In the name of God.

In [1]:
```
# Step 1: Install required libraries
!pip install -q kaggle
!pip install -q opencv-python
!pip install -q numpy

# Step 2: Upload the kaggle.json file
from google.colab import files
files.upload()  # Upload the kaggle.json file here

# Step 3: Create folder for kaggle and transfer file
import os
os.makedirs('/root/.kaggle', exist_ok=True)
!cp kaggle.json /root/.kaggle/

# Step 4: Set permissions
!chmod 600 /root/.kaggle/kaggle.json

# Step 5: Download the dataset from Kaggle
!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia

# Step 6: Extract the ZIP file
import zipfile

with zipfile.ZipFile('chest-xray-pneumonia.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/chest-xray-pneumonia')

# Step 7: Check the directory structure
import os

# Display available directories
print("Directory structure after extraction:")
for root, dirs, files in os.walk('/content/chest-xray-pneumonia'):
    print(f"Folder: {root} - Files: {files}")
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle (1).json to kaggle (1).json
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
Dataset URL: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia (https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia)
License(s): other
Downloading chest-xray-pneumonia.zip to /content
100% 2.29G/2.29G [00:10<00:00, 228MB/s]
100% 2.29G/2.29G [00:10<00:00, 224MB/s]
Directory structure after extraction:
Folder: /content/chest-xray-pneumonia - Files: []
Folder: /content/chest-xray-pneumonia/chest_xray - Files: []
Folder: /content/chest-xray-pneumonia/chest_xray/val - Files: []
```

In [2]:

```
import shutil

# Remove extra folders
shutil.rmtree('/content/chest-xray-pneumonia/chest_xray/__MACOSX', ignore_errors=True)
shutil.rmtree('/content/chest-xray-pneumonia/chest_xray/chest_xray', ignore_errors=True)

print("Deleted extra folders.")
```

```
Deleted extra folders.
```

In [3]:
```
import os
import cv2
import numpy as np

def load_images(data_path):
    images = []
    labels = []

    for label in ['NORMAL', 'PNEUMONIA']:
        label_path = os.path.join(data_path, label)
        if not os.path.exists(label_path):
            print(f'The file does not exist: {label_path}')
            continue

        for img_file in os.listdir(label_path):
            img_path = os.path.join(label_path, img_file)
            img = cv2.imread(img_path)
            if img is None:
                print(f'Image failed to load: {img_path}')
                continue
            img = cv2.resize(img, (128, 128))  # Resize to 128x128
            images.append(img)
            labels.append(int(label == 'PNEUMONIA'))  # Labels are numerical

    return np.array(images), np.array(labels)

# Upload images
data_path = '/content/chest-xray-pneumonia/chest_xray'
images, labels = load_images(data_path)

print(f'Number of images uploaded: {len(images)}')
print(f'Dimensions of each image: {images.shape[1:]}')
print(f'Tags: {labels[:10]}')  # Showing the first 10 tags
```

```
The file does not exist: /content/chest-xray-pneumonia/chest_xray/NORMAL
The file does not exist: /content/chest-xray-pneumonia/chest_xray/PNEUMONIA
Number of images uploaded: 0
Dimensions of each image: ()
Tags: []
```

