```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img,
img_to_array
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input,
Dropout, Conv2D, BatchNormalization, Activation, Add, MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,
ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import seaborn as sns
import pandas as pd
from PIL import Image
import warnings

warnings.filterwarnings('ignore', category=UserWarning, module='keras')

# Suppress CUDA warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

# Print TensorFlow version and GPU availability
print("TensorFlow Version:", tf.__version__)
print("Num GPUs Available:", len(tf.config.list_physical_devices('GPU')))

# Define paths to dataset directories
train_dir = '/kaggle/input/chest-xray-pneumonia/chest_xray/train'
test_dir = '/kaggle/input/chest-xray-pneumonia/chest_xray/test'

# Verify dataset directories
if not os.path.exists(train_dir):
    raise FileNotFoundError(f"Training directory not found: {train_dir}")
if not os.path.exists(test_dir):
    raise FileNotFoundError(f"Test directory not found: {test_dir}")

# Custom ImageDataGenerator
def create_datagen():
    return ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
```

```
        height_shift_range=0.2,
        zoom_range=0.2,
        shear_range=0.2,
        brightness_range=[0.8, 1.2],
        horizontal_flip=True,
        fill_mode='nearest'
    )

# Oversample Normal class
def oversample_normal_images(train_dir):
    normal_dir = os.path.join(train_dir, 'NORMAL')
    pneumonia_dir = os.path.join(train_dir, 'PNEUMONIA')
    normal_files = [os.path.join(normal_dir, f) for f in os.listdir(normal_dir)
if f.endswith('.jpeg')]
    pneumonia_files = [os.path.join(pneumonia_dir, f) for f in
os.listdir(pneumonia_dir) if f.endswith('.jpeg')]

    # Oversample Normal to match Pneumonia count (~3875)
    normal_oversampled = normal_files * (len(pneumonia_files) //
len(normal_files)) + normal_files[:len(pneumonia_files) % len(normal_files)]
    files = normal_oversampled + pneumonia_files
    labels = [0] * len(normal_oversampled) + [1] * len(pneumonia_files)
    return files, labels

# Load and balance training data
train_files, train_labels = oversample_normal_images(train_dir)
train_df = pd.DataFrame({'filename': train_files, 'class': train_labels})
train_df['class'] = train_df['class'].map({0: 'NORMAL', 1: 'PNEUMONIA'})

# Split into training and validation sets (70% train, 30% validation)
train_df, val_df = train_test_split(train_df, test_size=0.3,
stratify=train_df['class'], random_state=42)

# Print DataFrame info
print("Training DataFrame head:\n", train_df.head())
print("Validation DataFrame head:\n", val_df.head())
print("Training class counts:\n", train_df['class'].value_counts())
print("Validation class counts:\n", val_df['class'].value_counts())

# Data generators
train_datagen = create_datagen()
val_datagen = create_datagen()
test_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 16
```

```
training_data = train_datagen.flow_from_dataframe(
    train_df,
    x_col='filename',
    y_col='class',
    target_size=(224, 224),
    color_mode='grayscale',
    class_mode='binary',
    batch_size=batch_size,
    shuffle=True,
    seed=42
)

validation_data = val_datagen.flow_from_dataframe(
    val_df,
    x_col='filename',
    y_col='class',
    target_size=(224, 224),
    color_mode='grayscale',
    class_mode='binary',
    batch_size=batch_size,
    shuffle=True,  # Ensure shuffling for balanced batches
    seed=42
)

testing_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    color_mode='grayscale',
    class_mode='binary',
    batch_size=batch_size,
    shuffle=False
)

# Print class distributions
print("Training class distribution:", np.bincount(training_data.classes))
print("Validation class distribution:", np.bincount(validation_data.classes))
print("Class indices:", training_data.class_indices)

# Debug validation batch
images, labels = next(validation_data)
print("Validation batch images shape:", images.shape)  # Should be (16, 224, 224,
1)
print("Validation batch labels shape:", labels.shape)  # Should be (16,)
print("Validation batch labels:", labels)  # Should contain 0s and 1s
```

```
# Visualize sample training and validation images
def visualize_samples(df, title, num_samples=5):
    normal_files = df[df['class'] == 'NORMAL']['filename'].sample(num_samples,
random_state=42).tolist()
    pneumonia_files = df[df['class'] ==
'PNEUMONIA']['filename'].sample(num_samples, random_state=42).tolist()

    plt.figure(figsize=(15, 6))
    for i, file in enumerate(normal_files + pneumonia_files):
        img = load_img(file, target_size=(224, 224), color_mode='grayscale')
        img_array = img_to_array(img) / 255.0
        plt.subplot(2, num_samples, i + 1)
        plt.imshow(img_array.squeeze(), cmap='gray')
        plt.title('NORMAL' if i < num_samples else 'PNEUMONIA')
        plt.axis('off')
    plt.suptitle(title)
    plt.tight_layout()
    plt.savefig(f'{title.lower().replace(" ", "_")}.png')
    plt.show()

visualize_samples(train_df, "Training Samples")
visualize_samples(val_df, "Validation Samples")

# Custom CNN with skip connections
inputs = Input(shape=(224, 224, 1))

# Block 1
x = Conv2D(32, (3, 3), padding='same')(inputs)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Conv2D(32, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Block 2 with skip connection
identity = x
x = Conv2D(64, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Conv2D(64, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
identity = Conv2D(64, (1, 1), padding='same')(identity)
x = Add()([x, identity])
```

```
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Block 3 with skip connection
identity = x
x = Conv2D(128, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Conv2D(128, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
identity = Conv2D(128, (1, 1), padding='same')(identity)
x = Add()([x, identity])
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Block 4 with skip connection
identity = x
x = Conv2D(256, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Conv2D(256, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
identity = Conv2D(256, (1, 1), padding='same')(identity)
x = Add()([x, identity])
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Block 5 with skip connection
identity = x
x = Conv2D(512, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Conv2D(512, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
identity = Conv2D(512, (1, 1), padding='same')(identity)
x = Add()([x, identity])
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Block 6 with skip connection
identity = x
x = Conv2D(1024, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Conv2D(1024, (3, 3), padding='same')(x)
```

