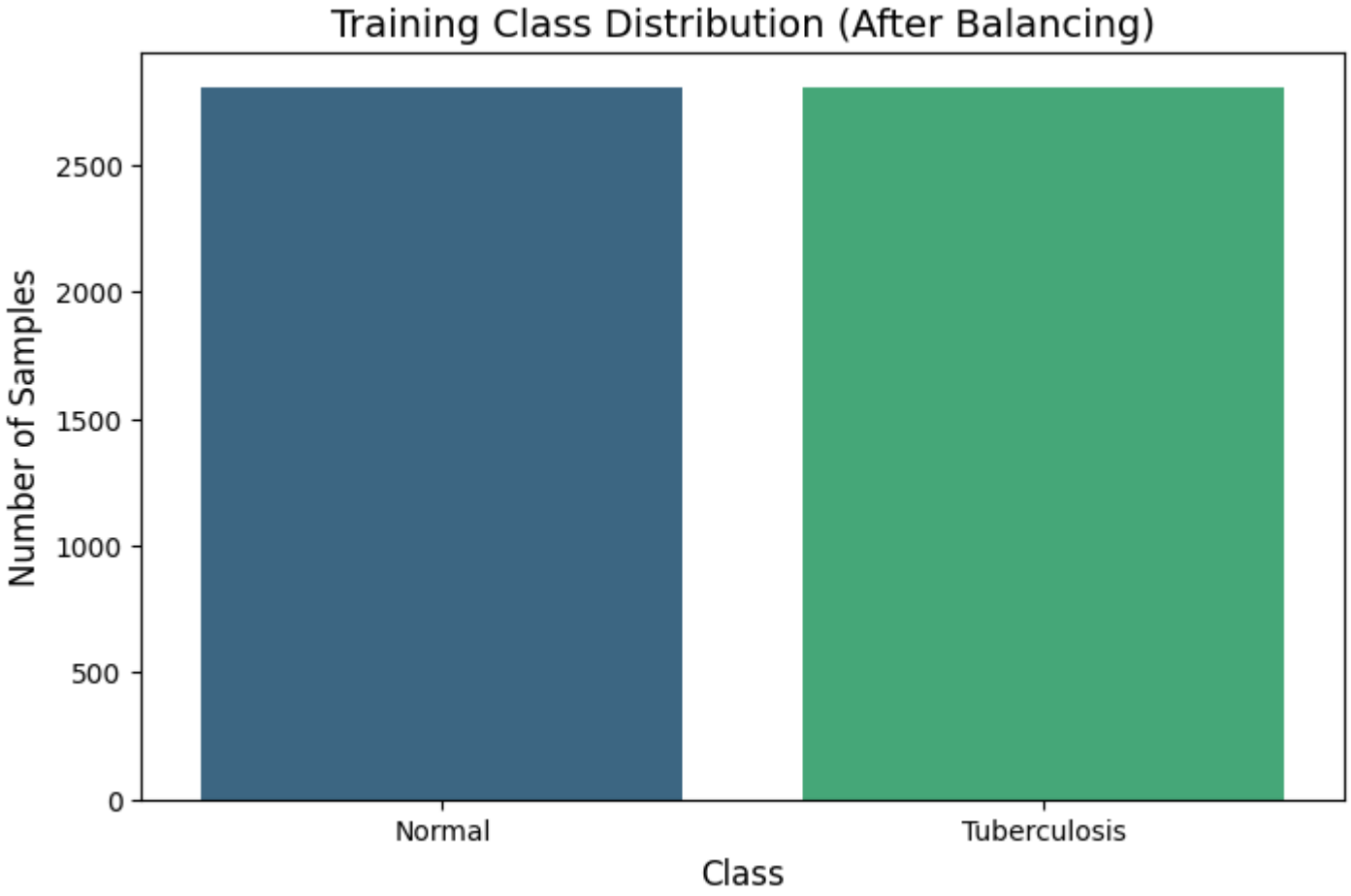



```
<ipython-input-3-7349cddcffdc>:82: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to

  sns.barplot(x=unique, y=counts, palette="viridis")
```



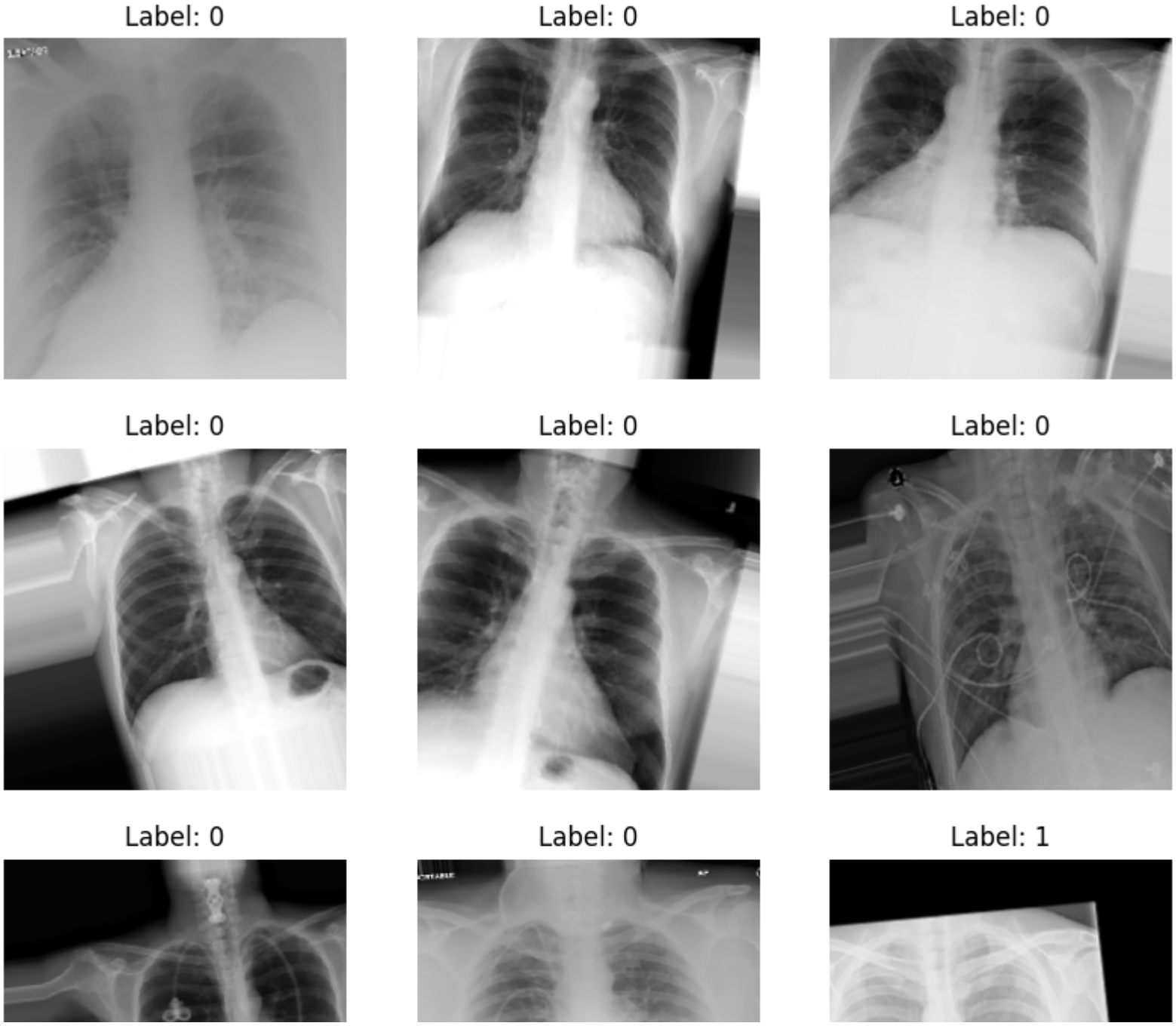
Class

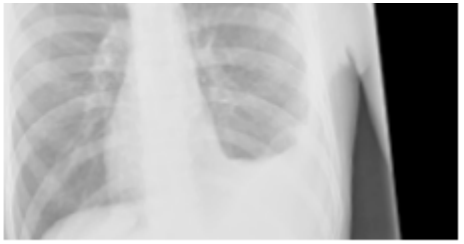
```
Class distribution AFTER handling imbalance with SMOTE:
<ipython-input-3-7349cddcffdc>:82: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to

  sns.barplot(x=unique, y=counts, palette="viridis")
```




```
Visualizing some synthetic samples after SMOTE...
```

Synthetic Samples After SMOTE


Label: 0
Label: 0
Label: 0
Label: 0
Label: 0
Label: 0
Label: 0
Label: 0
Label: 1

```
 1
 2 cnn_model = Sequential([
 3     Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
 4     MaxPooling2D((2, 2)),
 5     BatchNormalization(),
 6
 7     Conv2D(64, (3, 3), activation='relu'),
 8     MaxPooling2D((2, 2)),
 9     BatchNormalization(),
10
11     Flatten(),
12     Dense(64, kernel_initializer='uniform', activation='relu'),
13     Dropout(0.4),
14
15     Dense(1, activation='sigmoid')
16 ])
17
18 cnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
19 cnn_history = cnn_model.fit(X_train_res_images, y_train_res, epochs=20, class_weight=class_weights)
20
21 cnn_predictions = cnn_model.predict(val_data, steps=int(np.ceil(val_data.samples / val_data.batch_size)), verbose=2
22 cnn_predictions = (cnn_predictions > 0.6).astype(int)
23
24 print("CNN Model Accuracy:", accuracy_score(val_data.classes, cnn_predictions))
25 print("CNN Classification Report:\n", classification_report(val_data.classes, cnn_predictions))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass
```

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Y
  self._warn_if_super_not_called()
105/105 ━━━━━━━━━━━━━━━━━━━━ 418s 4s/step - accuracy: 0.7704 - loss: 20.6091 - val_accuracy: 0.1679 - val_loss: 4.
Epoch 2/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 117s 798ms/step - accuracy: 0.7886 - loss: 5.4675 - val_accuracy: 0.1810 - val_loss:
Epoch 3/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 143s 834ms/step - accuracy: 0.8652 - loss: 1.6285 - val_accuracy: 0.7798 - val_loss:
Epoch 4/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 99s 837ms/step - accuracy: 0.8651 - loss: 2.9271 - val_accuracy: 0.8833 - val_loss: 0
Epoch 5/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 134s 825ms/step - accuracy: 0.8959 - loss: 2.0373 - val_accuracy: 0.9357 - val_loss:
Epoch 6/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 144s 832ms/step - accuracy: 0.9046 - loss: 1.6156 - val_accuracy: 0.1667 - val_loss:
Epoch 7/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 142s 830ms/step - accuracy: 0.8759 - loss: 3.1349 - val_accuracy: 0.5988 - val_loss:
Epoch 8/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 95s 860ms/step - accuracy: 0.9064 - loss: 0.8253 - val_accuracy: 0.1690 - val_loss: 4
Epoch 9/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 134s 784ms/step - accuracy: 0.9140 - loss: 0.4592 - val_accuracy: 0.9357 - val_loss:
Epoch 10/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 89s 795ms/step - accuracy: 0.9293 - loss: 0.3737 - val_accuracy: 0.9345 - val_loss: 0
Epoch 11/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 89s 805ms/step - accuracy: 0.9297 - loss: 0.4088 - val_accuracy: 0.8738 - val_loss: 0
Epoch 12/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 91s 822ms/step - accuracy: 0.9157 - loss: 0.4597 - val_accuracy: 0.8905 - val_loss: 0
Epoch 13/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 87s 787ms/step - accuracy: 0.9268 - loss: 0.3636 - val_accuracy: 0.8690 - val_loss: 0
Epoch 14/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 146s 813ms/step - accuracy: 0.9309 - loss: 0.4449 - val_accuracy: 0.6988 - val_loss:
Epoch 15/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 145s 852ms/step - accuracy: 0.9153 - loss: 0.5416 - val_accuracy: 0.9357 - val_loss:
Epoch 16/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 141s 835ms/step - accuracy: 0.9203 - loss: 0.4118 - val_accuracy: 0.9333 - val_loss:
Epoch 17/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 142s 839ms/step - accuracy: 0.9165 - loss: 0.4041 - val_accuracy: 0.9464 - val_loss:
Epoch 18/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 92s 829ms/step - accuracy: 0.9307 - loss: 0.3270 - val_accuracy: 0.8714 - val_loss: 0
Epoch 19/20
```