In [4]:
```
import os

# Show directory structure
base_path = '/content/chest-xray-pneumonia'
for dirpath, dirnames, filenames in os.walk(base_path):
    print(f'Folder: {dirpath}, Files: {filenames}')
```

```
Folder: /content/chest-xray-pneumonia, Files: []
Folder: /content/chest-xray-pneumonia/chest_xray, Files: []
Folder: /content/chest-xray-pneumonia/chest_xray/val, Files: []
Folder: /content/chest-xray-pneumonia/chest_xray/val/PNEUMONIA, Files: ['person1951_bacteria_4882.jpeg', 'pe
rson1952_bacteria_4883.jpeg', 'person1946_bacteria_4874.jpeg', 'person1950_bacteria_4881.jpeg', 'person1946_
bacteria_4875.jpeg', 'person1947_bacteria_4876.jpeg', 'person1954_bacteria_4886.jpeg', 'person1949_bacteria_
4880.jpeg']
Folder: /content/chest-xray-pneumonia/chest_xray/val/NORMAL, Files: ['NORMAL2-IM-1442-0001.jpeg', 'NORMAL2-I
M-1437-0001.jpeg', 'NORMAL2-IM-1440-0001.jpeg', 'NORMAL2-IM-1427-0001.jpeg', 'NORMAL2-IM-1436-0001.jpeg', 'N
ORMAL2-IM-1430-0001.jpeg', 'NORMAL2-IM-1431-0001.jpeg', 'NORMAL2-IM-1438-0001.jpeg']
Folder: /content/chest-xray-pneumonia/chest_xray/test, Files: []
Folder: /content/chest-xray-pneumonia/chest_xray/test/PNEUMONIA, Files: ['person150_bacteria_715.jpeg', 'per
son122_bacteria_583.jpeg', 'person3_virus_16.jpeg', 'person175_bacteria_833.jpeg', 'person46_virus_96.jpeg',
'person59_virus_116.jpeg', 'person120_bacteria_570.jpeg', 'person1633_virus_2829.jpeg', 'person1608_virus_27
86.jpeg', 'person120_bacteria_572.jpeg', 'person117_bacteria_553.jpeg', 'person155_bacteria_730.jpeg', 'pers
on85_bacteria_424.jpeg', 'person96_bacteria_466.jpeg', 'person85_bacteria_419.jpeg', 'person1673_virus_2889.
jpeg', 'person161_bacteria_759.jpeg', 'person117_bacteria_557.jpeg', 'person100_bacteria_478.jpeg', 'person1
21_bacteria_579.jpeg', 'person1647_virus_2848.jpeg', 'person71_virus_132.jpeg', 'person143_bacteria_689.jpe
g', 'person22_virus_55.jpeg', 'person96_bacteria_465.jpeg', 'person80_bacteria_392.jpeg', 'person136_bacteri
```

In [5]:
```
import os

data_path = '/content/chest-xray-pneumonia/chest_xray'
categories = ['train', 'val', 'test']
subfolders = ['NORMAL', 'PNEUMONIA']

for category in categories:
    for subfolder in subfolders:
        folder_path = os.path.join(data_path, category, subfolder)
        if os.path.exists(folder_path):
            files = os.listdir(folder_path)
            print(f'Folder: {folder_path} - Number of files: {len(files)}')
        else:
            print(f'The folder does not exist: {folder_path}')
```

```
Folder: /content/chest-xray-pneumonia/chest_xray/train/NORMAL - Number of files: 1341
Folder: /content/chest-xray-pneumonia/chest_xray/train/PNEUMONIA - Number of files: 3875
Folder: /content/chest-xray-pneumonia/chest_xray/val/NORMAL - Number of files: 8
Folder: /content/chest-xray-pneumonia/chest_xray/val/PNEUMONIA - Number of files: 8
Folder: /content/chest-xray-pneumonia/chest_xray/test/NORMAL - Number of files: 234
Folder: /content/chest-xray-pneumonia/chest_xray/test/PNEUMONIA - Number of files: 390
```

In [6]:
```
import os
import cv2
import numpy as np

def load_images(data_path):
    images = []
    labels = []

    for category in ['train', 'val', 'test']:
        for label in ['NORMAL', 'PNEUMONIA']:
            folder_path = os.path.join(data_path, category, label)
            if os.path.exists(folder_path):
                for img_file in os.listdir(folder_path):
                    img_path = os.path.join(folder_path, img_file)
                    img = cv2.imread(img_path)
                    if img is not None:
                        img = cv2.resize(img, (128, 128))  # Resize to 128x128
                        images.append(img)
                        labels.append(0 if label == 'NORMAL' else 1)  # Labels: 0 for normal and 1 for pneumonia
                    else:
                        print(f'Image failed to load: {img_path}')
            else:
                print(f'The folder does not exist: {folder_path}')

    return np.array(images), np.array(labels)

# Upload images
data_path = '/content/chest-xray-pneumonia/chest_xray'
images, labels = load_images(data_path)

print(f'Number of images uploaded: {len(images)}')
print(f'Dimensions of each image: {images[0].shape}' if len(images) > 0 else 'Dimensions of each image: (no imag
print(f'Tags: {np.unique(labels, return_counts=True)}')
```

```
Number of images uploaded: 5856
Dimensions of each image: (128, 128, 3)
Tags: (array([0, 1]), array([1583, 4273]))
```