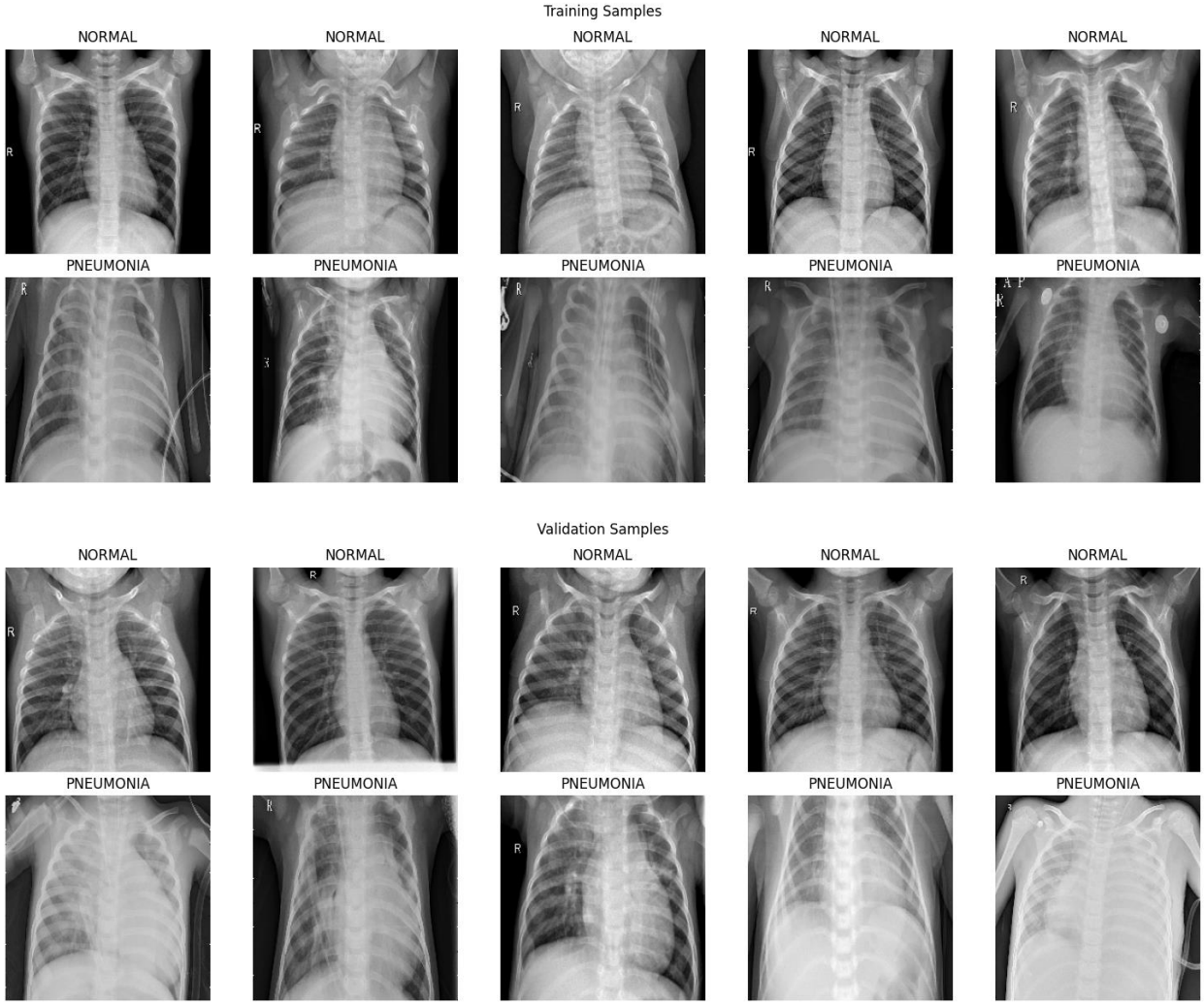
```
x = BatchNormalization()(x)
identity = Conv2D(1024, (1, 1), padding='same')(identity)
x = Add()([x, identity])
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Classification head
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.02))(x)
x = Dropout(0.6)(x)
outputs = Dense(1, activation='sigmoid')(x)

model = Model(inputs, outputs)

# Compile model
model.compile(
    loss='binary_crossentropy',
    optimizer=Adam(learning_rate=5e-5),
    metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]
)

# Class weights
class_weight_dict = {0: 1.5, 1: 0.5}  # Prioritize Normal class

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=20,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=7,
min_lr=1e-6)
checkpoint = ModelCheckpoint('best_model.keras', monitor='val_accuracy',
save_best_only=True, mode='max')

# Train model
history = model.fit(
    training_data,
    epochs=70,
    validation_data=validation_data,
    class_weight=class_weight_dict,
    callbacks=[early_stopping, reduce_lr, checkpoint]
)

# Load best model
model.load_weights('best_model.keras')
```

```
# Evaluate on validation data
val_loss, val_accuracy, val_precision, val_recall =
model.evaluate(validation_data)
print(f"Validation Accuracy: {val_accuracy:.4f}, Loss: {val_loss:.4f}, Precision:
{val_precision:.4f}, Recall: {val_recall:.4f}")

# Evaluate on test data
test_loss, test_accuracy, test_precision, test_recall =
model.evaluate(testing_data)
print(f"Test Accuracy: {test_accuracy:.4f}, Precision: {test_precision:.4f},
Recall: {test_recall:.4f}")

# Predict on test data
predictions = model.predict(testing_data)
predicted_classes = (predictions > 0.5).astype(int).flatten()
true_classes = testing_data.classes

# Confusion Matrix
cm = confusion_matrix(true_classes, predicted_classes)
print("Confusion Matrix:")
print(cm)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.savefig('confusion_matrix.png')
plt.show()

# Accuracy and Loss Plots
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.title('Training and Validation Loss')
plt.legend()
plt.tight_layout()
plt.savefig('accuracy_loss_curves.png')
plt.show()

# Classification Report
print("\nClassification Report for Test Set:")
print(classification_report(true_classes, predicted_classes,
target_names=['NORMAL', 'PNEUMONIA']))

# Data Quality Check: Visualize sample images
def visualize_samples(directory, num_samples=5):
    normal_dir = os.path.join(directory, 'NORMAL')
    pneumonia_dir = os.path.join(directory, 'PNEUMONIA')
    normal_files = [os.path.join(normal_dir, f) for f in os.listdir(normal_dir)
if f.endswith('.jpeg')][:num_samples]
    pneumonia_files = [os.path.join(pneumonia_dir, f) for f in
os.listdir(pneumonia_dir) if f.endswith('.jpeg')][:num_samples]

    plt.figure(figsize=(15, 6))
    for i, file in enumerate(normal_files + pneumonia_files):
        img = load_img(file, target_size=(224, 224), color_mode='grayscale')
        img_array = img_to_array(img) / 255.0
        plt.subplot(2, num_samples, i + 1)
        plt.imshow(img_array.squeeze(), cmap='gray')
        plt.title('NORMAL' if i < num_samples else 'PNEUMONIA')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig('sample_images.png')
    plt.show()

visualize_samples(train_dir)
```

```
filename      class
5197  /kaggle/input/chest-xray-pneumonia/chest_xray/...  PNEUMONIA
2399  /kaggle/input/chest-xray-pneumonia/chest_xray/...     NORMAL
2813  /kaggle/input/chest-xray-pneumonia/chest_xray/...     NORMAL
6588  /kaggle/input/chest-xray-pneumonia/chest_xray/...  PNEUMONIA
2920  /kaggle/input/chest-xray-pneumonia/chest_xray/...     NORMAL
Validation DataFrame head:
                                                filename      class
1346  /kaggle/input/chest-xray-pneumonia/chest_xray/...     NORMAL
1753  /kaggle/input/chest-xray-pneumonia/chest_xray/...     NORMAL
5878  /kaggle/input/chest-xray-pneumonia/chest_xray/...  PNEUMONIA
47    /kaggle/input/chest-xray-pneumonia/chest_xray/...     NORMAL
5699  /kaggle/input/chest-xray-pneumonia/chest_xray/...  PNEUMONIA
```

```
Training class counts:
 class
PNEUMONIA    2713
NORMAL       2712
Name: count, dtype: int64
Validation class counts:
 class
NORMAL       1163
PNEUMONIA    1162
Name: count, dtype: int64
Found 5425 validated image filenames belonging to 2 classes.
Found 2325 validated image filenames belonging to 2 classes.
Found 624 images belonging to 2 classes.
Training class distribution: [2712 2713]
Validation class distribution: [1163 1162]
Class indices: {'NORMAL': 0, 'PNEUMONIA': 1}
Validation batch images shape: (16, 224, 224, 1)
Validation batch labels shape: (16,)
Validation batch labels: [1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1.]
```