```
105/105 ━━━━━━━━━━━━━━━━━━━━ 144s 836ms/step - accuracy: 0.9066 - loss: 0.6851 - val_accuracy: 0.8869 - val_loss:
Epoch 20/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 139s 825ms/step - accuracy: 0.9256 - loss: 0.3783 - val_accuracy: 0.9310 - val_loss:
27/27 - 10s - 361ms/step
CNN Model Accuracy: 0.9178571428571428
CNN Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       700
           1       0.99      0.51      0.68       140

    accuracy                           0.92       840
   macro avg       0.95      0.76      0.81       840
weighted avg       0.92      0.92      0.91       840
```

```
densenet_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
densenet_model.trainable = False

x = densenet_model.output
x = GlobalAveragePooling2D()(x)
dropout_layer = Dropout(0.5)(x)
output = Dense(1, activation='sigmoid')(dropout_layer)

final_densenet_model = Model(inputs=densenet_model.input, outputs=output)

final_densenet_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
densenet_history = final_densenet_model.fit(
    X_train_res_images, y_train_res, epochs=20, class_weight=class_weights
)

densenet_predictions = final_densenet_model.predict(
    val_data, steps=int(np.ceil(val_data.samples / val_data.batch_size)), verbose=2
)
densenet_predictions = (densenet_predictions > 0.6).astype(int)

print("DenseNet Model Accuracy:", accuracy_score(val_data.classes, densenet_predictions))
```

```
22 print("DenseNet Model Accuracy: ", accuracy_score(val_data.classes, densenet_predictions))
23 print("DenseNet Classification Report:\n", classification_report(val_data.classes, densenet_predictions))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 144s 1s/step - accuracy: 0.5987 - loss: 0.7670 - val_accuracy: 0.8750 - val_loss: 0.3
Epoch 2/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 108s 849ms/step - accuracy: 0.8151 - loss: 0.4141 - val_accuracy: 0.9024 - val_loss:
Epoch 3/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 96s 859ms/step - accuracy: 0.8574 - loss: 0.3510 - val_accuracy: 0.9167 - val_loss: 0
Epoch 4/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 92s 835ms/step - accuracy: 0.8662 - loss: 0.3233 - val_accuracy: 0.9143 - val_loss: 0
Epoch 5/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 90s 804ms/step - accuracy: 0.8836 - loss: 0.2975 - val_accuracy: 0.9250 - val_loss: 0
Epoch 6/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 153s 927ms/step - accuracy: 0.9039 - loss: 0.2478 - val_accuracy: 0.9333 - val_loss:
Epoch 7/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 91s 811ms/step - accuracy: 0.9048 - loss: 0.2372 - val_accuracy: 0.9250 - val_loss: 0
Epoch 8/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 145s 851ms/step - accuracy: 0.9092 - loss: 0.2515 - val_accuracy: 0.9214 - val_loss:
Epoch 9/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 93s 835ms/step - accuracy: 0.9144 - loss: 0.2271 - val_accuracy: 0.9274 - val_loss: 0
Epoch 10/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 144s 853ms/step - accuracy: 0.9158 - loss: 0.2150 - val_accuracy: 0.9286 - val_loss:
Epoch 11/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 91s 824ms/step - accuracy: 0.9262 - loss: 0.2152 - val_accuracy: 0.9286 - val_loss: 0
Epoch 12/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 92s 821ms/step - accuracy: 0.9140 - loss: 0.2348 - val_accuracy: 0.9286 - val_loss: 0
Epoch 13/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 92s 832ms/step - accuracy: 0.9200 - loss: 0.2230 - val_accuracy: 0.9345 - val_loss: 0
Epoch 14/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 94s 843ms/step - accuracy: 0.9109 - loss: 0.2295 - val_accuracy: 0.9321 - val_loss: 0
Epoch 15/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 139s 826ms/step - accuracy: 0.9086 - loss: 0.2302 - val_accuracy: 0.9512 - val_loss:
Epoch 16/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 90s 801ms/step - accuracy: 0.9009 - loss: 0.2504 - val_accuracy: 0.9440 - val_loss: 0
Epoch 17/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 144s 838ms/step - accuracy: 0.9269 - loss: 0.1995 - val_accuracy: 0.9548 - val_loss:
Epoch 18/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 91s 814ms/step - accuracy: 0.9152 - loss: 0.2127 - val_accuracy: 0.9607 - val_loss: 0
```

```
105/105 ━━━━━━━━━━━━━━━━━━━━ 91s 814ms/step - accuracy: 0.9132 - loss: 0.2127 - val_accuracy: 0.9607 - val_loss: 0
Epoch 19/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 141s 822ms/step - accuracy: 0.9211 - loss: 0.2065 - val_accuracy: 0.9607 - val_loss:
Epoch 20/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 143s 823ms/step - accuracy: 0.9106 - loss: 0.2247 - val_accuracy: 0.9595 - val_loss:
27/27 - 27s - 1s/step
DenseNet Model Accuracy: 0.9464285714285714
DenseNet Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       700
           1       0.94      0.72      0.82       140

    accuracy                           0.95       840
   macro avg       0.95      0.86      0.89       840
weighted avg       0.95      0.95      0.94       840
```

```
 1
 2 from tensorflow.keras.applications import ResNet50
 3 from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense
 4 from tensorflow.keras.models import Model
 5
 6 resnet_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
 7 resnet_model.trainable = False
 8
 9 x = resnet_model.output
10 x = GlobalAveragePooling2D()(x)
11 dropout_layer = Dropout(0.5)(x)
12 output = Dense(1, activation='sigmoid')(dropout_layer)
13
14 final_resnet_model = Model(inputs=resnet_model.input, outputs=output)
15
16 final_resnet_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
17 resnet_history = final_resnet_model.fit(
18     X_train_res_images, y_train_res, epochs=20, class_weight=class_weights
19 )
20
```

```
resnet_predictions = final_resnet_model.predict(
    val_data, steps=int(np.ceil(val_data.samples / val_data.batch_size)), verbose=2
)
resnet_predictions = (resnet_predictions > 0.6).astype(int)

print("ResNet50 Model Accuracy:", accuracy_score(val_data.classes, resnet_predictions))
print("ResNet50 Classification Report:\n", classification_report(val_data.classes, resnet_predictions))

```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 117s 921ms/step - accuracy: 0.5390 - loss: 0.7437 - val_accuracy: 0.8333 - val_loss:
Epoch 2/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 96s 859ms/step - accuracy: 0.5603 - loss: 0.7079 - val_accuracy: 0.8333 - val_loss: 0
Epoch 3/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 94s 848ms/step - accuracy: 0.5844 - loss: 0.6934 - val_accuracy: 0.2321 - val_loss: 0
Epoch 4/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 97s 876ms/step - accuracy: 0.5879 - loss: 0.6764 - val_accuracy: 0.3226 - val_loss: 0
Epoch 5/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 95s 845ms/step - accuracy: 0.5594 - loss: 0.6824 - val_accuracy: 0.2643 - val_loss: 0
Epoch 6/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 141s 844ms/step - accuracy: 0.4973 - loss: 0.6951 - val_accuracy: 0.1667 - val_loss:
Epoch 7/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 142s 844ms/step - accuracy: 0.4628 - loss: 0.6908 - val_accuracy: 0.3119 - val_loss:
Epoch 8/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 144s 851ms/step - accuracy: 0.5746 - loss: 0.6816 - val_accuracy: 0.3976 - val_loss:
Epoch 9/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 137s 818ms/step - accuracy: 0.5534 - loss: 0.6898 - val_accuracy: 0.1667 - val_loss:
Epoch 10/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 92s 835ms/step - accuracy: 0.4976 - loss: 0.6899 - val_accuracy: 0.2821 - val_loss: 0
Epoch 11/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 92s 830ms/step - accuracy: 0.4596 - loss: 0.6874 - val_accuracy: 0.8262 - val_loss: 0
Epoch 12/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 142s 821ms/step - accuracy: 0.5827 - loss: 0.6935 - val_accuracy: 0.1714 - val_loss:
Epoch 13/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 90s 810ms/step - accuracy: 0.5520 - loss: 0.6827 - val_accuracy: 0.4774 - val_loss: 0
Epoch 14/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 93s 843ms/step - accuracy: 0.5558 - loss: 0.6694 - val_accuracy: 0.5452 - val_loss: 0
Epoch 15/20
```

```
Epoch 15/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 92s 837ms/step - accuracy: 0.5240 - loss: 0.7033 - val_accuracy: 0.8083 - val_loss: 0
Epoch 16/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 96s 858ms/step - accuracy: 0.6696 - loss: 0.6685 - val_accuracy: 0.1667 - val_loss: 0
Epoch 17/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 94s 847ms/step - accuracy: 0.5150 - loss: 0.6738 - val_accuracy: 0.1667 - val_loss: 0
Epoch 18/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 94s 853ms/step - accuracy: 0.5376 - loss: 0.6777 - val_accuracy: 0.4988 - val_loss: 0
Epoch 19/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 95s 850ms/step - accuracy: 0.5959 - loss: 0.6760 - val_accuracy: 0.2679 - val_loss: 0
Epoch 20/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 141s 857ms/step - accuracy: 0.5792 - loss: 0.6871 - val_accuracy: 0.8190 - val_loss:
27/27 - 18s - 657ms/step
ResNet50 Model Accuracy: 0.8333333333333334
ResNet50 Classification Report:
              precision    recall  f1-score   support

           0       0.83      1.00      0.91       700
           1       0.00      0.00      0.00       140

    accuracy                           0.83       840
   macro avg       0.42      0.50      0.45       840
weighted avg       0.69      0.83      0.76       840

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precisior
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precisior
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
# EfficientNetB0 Model
from tensorflow.keras.applications import EfficientNetB0


# Load pre-trained EfficientNetB0
efficientnet_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
efficientnet_model.trainable = False

x = efficientnet_model.output
x = GlobalAveragePooling2D()(x)  # Pooling layer
```

```
dropout_layer = Dropout(0.5)(x)  # Dropout to reduce overfitting
output = Dense(1, activation='sigmoid')(dropout_layer)  # Output layer for binary classification

final_efficientnet_model = Model(inputs=efficientnet_model.input, outputs=output)

# Compile the model
final_efficientnet_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
efficientnet_history = final_efficientnet_model.fit(
    X_train_res_images, y_train_res,
    epochs=20,
    class_weight=class_weights
)

# Predictions
efficientnet_predictions = final_efficientnet_model.predict(
    val_data,
    steps=int(np.ceil(val_data.samples / val_data.batch_size)),
    verbose=2
)
efficientnet_predictions = (efficientnet_predictions > 0.6).astype(int)

# Evaluate the model
print("EfficientNetB0 Model Accuracy:", accuracy_score(val_data.classes, efficientnet_predictions))
print("EfficientNetB0 Classification Report:\n", classification_report(val_data.classes, efficientnet_predictions))