I0000 00:00:1745694223.003736 31 gpu_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB memory: -> device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5

I0000 00:00:1745694223.004405 31 gpu_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB memory: -> device: 1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5

Epoch 1/70

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

I0000 00:00:1745694239.730685 103 service.cc:148] XLA service 0x782500003e40 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:

I0000 00:00:1745694239.731681 103 service.cc:156] StreamExecutor device (0): Tesla T4, Compute Capability 7.5

I0000 00:00:1745694239.731707 103 service.cc:156] StreamExecutor device (1): Tesla T4, Compute Capability 7.5

I0000 00:00:1745694241.085003 103 cuda_dnn.cc:529] Loaded cuDNN version 90300

[1m 1/340[0m [37m━━[0m [1m3:05:22[0m 33s/step - accuracy: 0.5625 - loss: 9.8745 - precision: 0.5333 - recall: 1.0000I0000 00:00:1745694257.622962 103 device_compiler.h:188] Compiled cluster using XLA! This line is logged at most once for the lifetime of the process.

[1m340/340[0m [32m━━[0m[37m[0m [1m142s[0m 322ms/step - accuracy: 0.7961 - loss: 7.9182 - precision: 0.8840 - recall: 0.6825 - val_accuracy: 0.4998 - val_loss: 10.9099 - val_precision: 0.4998 - val_recall: 1.0000 - learning_rate: 5.0000e-05

Epoch 2/70

[1m340/340[0m [32m━━[0m[37m[0m [1m94s[0m 273ms/step - accuracy: 0.8848 - loss: 5.3298 - precision: 0.9639 - recall: 0.7965 - val_accuracy: 0.8877 - val_loss: 4.1780 - val_precision: 0.9395 - val_recall: 0.8287 - learning_rate: 5.0000e-05

Epoch 3/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 265ms/step - accuracy: 0.8996 - loss: 3.7124 - precision: 0.9708 - recall: 0.8258 - val_accuracy: 0.9239 - val_loss: 2.8855 - val_precision: 0.9833 - val_recall: 0.8623 - learning_rate: 5.0000e-05

Epoch 4/70

[1m340/340[0m [32m━━[0m[37m[0m [1m89s[0m 259ms/step - accuracy: 0.9086 - loss: 2.6471 - precision: 0.9781 - recall: 0.8387 -

val_accuracy: 0.8951 - val_loss: 2.2914 - val_precision: 0.9935 - val_recall: 0.7952 - learning_rate: 5.0000e-05

Epoch 5/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 264ms/step - accuracy: 0.9240 - loss: 1.9265 - precision: 0.9777 - recall: 0.8683 - val_accuracy: 0.9071 - val_loss: 1.6356 - val_precision: 0.9958 - val_recall: 0.8176 - learning_rate: 5.0000e-05

Epoch 6/70

[1m340/340[0m [32m━━[0m[37m[0m [1m92s[0m 267ms/step - accuracy: 0.9226 - loss: 1.4482 - precision: 0.9723 - recall: 0.8692 - val_accuracy: 0.9527 - val_loss: 1.1835 - val_precision: 0.9622 - val_recall: 0.9423 - learning_rate: 5.0000e-05

Epoch 7/70

[1m340/340[0m [32m━━[0m[37m[0m [1m92s[0m 267ms/step - accuracy: 0.9241 - loss: 1.1178 - precision: 0.9736 - recall: 0.8711 - val_accuracy: 0.9531 - val_loss: 0.9316 - val_precision: 0.9451 - val_recall: 0.9621 - learning_rate: 5.0000e-05

Epoch 8/70

[1m340/340[0m [32m━━[0m[37m[0m [1m92s[0m 265ms/step - accuracy: 0.9375 - loss: 0.8725 - precision: 0.9805 - recall: 0.8921 - val_accuracy: 0.9488 - val_loss: 0.7335 - val_precision: 0.9888 - val_recall: 0.9079 - learning_rate: 5.0000e-05

Epoch 9/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 261ms/step - accuracy: 0.9441 - loss: 0.6738 - precision: 0.9829 - recall: 0.9028 - val_accuracy: 0.9282 - val_loss: 0.6361 - val_precision: 0.9863 - val_recall: 0.8683 - learning_rate: 5.0000e-05

Epoch 10/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9311 - loss: 0.5565 - precision: 0.9791 - recall: 0.8810 - val_accuracy: 0.9475 - val_loss: 0.4940 - val_precision: 0.9943 - val_recall: 0.9002 - learning_rate: 5.0000e-05

Epoch 11/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 261ms/step - accuracy: 0.9514 - loss: 0.4326 - precision: 0.9851 - recall: 0.9170 -

val_accuracy: 0.9484 - val_loss: 0.4341 - val_precision: 0.9264 - val_recall: 0.9742 - learning_rate: 5.0000e-05

Epoch 12/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9353 - loss: 0.3787 - precision: 0.9765 - recall: 0.8935 - val_accuracy: 0.9243 - val_loss: 0.4058 - val_precision: 0.9940 - val_recall: 0.8537 - learning_rate: 5.0000e-05

Epoch 13/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 264ms/step - accuracy: 0.9516 - loss: 0.3025 - precision: 0.9842 - recall: 0.9183 - val_accuracy: 0.9583 - val_loss: 0.2922 - val_precision: 0.9917 - val_recall: 0.9243 - learning_rate: 5.0000e-05

Epoch 14/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9529 - loss: 0.2498 - precision: 0.9860 - recall: 0.9202 - val_accuracy: 0.9437 - val_loss: 0.3159 - val_precision: 0.9981 - val_recall: 0.8890 - learning_rate: 5.0000e-05

Epoch 15/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9500 - loss: 0.2262 - precision: 0.9868 - recall: 0.9130 - val_accuracy: 0.9510 - val_loss: 0.2408 - val_precision: 0.9962 - val_recall: 0.9053 - learning_rate: 5.0000e-05

Epoch 16/70

[1m340/340[0m [32m━━[0m[37m[0m [1m89s[0m 257ms/step - accuracy: 0.9529 - loss: 0.1864 - precision: 0.9859 - recall: 0.9187 - val_accuracy: 0.9153 - val_loss: 0.3822 - val_precision: 0.9990 - val_recall: 0.8313 - learning_rate: 5.0000e-05

Epoch 17/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 262ms/step - accuracy: 0.9538 - loss: 0.1652 - precision: 0.9864 - recall: 0.9225 - val_accuracy: 0.9652 - val_loss: 0.1787 - val_precision: 0.9954 - val_recall: 0.9346 - learning_rate: 5.0000e-05

Epoch 18/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9635 - loss: 0.1419 - precision: 0.9913 - recall: 0.9358 -

val_accuracy: 0.9622 - val_loss: 0.1544 - val_precision: 0.9590 - val_recall: 0.9656 - learning_rate: 5.0000e-05

Epoch 19/70

[1m340/340[0m [32m━━[0m[37m[0m [1m93s[0m 268ms/step - accuracy: 0.9623 - loss: 0.1288 - precision: 0.9879 - recall: 0.9364 - val_accuracy: 0.9234 - val_loss: 0.2591 - val_precision: 0.9980 - val_recall: 0.8485 - learning_rate: 5.0000e-05

Epoch 20/70

[1m340/340[0m [32m━━[0m[37m[0m [1m92s[0m 268ms/step - accuracy: 0.9527 - loss: 0.1352 - precision: 0.9831 - recall: 0.9209 - val_accuracy: 0.9669 - val_loss: 0.1244 - val_precision: 0.9831 - val_recall: 0.9501 - learning_rate: 5.0000e-05