```

```
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Y
  self._warn_if_super_not_called()
105/105 ━━━━━━━━━━━━━━━━━━━━ 666s 6s/step - accuracy: 0.5069 - loss: 0.6939 - val_accuracy: 0.1667 - val_loss: 0.7
Epoch 2/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 510s 5s/step - accuracy: 0.3460 - loss: 0.7048 - val_accuracy: 0.8333 - val_loss: 0.6
Epoch 3/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 546s 5s/step - accuracy: 0.5256 - loss: 0.6934 - val_accuracy: 0.8333 - val_loss: 0.6
Epoch 4/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 498s 5s/step - accuracy: 0.4665 - loss: 0.7021 - val_accuracy: 0.8333 - val_loss: 0.6
```

```
Epoch 5/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 487s 5s/step - accuracy: 0.5855 - loss: 0.6991 - val_accuracy: 0.8333 - val_loss: 0.6
Epoch 6/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 506s 5s/step - accuracy: 0.6224 - loss: 0.6843 - val_accuracy: 0.1667 - val_loss: 0.7
Epoch 7/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 492s 5s/step - accuracy: 0.3264 - loss: 0.6865 - val_accuracy: 0.1667 - val_loss: 0.7
Epoch 8/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 484s 5s/step - accuracy: 0.4171 - loss: 0.6827 - val_accuracy: 0.1667 - val_loss: 0.7
Epoch 9/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 487s 5s/step - accuracy: 0.3381 - loss: 0.6998 - val_accuracy: 0.1690 - val_loss: 0.6
Epoch 10/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 482s 5s/step - accuracy: 0.5498 - loss: 0.6909 - val_accuracy: 0.8333 - val_loss: 0.6
Epoch 11/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 561s 5s/step - accuracy: 0.7367 - loss: 0.6685 - val_accuracy: 0.1667 - val_loss: 0.7
Epoch 12/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 487s 5s/step - accuracy: 0.3213 - loss: 0.6925 - val_accuracy: 0.1667 - val_loss: 0.7
Epoch 13/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 482s 5s/step - accuracy: 0.3741 - loss: 0.6902 - val_accuracy: 0.8333 - val_loss: 0.6
Epoch 14/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 475s 4s/step - accuracy: 0.7207 - loss: 0.6829 - val_accuracy: 0.1667 - val_loss: 0.7
Epoch 15/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 496s 4s/step - accuracy: 0.3323 - loss: 0.7021 - val_accuracy: 0.8333 - val_loss: 0.6
Epoch 16/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 469s 4s/step - accuracy: 0.5952 - loss: 0.6783 - val_accuracy: 0.1667 - val_loss: 0.7
Epoch 17/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 463s 4s/step - accuracy: 0.5938 - loss: 0.6816 - val_accuracy: 0.1667 - val_loss: 0.7
Epoch 18/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 509s 4s/step - accuracy: 0.4427 - loss: 0.6943 - val_accuracy: 0.8333 - val_loss: 0.6
Epoch 19/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 494s 4s/step - accuracy: 0.5890 - loss: 0.6959 - val_accuracy: 0.8333 - val_loss: 0.6
Epoch 20/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 509s 4s/step - accuracy: 0.5189 - loss: 0.6957 - val_accuracy: 0.1667 - val_loss: 0.6
27/27 - 89s - 3s/step
EfficientNetB0 Model Accuracy: 0.8333333333333334
EfficientNetB0 Classification Report:
              precision    recall  f1-score   support

           0       0.83      1.00      0.91       700
           1       0.00      0.00      0.00       140
```

```
          accuracy                           0.83       840
         macro avg       0.42      0.50      0.45       840
      weighted avg       0.69      0.83      0.76       840

      /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precisior
        _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
      /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precisior
        _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
# InceptionV3 Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense
from tensorflow.keras.models import Model
from sklearn.metrics import accuracy_score, classification_report
from tensorflow.keras.applications import InceptionV3

inception_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
inception_model.trainable = False

x = inception_model.output
x = GlobalAveragePooling2D()(x)
dropout_layer = Dropout(0.5)(x)
output = Dense(1, activation='sigmoid')(dropout_layer)

final_inception_model = Model(inputs=inception_model.input, outputs=output)

final_inception_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
inception_history = final_inception_model.fit(
    X_train_res_images, y_train_res, epochs=20, class_weight=class_weights
)

inception_predictions = final_inception_model.predict(
    val_data, steps=int(np.ceil(val_data.samples / val_data.batch_size)), verbose=2
)
inception_predictions = (inception_predictions > 0.6).astype(int)

print("InceptionV3 Model Accuracy:", accuracy_score(val_data.classes, inception_predictions))
print("InceptionV3 Classification Report:\n", classification_report(val_data.classes, inception_predictions))
```