Epoch 21/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 264ms/step - accuracy: 0.9609 - loss: 0.1109 - precision: 0.9864 - recall: 0.9333 - val_accuracy: 0.9553 - val_loss: 0.1574 - val_precision: 0.9962 - val_recall: 0.9139 - learning_rate: 5.0000e-05

Epoch 22/70

[1m340/340[0m [32m━━[0m[37m[0m [1m92s[0m 265ms/step - accuracy: 0.9692 - loss: 0.0903 - precision: 0.9910 - recall: 0.9464 - val_accuracy: 0.9677 - val_loss: 0.1219 - val_precision: 0.9936 - val_recall: 0.9415 - learning_rate: 5.0000e-05

Epoch 23/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9677 - loss: 0.0920 - precision: 0.9897 - recall: 0.9444 - val_accuracy: 0.9118 - val_loss: 0.2630 - val_precision: 1.0000 - val_recall: 0.8236 - learning_rate: 5.0000e-05

Epoch 24/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 262ms/step - accuracy: 0.9660 - loss: 0.0907 - precision: 0.9876 - recall: 0.9443 - val_accuracy: 0.9019 - val_loss: 0.3692 - val_precision: 1.0000 - val_recall: 0.8038 - learning_rate: 5.0000e-05

Epoch 25/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9670 - loss: 0.0813 - precision: 0.9882 - recall: 0.9447 -

val_accuracy: 0.9570 - val_loss: 0.1244 - val_precision: 0.9935 - val_recall: 0.9200 - learning_rate: 5.0000e-05

Epoch 26/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9672 - loss: 0.0819 - precision: 0.9886 - recall: 0.9447 - val_accuracy: 0.8761 - val_loss: 0.3150 - val_precision: 1.0000 - val_recall: 0.7522 - learning_rate: 5.0000e-05

Epoch 27/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 264ms/step - accuracy: 0.9709 - loss: 0.0717 - precision: 0.9913 - recall: 0.9482 - val_accuracy: 0.9789 - val_loss: 0.0769 - val_precision: 0.9912 - val_recall: 0.9664 - learning_rate: 5.0000e-05

Epoch 28/70

[1m340/340[0m [32m━━[0m[37m[0m [1m89s[0m 258ms/step - accuracy: 0.9697 - loss: 0.0681 - precision: 0.9908 - recall: 0.9489 - val_accuracy: 0.9695 - val_loss: 0.1005 - val_precision: 0.9573 - val_recall: 0.9828 - learning_rate: 5.0000e-05

Epoch 29/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9599 - loss: 0.0865 - precision: 0.9870 - recall: 0.9335 - val_accuracy: 0.9690 - val_loss: 0.0960 - val_precision: 0.9572 - val_recall: 0.9819 - learning_rate: 5.0000e-05

Epoch 30/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9720 - loss: 0.0618 - precision: 0.9877 - recall: 0.9548 - val_accuracy: 0.9535 - val_loss: 0.1234 - val_precision: 0.9203 - val_recall: 0.9931 - learning_rate: 5.0000e-05

Epoch 31/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9701 - loss: 0.0708 - precision: 0.9891 - recall: 0.9514 - val_accuracy: 0.8443 - val_loss: 0.5374 - val_precision: 1.0000 - val_recall: 0.6885 - learning_rate: 5.0000e-05

Epoch 32/70

[1m340/340[0m [32m━━[0m[37m[0m [1m89s[0m 259ms/step - accuracy: 0.9735 - loss: 0.0634 - precision: 0.9918 - recall: 0.9536 -

val_accuracy: 0.9690 - val_loss: 0.0914 - val_precision: 0.9919 - val_recall: 0.9458 - learning_rate: 5.0000e-05

Epoch 33/70

[1m340/340[0m [32m━━[0m[37m[0m [1m89s[0m 257ms/step - accuracy: 0.9702 - loss: 0.0635 - precision: 0.9881 - recall: 0.9514 - val_accuracy: 0.9738 - val_loss: 0.0800 - val_precision: 0.9885 - val_recall: 0.9587 - learning_rate: 5.0000e-05

Epoch 34/70

[1m340/340[0m [32m━━[0m[37m[0m [1m89s[0m 258ms/step - accuracy: 0.9764 - loss: 0.0529 - precision: 0.9942 - recall: 0.9588 - val_accuracy: 0.9751 - val_loss: 0.0699 - val_precision: 0.9742 - val_recall: 0.9759 - learning_rate: 5.0000e-05

Epoch 35/70

[1m340/340[0m [32m━━[0m[37m[0m [1m89s[0m 258ms/step - accuracy: 0.9725 - loss: 0.0645 - precision: 0.9896 - recall: 0.9551 - val_accuracy: 0.9669 - val_loss: 0.1066 - val_precision: 0.9945 - val_recall: 0.9389 - learning_rate: 5.0000e-05

Epoch 36/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 262ms/step - accuracy: 0.9712 - loss: 0.0587 - precision: 0.9922 - recall: 0.9511 - val_accuracy: 0.9480 - val_loss: 0.1546 - val_precision: 0.9990 - val_recall: 0.8967 - learning_rate: 5.0000e-05

Epoch 37/70

[1m340/340[0m [32m━━[0m[37m[0m [1m89s[0m 259ms/step - accuracy: 0.9747 - loss: 0.0595 - precision: 0.9946 - recall: 0.9554 - val_accuracy: 0.9561 - val_loss: 0.1385 - val_precision: 0.9953 - val_recall: 0.9165 - learning_rate: 5.0000e-05

Epoch 38/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 262ms/step - accuracy: 0.9649 - loss: 0.0687 - precision: 0.9871 - recall: 0.9411 - val_accuracy: 0.9708 - val_loss: 0.0949 - val_precision: 0.9991 - val_recall: 0.9423 - learning_rate: 5.0000e-05