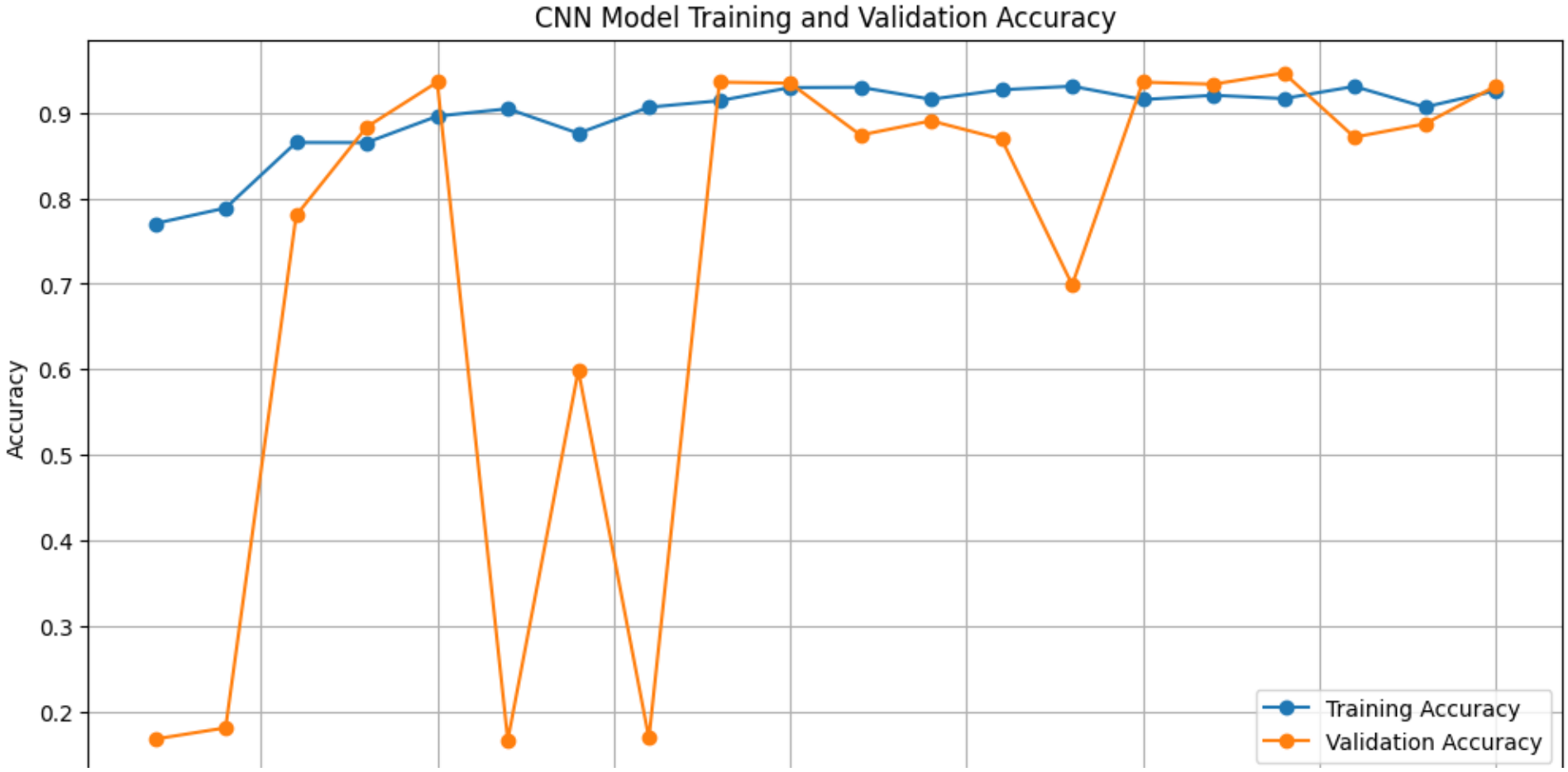
29

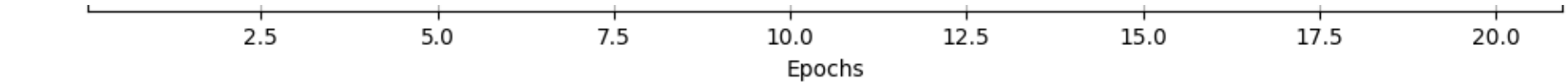
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weigh
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 681s 6s/step - accuracy: 0.7082 - loss: 0.5768 - val_accuracy: 0.9060 - val_loss: 0.2
Epoch 2/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 662s 6s/step - accuracy: 0.8837 - loss: 0.3111 - val_accuracy: 0.9155 - val_loss: 0.2
Epoch 3/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 705s 6s/step - accuracy: 0.9083 - loss: 0.2428 - val_accuracy: 0.9381 - val_loss: 0.2
Epoch 4/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 667s 6s/step - accuracy: 0.9091 - loss: 0.2317 - val_accuracy: 0.9250 - val_loss: 0.1
Epoch 5/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 689s 6s/step - accuracy: 0.9201 - loss: 0.2431 - val_accuracy: 0.9190 - val_loss: 0.1
Epoch 6/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 688s 6s/step - accuracy: 0.9207 - loss: 0.2189 - val_accuracy: 0.9464 - val_loss: 0.1
Epoch 7/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 745s 7s/step - accuracy: 0.9281 - loss: 0.2102 - val_accuracy: 0.9274 - val_loss: 0.2
Epoch 8/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 668s 6s/step - accuracy: 0.9189 - loss: 0.2123 - val_accuracy: 0.9488 - val_loss: 0.1
Epoch 9/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 694s 6s/step - accuracy: 0.9314 - loss: 0.1949 - val_accuracy: 0.9381 - val_loss: 0.1
Epoch 10/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 673s 6s/step - accuracy: 0.9443 - loss: 0.1692 - val_accuracy: 0.9452 - val_loss: 0.1
Epoch 11/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 673s 6s/step - accuracy: 0.9280 - loss: 0.1842 - val_accuracy: 0.9381 - val_loss: 0.1
Epoch 12/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 662s 6s/step - accuracy: 0.9227 - loss: 0.2294 - val_accuracy: 0.9405 - val_loss: 0.1
Epoch 13/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 662s 6s/step - accuracy: 0.9301 - loss: 0.1801 - val_accuracy: 0.9369 - val_loss: 0.1
Epoch 14/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 682s 6s/step - accuracy: 0.9446 - loss: 0.1587 - val_accuracy: 0.9429 - val_loss: 0.1
Epoch 15/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 665s 6s/step - accuracy: 0.9344 - loss: 0.1738 - val_accuracy: 0.9452 - val_loss: 0.1
Epoch 16/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 697s 6s/step - accuracy: 0.9355 - loss: 0.1747 - val_accuracy: 0.9440 - val_loss: 0.1
Epoch 17/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 687s 6s/step - accuracy: 0.9364 - loss: 0.1885 - val_accuracy: 0.9440 - val_loss: 0.1
Epoch 18/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 719s 6s/step - accuracy: 0.9165 - loss: 0.2286 - val_accuracy: 0.9393 - val_loss: 0.1
Epoch 19/20
```

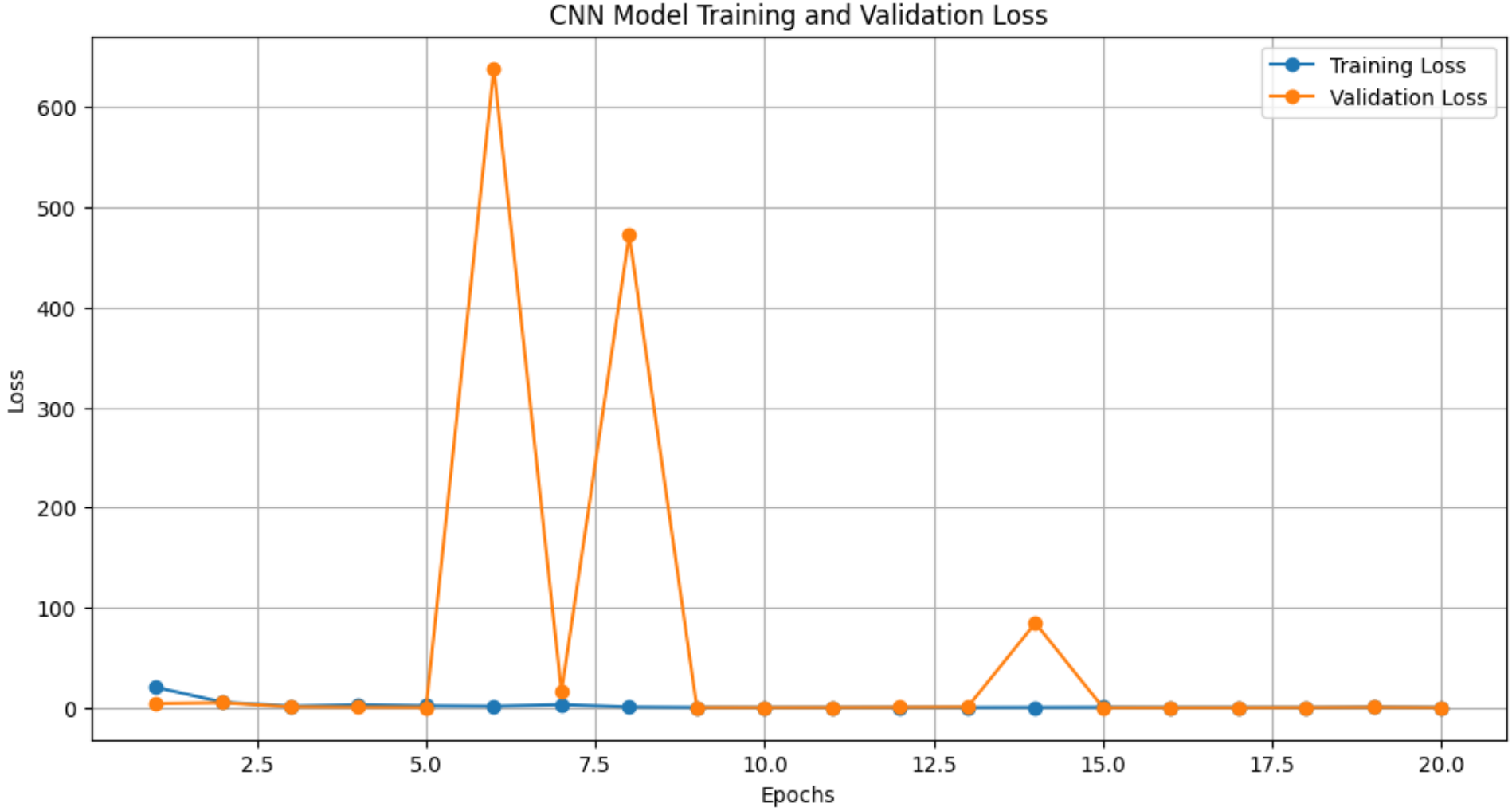
```
Epoch 19/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 702s 6s/step - accuracy: 0.9392 - loss: 0.1547 - val_accuracy: 0.9417 - val_loss: 0.1
Epoch 20/20
105/105 ━━━━━━━━━━━━━━━━━━━━ 681s 6s/step - accuracy: 0.9362 - loss: 0.1700 - val_accuracy: 0.9417 - val_loss: 0.1
27/27 - 129s - 5s/step
InceptionV3 Model Accuracy: 0.9464285714285714
InceptionV3 Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.99      0.97       700
           1       0.96      0.71      0.81       140

    accuracy                           0.95       840
   macro avg       0.95      0.85      0.89       840
weighted avg       0.95      0.95      0.94       840
```

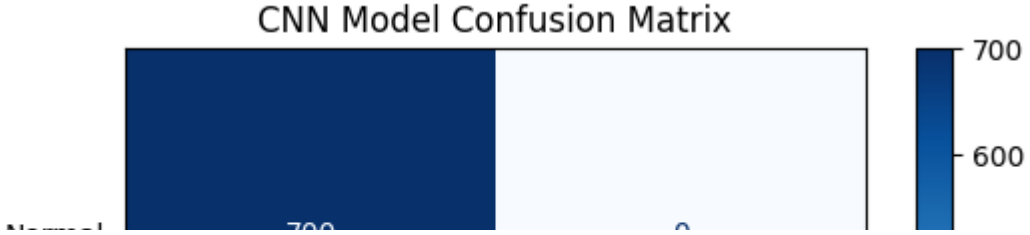
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

plt.figure(figsize=(12, 6))
plt.plot(cnn_history.history['accuracy'], label="Training Accuracy", marker='o')
plt.plot(cnn_history.history['val_accuracy'], label="Validation Accuracy", marker='o')
plt.title("CNN Model Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()


plt.figure(figsize=(12, 6))
plt.plot(cnn_history.history['loss'], label="Training Loss", marker='o')
plt.plot(cnn_history.history['val_loss'], label="Validation Loss", marker='o')
plt.title("CNN Model Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
```

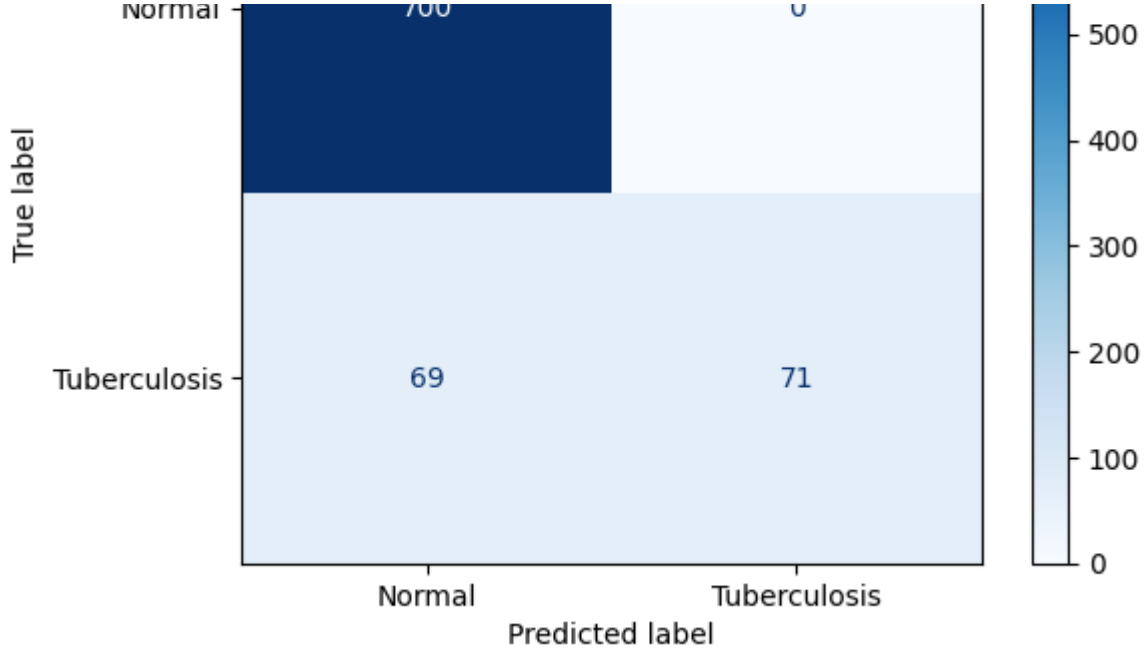
```
plt.legend()
plt.grid(True)
plt.show()

cm = confusion_matrix(val_data.classes, cnn_predictions)
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Normal', 'Tuberculosis'])
disp.plot(cmap='Blues', values_format='d')
plt.title("CNN Model Confusion Matrix")
plt.show()