Epoch 39/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9781 - loss: 0.0531 - precision: 0.9942 - recall: 0.9614 -

val_accuracy: 0.9785 - val_loss: 0.0661 - val_precision: 0.9728 - val_recall: 0.9845 - learning_rate: 5.0000e-05

Epoch 40/70

[1m340/340[0m [32m━━[0m[37m[0m [1m92s[0m 268ms/step - accuracy: 0.9765 - loss: 0.0520 - precision: 0.9895 - recall: 0.9631 - val_accuracy: 0.9703 - val_loss: 0.0886 - val_precision: 0.9513 - val_recall: 0.9914 - learning_rate: 5.0000e-05

Epoch 41/70

[1m340/340[0m [32m━━[0m[37m[0m [1m95s[0m 275ms/step - accuracy: 0.9803 - loss: 0.0484 - precision: 0.9910 - recall: 0.9699 - val_accuracy: 0.9858 - val_loss: 0.0493 - val_precision: 0.9904 - val_recall: 0.9811 - learning_rate: 5.0000e-05

Epoch 42/70

[1m340/340[0m [32m━━[0m[37m[0m [1m93s[0m 270ms/step - accuracy: 0.9743 - loss: 0.0545 - precision: 0.9908 - recall: 0.9582 - val_accuracy: 0.9802 - val_loss: 0.0636 - val_precision: 0.9938 - val_recall: 0.9664 - learning_rate: 5.0000e-05

Epoch 43/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9814 - loss: 0.0424 - precision: 0.9954 - recall: 0.9667 - val_accuracy: 0.9187 - val_loss: 0.2555 - val_precision: 0.9980 - val_recall: 0.8391 - learning_rate: 5.0000e-05

Epoch 44/70

[1m340/340[0m [32m━━[0m[37m[0m [1m89s[0m 259ms/step - accuracy: 0.9794 - loss: 0.0502 - precision: 0.9928 - recall: 0.9658 - val_accuracy: 0.9832 - val_loss: 0.0595 - val_precision: 0.9930 - val_recall: 0.9733 - learning_rate: 5.0000e-05

Epoch 45/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 261ms/step - accuracy: 0.9824 - loss: 0.0462 - precision: 0.9941 - recall: 0.9700 - val_accuracy: 0.9475 - val_loss: 0.1453 - val_precision: 0.9187 - val_recall: 0.9819 - learning_rate: 5.0000e-05

Epoch 46/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9793 - loss: 0.0506 - precision: 0.9934 - recall: 0.9655 -

val_accuracy: 0.9772 - val_loss: 0.0645 - val_precision: 0.9826 - val_recall: 0.9716 - learning_rate: 5.0000e-05

Epoch 47/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9815 - loss: 0.0435 - precision: 0.9933 - recall: 0.9697 - val_accuracy: 0.9720 - val_loss: 0.1000 - val_precision: 0.9973 - val_recall: 0.9466 - learning_rate: 5.0000e-05

Epoch 48/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 264ms/step - accuracy: 0.9786 - loss: 0.0539 - precision: 0.9907 - recall: 0.9663 - val_accuracy: 0.9716 - val_loss: 0.0810 - val_precision: 0.9832 - val_recall: 0.9596 - learning_rate: 5.0000e-05

Epoch 49/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 265ms/step - accuracy: 0.9816 - loss: 0.0493 - precision: 0.9956 - recall: 0.9670 - val_accuracy: 0.9871 - val_loss: 0.0479 - val_precision: 0.9991 - val_recall: 0.9750 - learning_rate: 1.0000e-05

Epoch 50/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 262ms/step - accuracy: 0.9885 - loss: 0.0307 - precision: 0.9943 - recall: 0.9822 - val_accuracy: 0.9862 - val_loss: 0.0486 - val_precision: 0.9829 - val_recall: 0.9897 - learning_rate: 1.0000e-05

Epoch 51/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9891 - loss: 0.0330 - precision: 0.9959 - recall: 0.9819 - val_accuracy: 0.9819 - val_loss: 0.0648 - val_precision: 0.9982 - val_recall: 0.9656 - learning_rate: 1.0000e-05

Epoch 52/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 262ms/step - accuracy: 0.9847 - loss: 0.0337 - precision: 0.9969 - recall: 0.9723 - val_accuracy: 0.9871 - val_loss: 0.0414 - val_precision: 0.9871 - val_recall: 0.9871 - learning_rate: 1.0000e-05

Epoch 53/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 260ms/step - accuracy: 0.9904 - loss: 0.0271 - precision: 0.9971 - recall: 0.9835 -

val_accuracy: 0.9871 - val_loss: 0.0512 - val_precision: 0.9982 - val_recall: 0.9759 - learning_rate: 1.0000e-05

Epoch 54/70

[1m340/340[0m [32m━━[0m[37m[0m [1m92s[0m 266ms/step - accuracy: 0.9894 - loss: 0.0294 - precision: 0.9967 - recall: 0.9818 - val_accuracy: 0.9897 - val_loss: 0.0432 - val_precision: 0.9965 - val_recall: 0.9828 - learning_rate: 1.0000e-05

Epoch 55/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 264ms/step - accuracy: 0.9915 - loss: 0.0269 - precision: 0.9981 - recall: 0.9845 - val_accuracy: 0.9892 - val_loss: 0.0418 - val_precision: 0.9965 - val_recall: 0.9819 - learning_rate: 1.0000e-05

Epoch 56/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9910 - loss: 0.0242 - precision: 0.9979 - recall: 0.9840 - val_accuracy: 0.9832 - val_loss: 0.0544 - val_precision: 0.9973 - val_recall: 0.9690 - learning_rate: 1.0000e-05