```




2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs

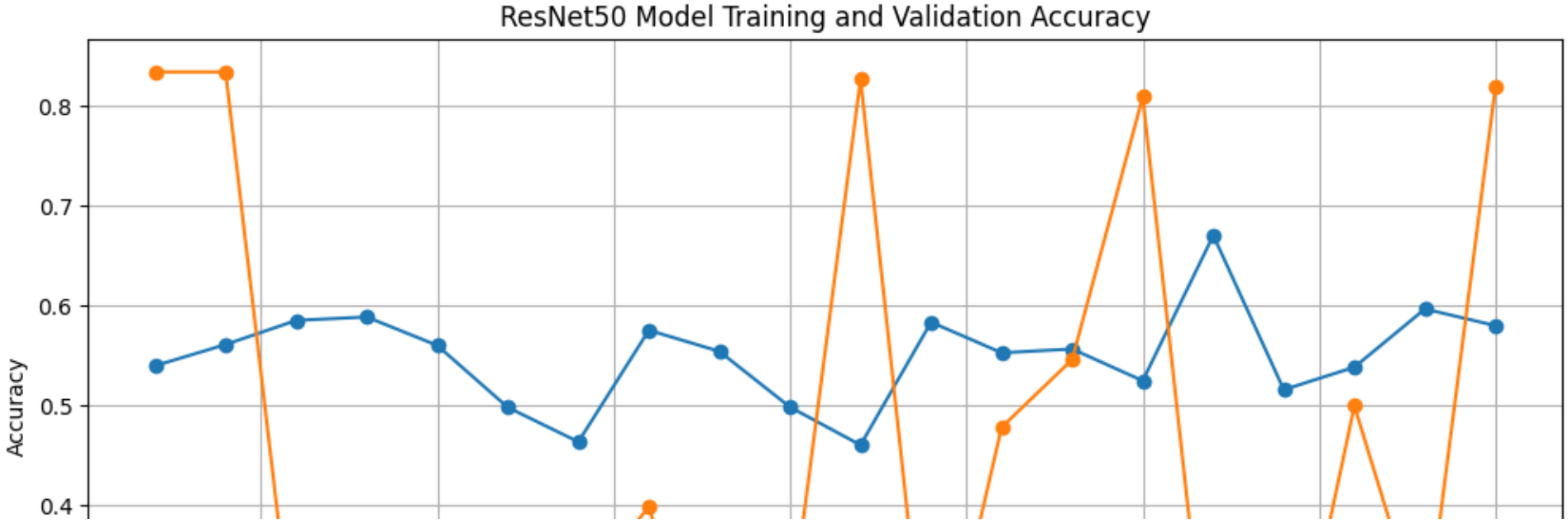

CNN Model Training and Validation Loss
Training Loss
Validation Loss
600
500
400
300
200
100
0
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs

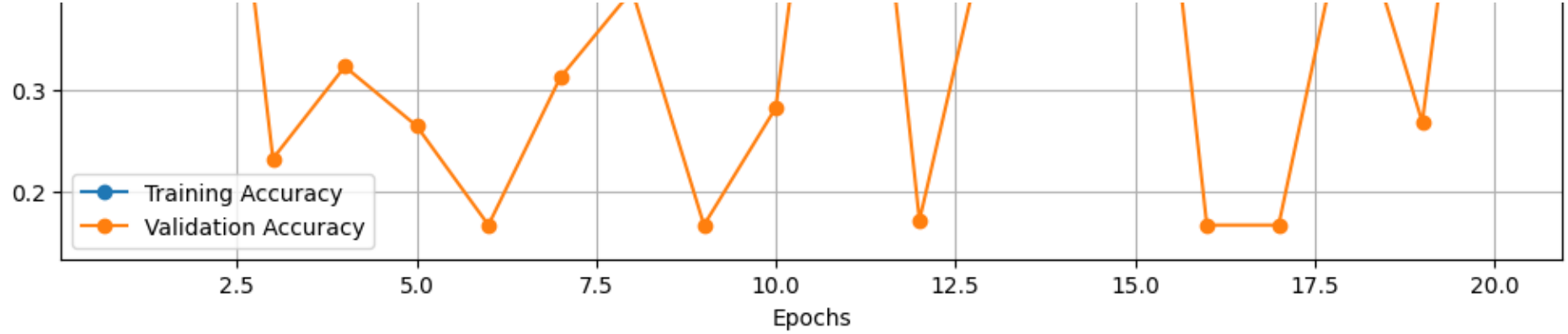

CNN Model Confusion Matrix
700
600

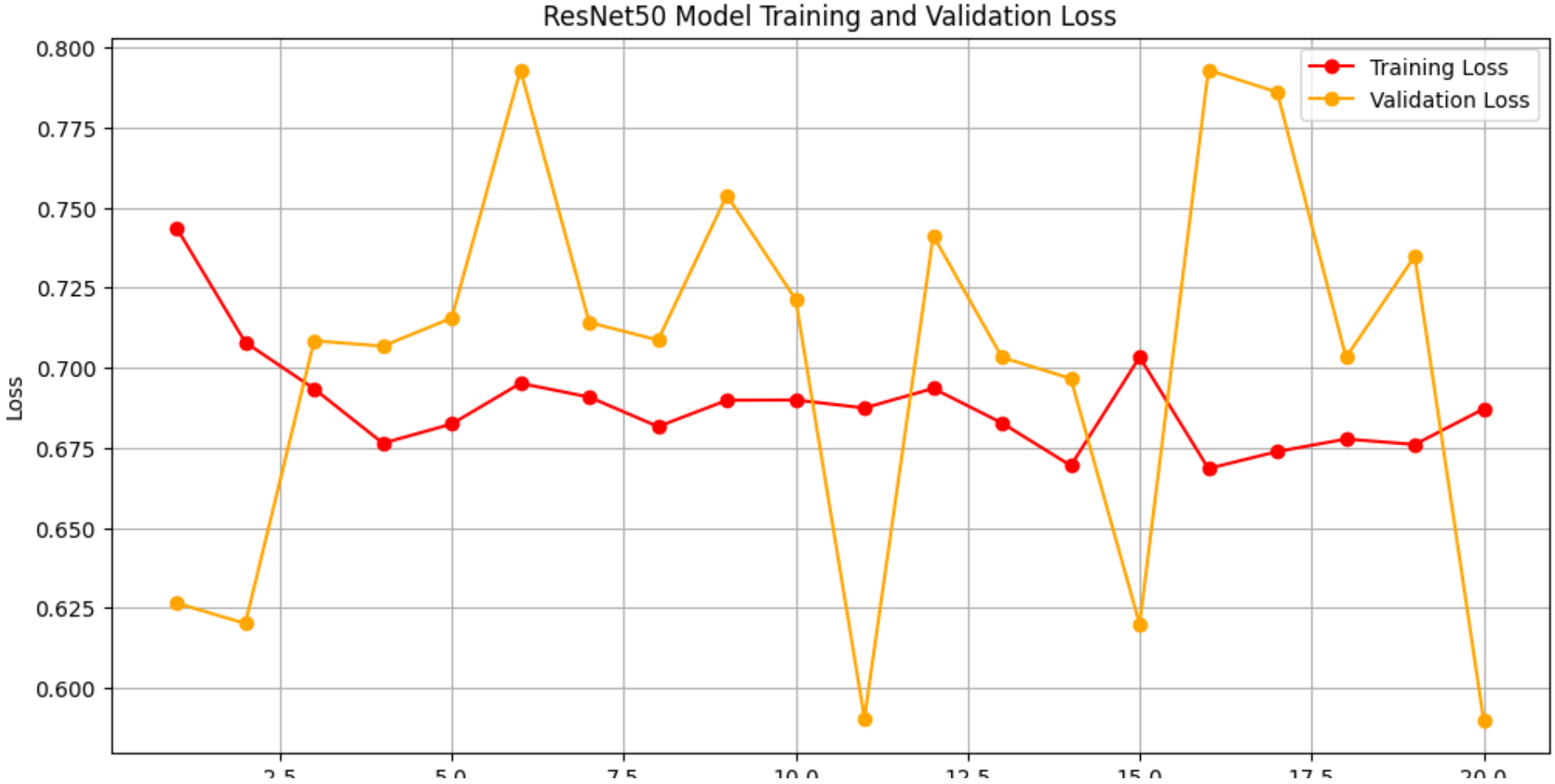



```
 1 import matplotlib.pyplot as plt
 2 import numpy as np
 3 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
 4
 5
 6 plt.figure(figsize=(12, 6))
 7 plt.plot(resnet_history.history['accuracy'], label='Training Accuracy', marker='o')
 8 plt.plot(resnet_history.history['val_accuracy'], label='Validation Accuracy', marker='o')
 9 plt.title('ResNet50 Model Training and Validation Accuracy')
10 plt.xlabel('Epochs')
11 plt.ylabel('Accuracy')
12 plt.legend()
13 plt.grid(True)
14 plt.show()
15
```

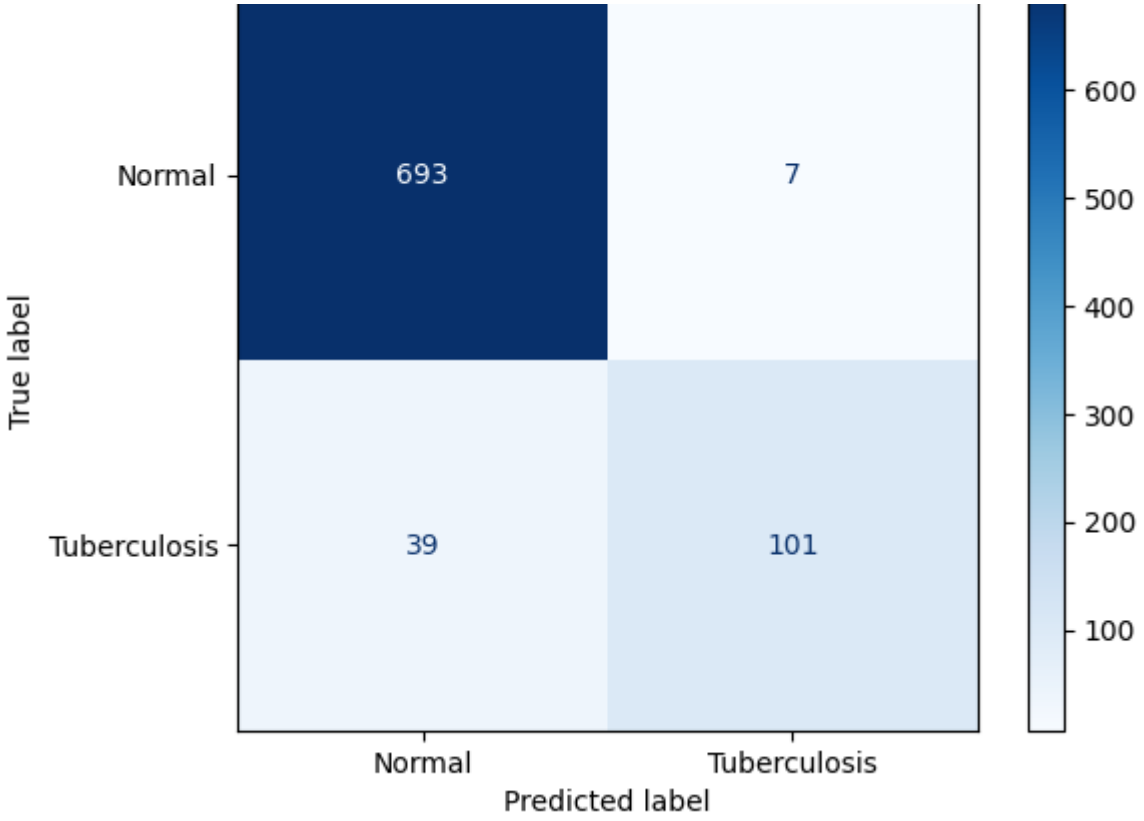
```
16
17 plt.figure(figsize=(12, 6))
18 plt.plot(resnet_history.history['loss'], label='Training Loss', marker='o', color='red')
19 plt.plot(resnet_history.history['val_loss'], label='Validation Loss', marker='o', color='orange')
20 plt.title('ResNet50 Model Training and Validation Loss')
21 plt.xlabel('Epochs')
22 plt.ylabel('Loss')
23 plt.legend()
24 plt.grid(True)
25 plt.show()
26
27
28 cm = confusion_matrix(val_data.classes, resnet_predictions)
29 disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Normal', 'Tuberculosis'])
30 disp.plot(cmap='Blues')
31 plt.title('ResNet50 Model Confusion Matrix')
32 plt.show()
33
34
```