Epoch 57/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 262ms/step - accuracy: 0.9913 - loss: 0.0245 - precision: 0.9983 - recall: 0.9842 - val_accuracy: 0.9871 - val_loss: 0.0443 - val_precision: 0.9905 - val_recall: 0.9836 - learning_rate: 1.0000e-05

Epoch 58/70

[1m340/340[0m [32m━━[0m[37m[0m [1m92s[0m 267ms/step - accuracy: 0.9926 - loss: 0.0229 - precision: 0.9975 - recall: 0.9875 - val_accuracy: 0.9892 - val_loss: 0.0396 - val_precision: 0.9880 - val_recall: 0.9905 - learning_rate: 1.0000e-05

Epoch 59/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 261ms/step - accuracy: 0.9926 - loss: 0.0230 - precision: 0.9982 - recall: 0.9869 - val_accuracy: 0.9875 - val_loss: 0.0509 - val_precision: 0.9991 - val_recall: 0.9759 - learning_rate: 1.0000e-05

Epoch 60/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 261ms/step - accuracy: 0.9887 - loss: 0.0276 - precision: 0.9972 - recall: 0.9793 -

val_accuracy: 0.9875 - val_loss: 0.0415 - val_precision: 0.9991 - val_recall: 0.9759 - learning_rate: 1.0000e-05

Epoch 61/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9869 - loss: 0.0301 - precision: 0.9936 - recall: 0.9798 - val_accuracy: 0.9862 - val_loss: 0.0474 - val_precision: 0.9965 - val_recall: 0.9759 - learning_rate: 1.0000e-05

Epoch 62/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 261ms/step - accuracy: 0.9898 - loss: 0.0257 - precision: 0.9969 - recall: 0.9833 - val_accuracy: 0.9871 - val_loss: 0.0358 - val_precision: 0.9974 - val_recall: 0.9768 - learning_rate: 1.0000e-05

Epoch 63/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 261ms/step - accuracy: 0.9891 - loss: 0.0262 - precision: 0.9948 - recall: 0.9834 - val_accuracy: 0.9824 - val_loss: 0.0616 - val_precision: 1.0000 - val_recall: 0.9647 - learning_rate: 1.0000e-05

Epoch 64/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 261ms/step - accuracy: 0.9881 - loss: 0.0267 - precision: 0.9971 - recall: 0.9787 - val_accuracy: 0.9888 - val_loss: 0.0386 - val_precision: 0.9939 - val_recall: 0.9836 - learning_rate: 1.0000e-05

Epoch 65/70

[1m340/340[0m [32m━━[0m[37m[0m [1m90s[0m 262ms/step - accuracy: 0.9874 - loss: 0.0306 - precision: 0.9942 - recall: 0.9804 - val_accuracy: 0.9867 - val_loss: 0.0393 - val_precision: 0.9991 - val_recall: 0.9742 - learning_rate: 1.0000e-05

Epoch 66/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9926 - loss: 0.0197 - precision: 0.9983 - recall: 0.9869 - val_accuracy: 0.9824 - val_loss: 0.0532 - val_precision: 0.9991 - val_recall: 0.9656 - learning_rate: 1.0000e-05

Epoch 67/70

[1m340/340[0m [32m━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9922 - loss: 0.0204 - precision: 0.9992 - recall: 0.9850 -

val_accuracy: 0.9905 - val_loss: 0.0365 - val_precision: 0.9931 - val_recall: 0.9880 - learning_rate: 1.0000e-05

Epoch 68/70

[1m340/340[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m91s[0m 263ms/step - accuracy: 0.9903 - loss: 0.0231 - precision: 0.9958 - recall: 0.9847 - val_accuracy: 0.9824 - val_loss: 0.0528 - val_precision: 0.9754 - val_recall: 0.9897 - learning_rate: 1.0000e-05

Epoch 69/70

[1m340/340[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m92s[0m 267ms/step - accuracy: 0.9917 - loss: 0.0240 - precision: 0.9983 - recall: 0.9852 - val_accuracy: 0.9914 - val_loss: 0.0332 - val_precision: 0.9974 - val_recall: 0.9854 - learning_rate: 1.0000e-05

Epoch 70/70

[1m340/340[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m90s[0m 262ms/step - accuracy: 0.9904 - loss: 0.0240 - precision: 0.9972 - recall: 0.9835 - val_accuracy: 0.9837 - val_loss: 0.0601 - val_precision: 0.9973 - val_recall: 0.9699 - learning_rate: 1.0000e-05

[1m146/146[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m27s[0m 182ms/step - accuracy: 0.9891 - loss: 0.0398 - precision: 0.9973 - recall: 0.9810

Validation Accuracy: 0.9884, Loss: 0.0361, Precision: 0.9939, Recall: 0.9828

[1m39/39[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m6s[0m 162ms/step - accuracy: 0.9435 - loss: 0.1825 - precision: 0.5523 - recall: 0.6294