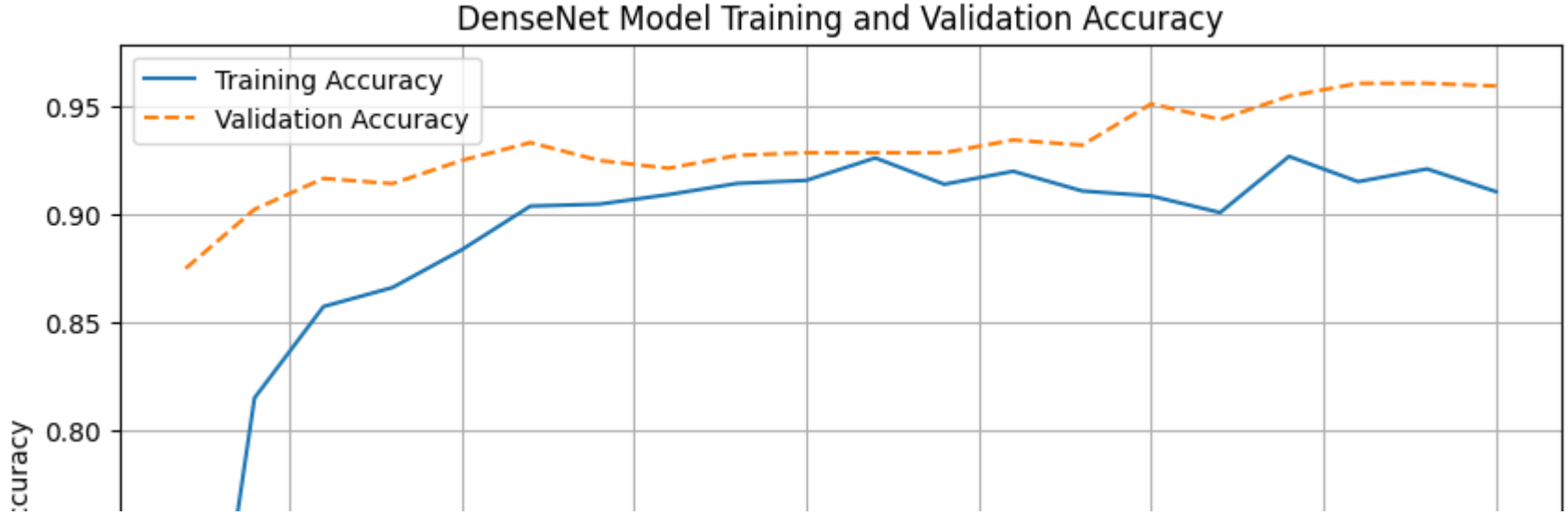


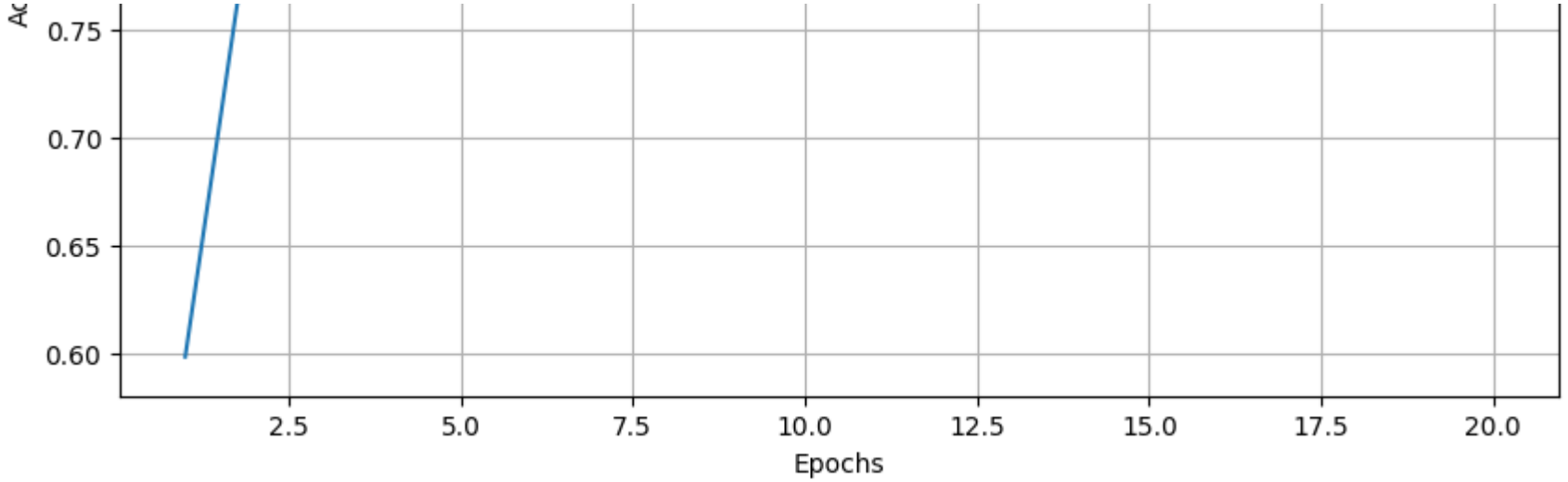

0.3
0.2
Training Accuracy
Validation Accuracy
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs


ResNet50 Model Training and Validation Loss
0.800
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
Loss
Training Loss
Validation Loss

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(val_data.classes, cnn_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Normal", "Tuberculosis"])


disp.plot(cmap=plt.cm.Blues)
plt.title("DenseNet Model Confusion Matrix")
plt.show()


plt.figure(figsize=(10, 6))
plt.plot(densenet_history.history['accuracy'], label="Training Accuracy")
plt.plot(densenet_history.history['val_accuracy'], label="Validation Accuracy", linestyle="--")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("DenseNet Model Training and Validation Accuracy")
plt.legend()
plt.grid()
plt.show()


plt.figure(figsize=(10, 6))
plt.plot(densenet_history.history['loss'], label="Training Loss")
plt.plot(densenet_history.history['val_loss'], label="Validation Loss", linestyle="--")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("DenseNet Model Training and Validation Loss")
plt.legend()
plt.grid()
plt.show()

```

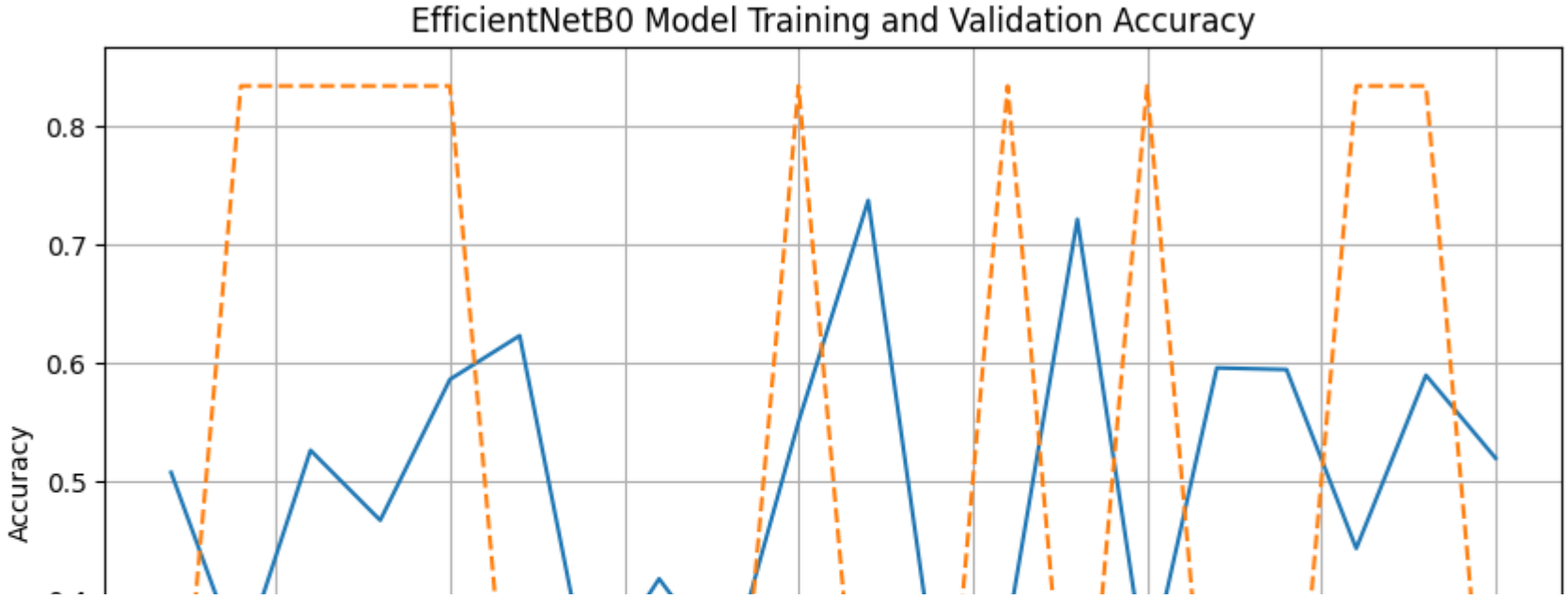

Normal
Tuberculosis
693
7
39
101
True label
Normal
Tuberculosis
Predicted label
600
500
400
300
200
100

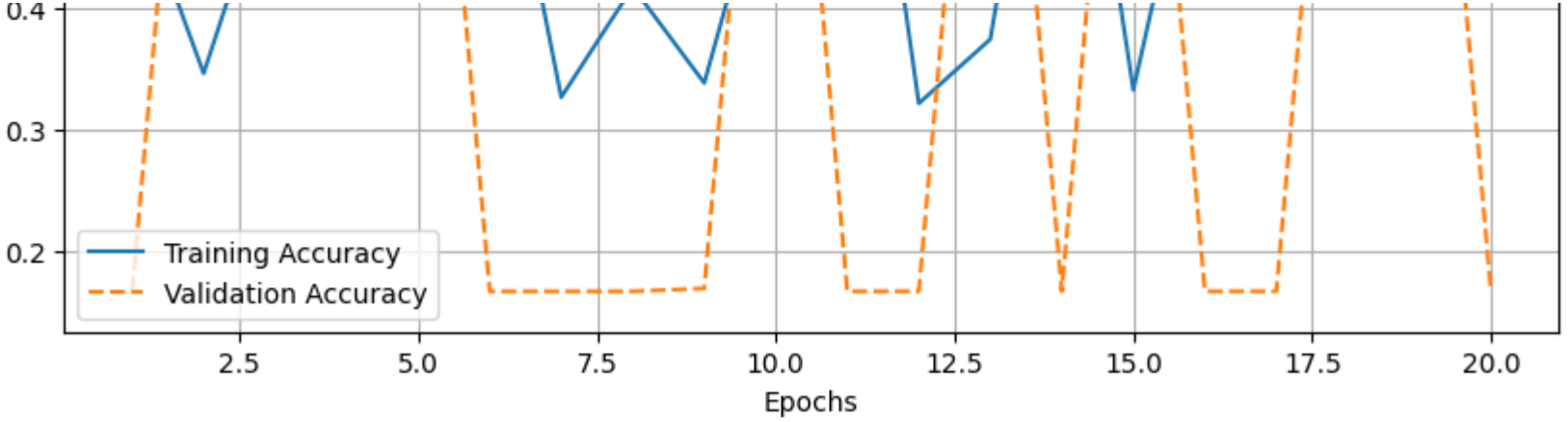

DenseNet Model Training and Validation Accuracy
Training Accuracy
Validation Accuracy
0.95
0.90
0.85
0.80

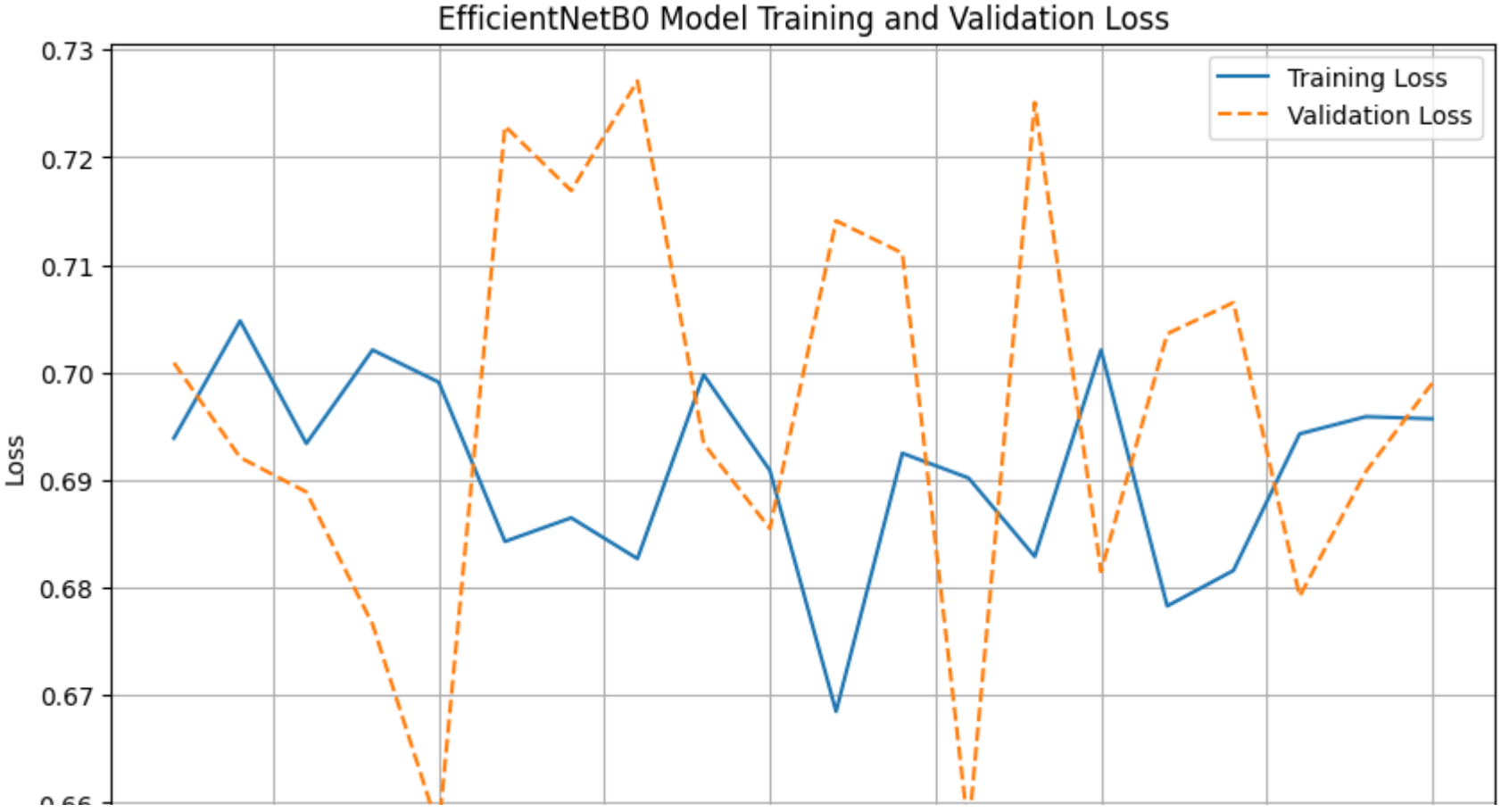



```
import matplotlib.pyplot as plt



plt.figure(figsize=(10, 6))
plt.plot(efficientnet_history.history['accuracy'], label="Training Accuracy")
plt.plot(efficientnet_history.history['val_accuracy'], '--', label="Validation Accuracy")
plt.title("EfficientNetB0 Model Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()



plt.figure(figsize=(10, 6))
plt.plot(efficientnet_history.history['loss'], label="Training Loss")
plt.plot(efficientnet_history.history['val_loss'], '--', label="Validation Loss")
plt.title("EfficientNetB0 Model Training and Validation Loss")
plt.xlabel("Epochs")
```

```
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


cm = confusion_matrix(val_data.classes, efficientnet_predictions)


plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Normal", "Tuberculosis"], yticklabels=["Normal", '
plt.title("Matrix for EfficientNetB0")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

```

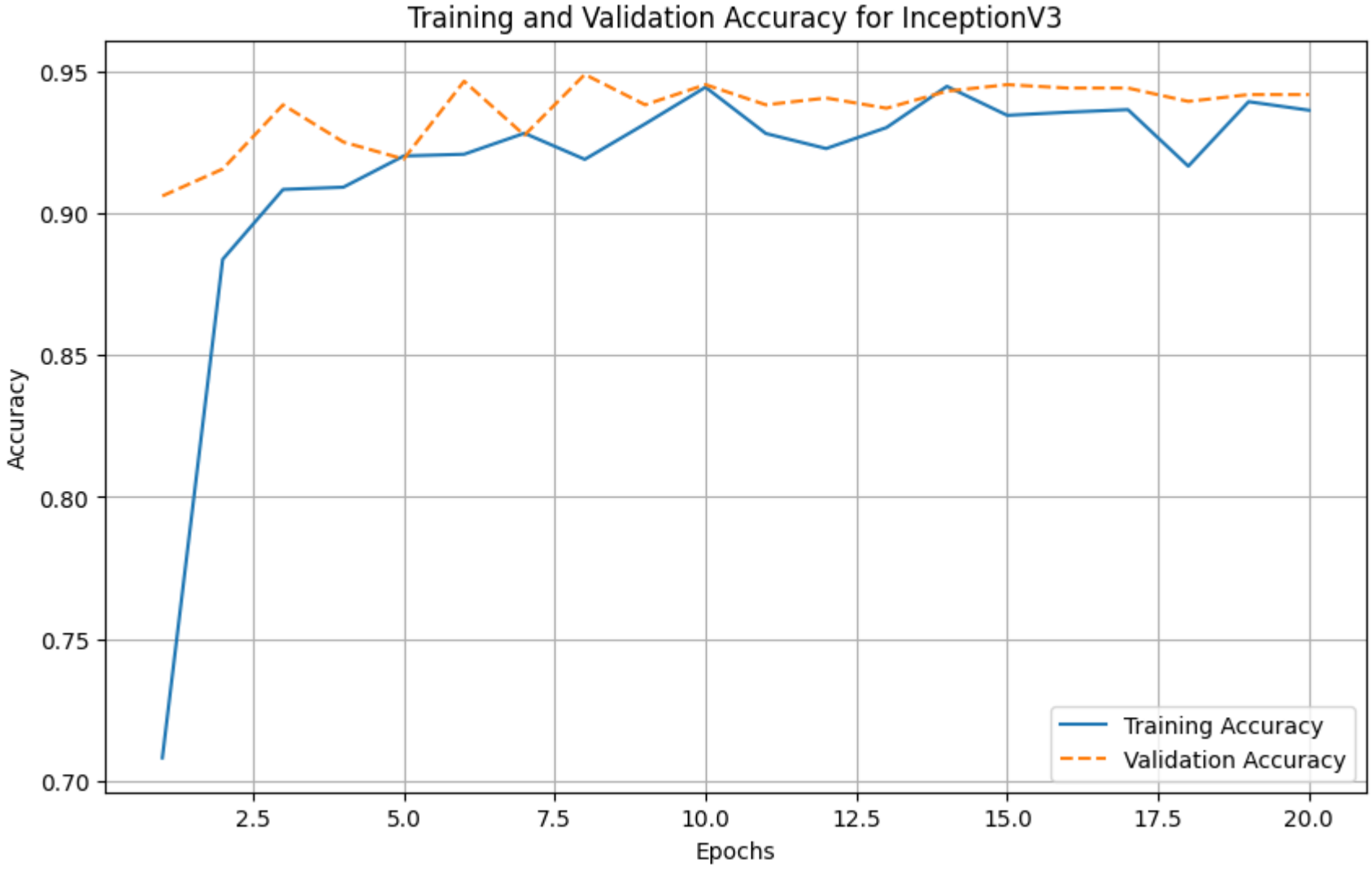


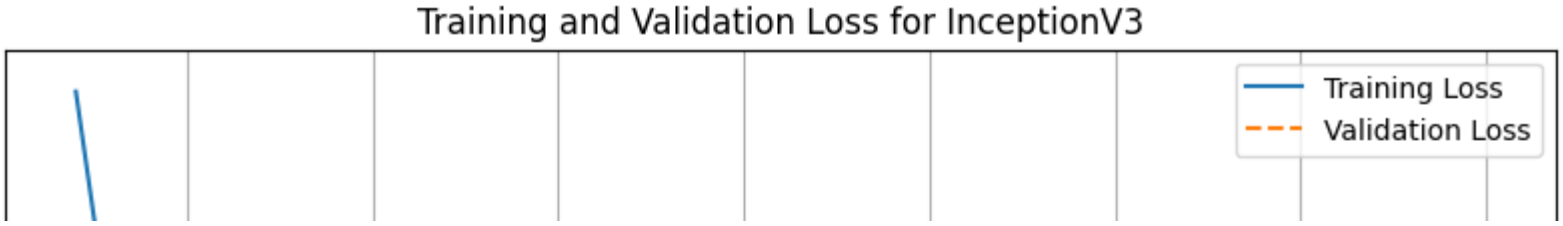

0.4
0.3
0.2
Training Accuracy
Validation Accuracy
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs

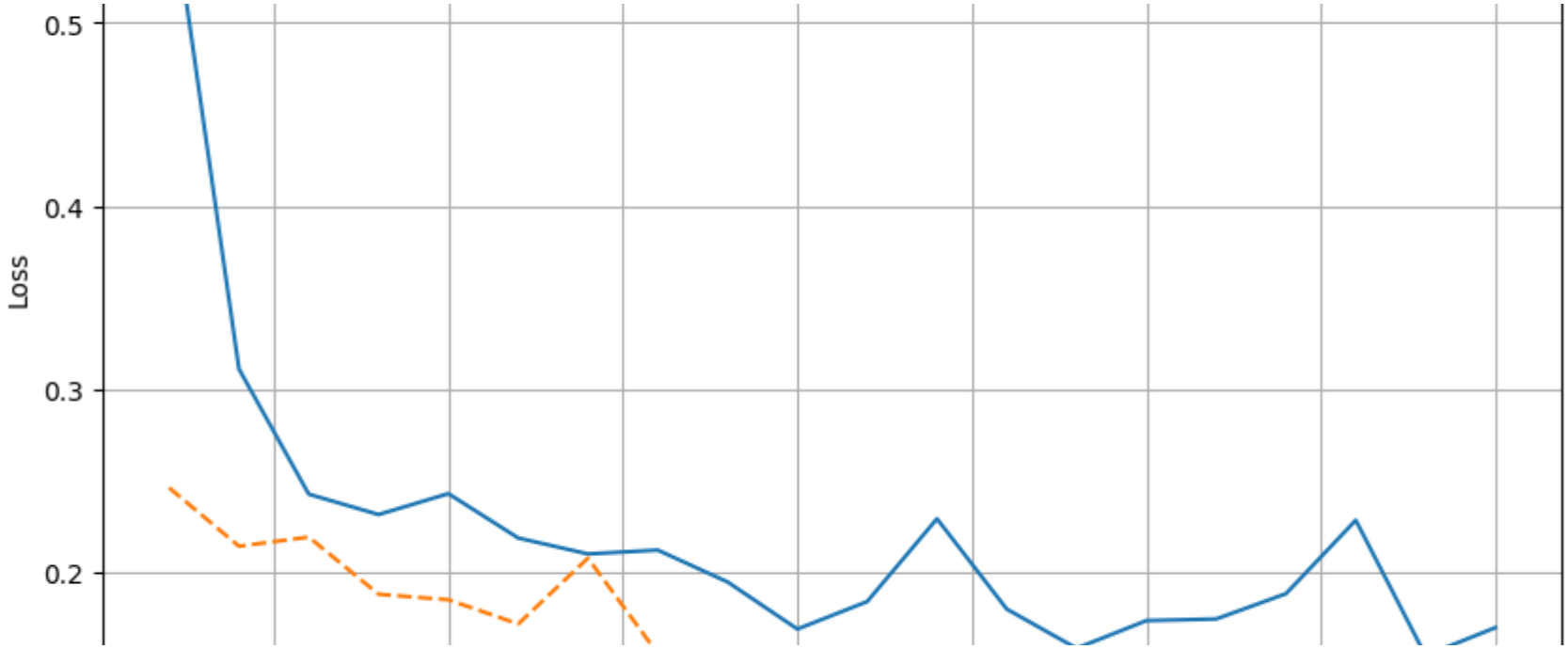

EfficientNetB0 Model Training and Validation Loss
0.73
0.72
0.71
0.70
0.69
0.68
0.67
Loss
Training Loss
Validation Loss