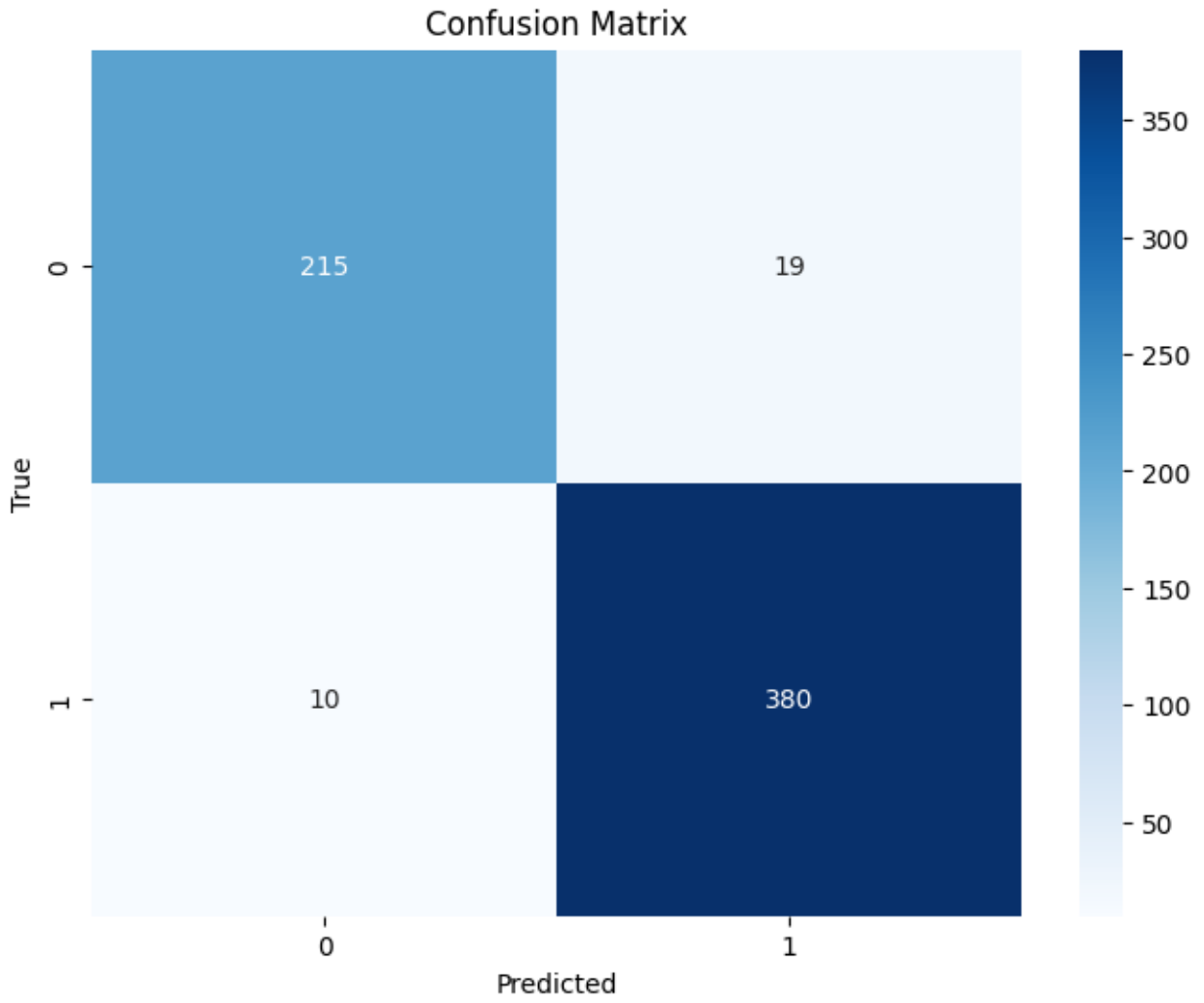
Test Accuracy: 0.9535, Precision: 0.9524, Recall: 0.9744

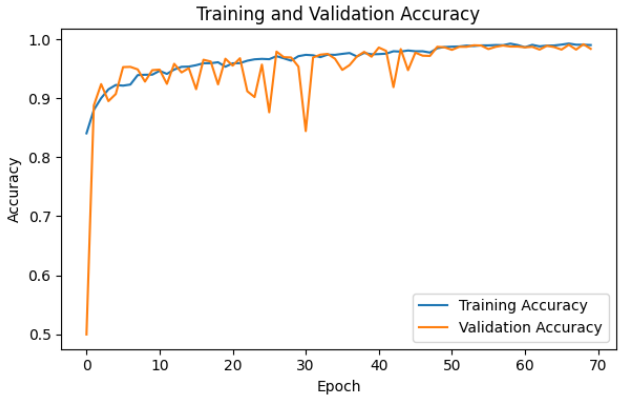
[1m39/39[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m4s[0m 73ms/step

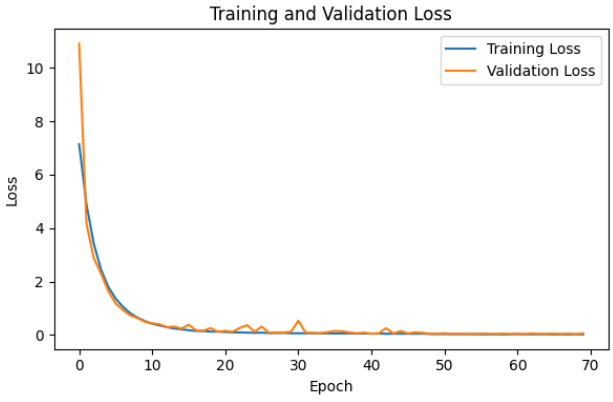
Confusion Matrix:

[[215 19]

 [10 380]]


Confusion Matrix
215
19
10
380
True
Predicted
0
1
350
300
250
200
150
100
50


Training and Validation Accuracy
Accuracy
Epoch
Training Accuracy
Validation Accuracy


Training and Validation Loss
Loss
Epoch
Training Loss
Validation Loss

Classification Report for Test Set:

	precision	recall	f1-score	support
NORMAL	0.96	0.92	0.94	234
PNEUMONIA	0.95	0.97	0.96	390
accuracy			0.95	624
macro avg	0.95	0.95	0.95	624
weighted avg	0.95	0.95	0.95	624