```
import numpy as np


plt.figure(figsize=(10, 6))
plt.plot(inception_history.history['accuracy'], label="Training Accuracy")
plt.plot(inception_history.history['val_accuracy'], label="Validation Accuracy", linestyle='--')
plt.title("Training and Validation Accuracy for InceptionV3")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()


plt.figure(figsize=(10, 6))
plt.plot(inception_history.history['loss'], label="Training Loss")
plt.plot(inception_history.history['val_loss'], label="Validation Loss", linestyle='--')
plt.title("Training and Validation Loss for InceptionV3")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(val_data.classes, inception_predictions)


plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt=".0f", cmap="Blues",
            xticklabels=["Normal", "Tuberculosis"],
            yticklabels=["Normal", "Tuberculosis"])
plt.title("Confusion Matrix for InceptionV3")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

38

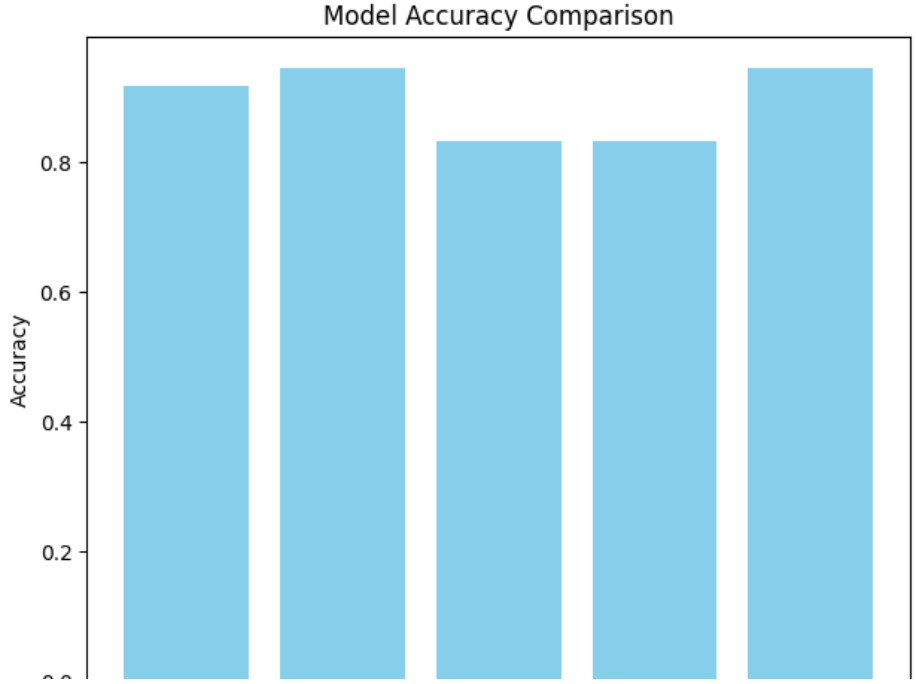

Training and Validation Accuracy for InceptionV3
Accuracy
Epochs
0.95
0.90
0.85
0.80
0.75
0.70
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training Accuracy
Validation Accuracy

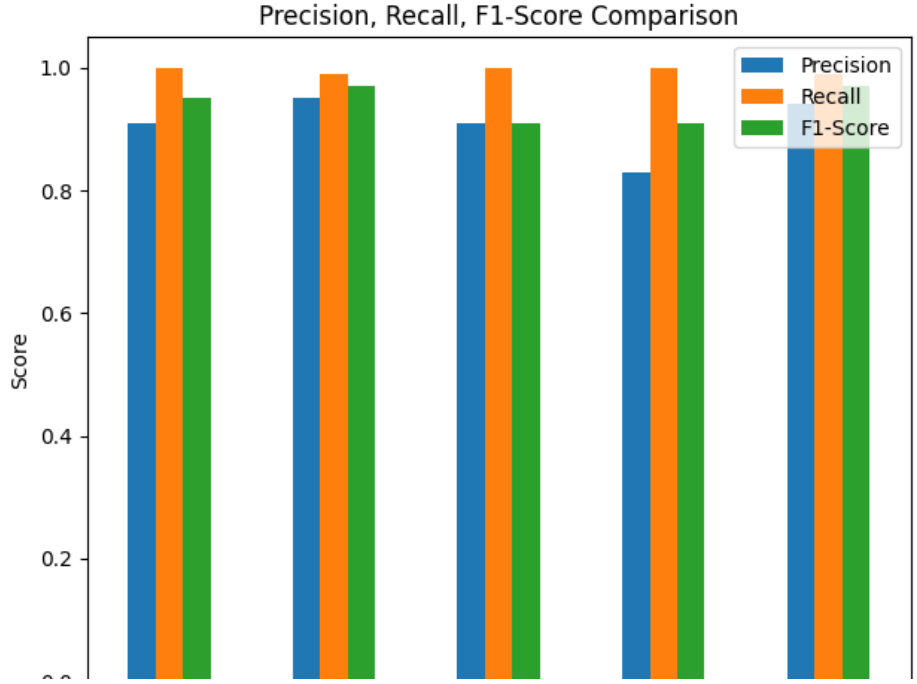

Training and Validation Loss for InceptionV3
Training Loss
Validation Loss

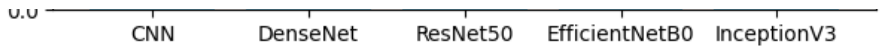


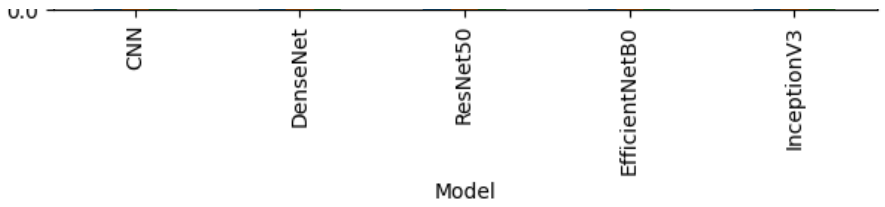

```
 1 import matplotlib.pyplot as plt
 2 import seaborn as sns
 3 import numpy as np
 4 import pandas as pd
 5
 6
 7 models = ['CNN', 'DenseNet', 'ResNet50', 'EfficientNetB0', 'InceptionV3']
 8 accuracies = [0.917857, 0.946429, 0.833333, 0.833333, 0.946429]
 9 precision = [0.91, 0.95, 0.91, 0.83, 0.94]
10 recall = [1.00, 0.99, 1.00, 1.00, 0.99]
11 f1_score = [0.95, 0.97, 0.91, 0.91, 0.97]
12
13
14 fig, axes = plt.subplots(1, 2, figsize=(14, 6))
15 fig.tight_layout(pad=5.0)
16
17
18 axes[0].bar(models, accuracies, color='skyblue')
```

```
axes[0].set_title('Model Accuracy Comparison')
axes[0].set_ylabel('Accuracy')


metrics_df = pd.DataFrame({
    'Model': models,
    'Precision': precision,
    'Recall': recall,
    'F1-Score': f1_score
})

metrics_df.set_index('Model').plot(kind='bar', ax=axes[1], color=['#1f77b4', '#ff7f0e', '#2ca02c'])
axes[1].set_title('Precision, Recall, F1-Score Comparison')
axes[1].set_ylabel('Score')


plt.show()

```






0.0
CNN
DenseNet
ResNet50
EfficientNetB0
InceptionV3


0.0
CNN
DenseNet
ResNet50
EfficientNetB0
InceptionV3
Model