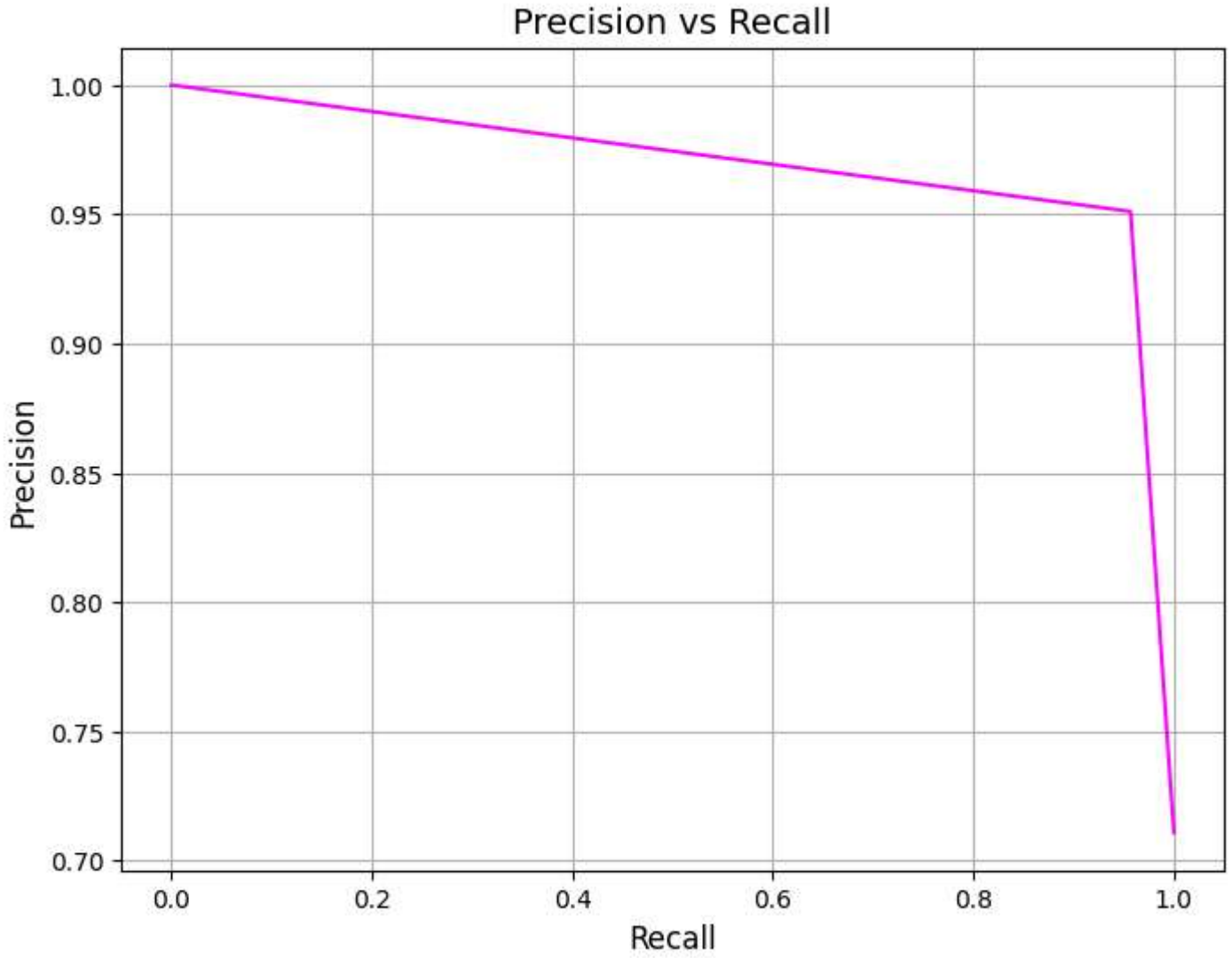
In [7]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Dividing data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)

# Converting tags to category format (one-hot encoding)
y_train = to_categorical(y_train, 2)
y_test = to_categorical(y_test, 2)

# Construction of CNN model
model = Sequential()

# Convolutional layers and pooling
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Dense and Dropout layers
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

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

# Model training
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Model evaluation
loss, accuracy = model.evaluate(X_test, y_test)
```

```
print(f'Model Accuracy: {accuracy * 100:.2f}%')
```

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

Epoch 1/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 22s 63ms/step - accuracy: 0.7199 - loss: 26.8553 - val_accuracy: 0.8882 - val_los
s: 0.2735
Epoch 2/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.8829 - loss: 0.3045 - val_accuracy: 0.9019 - val_loss:
0.2489
Epoch 3/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - accuracy: 0.9212 - loss: 0.2319 - val_accuracy: 0.9206 - val_loss:
0.2017
Epoch 4/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.9420 - loss: 0.1587 - val_accuracy: 0.9283 - val_loss:
0.2045
Epoch 5/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.9445 - loss: 0.1547 - val_accuracy: 0.9480 - val_loss:
0.1474
Epoch 6/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9570 - loss: 0.1196 - val_accuracy: 0.9488 - val_loss:
0.1571
Epoch 7/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - accuracy: 0.9620 - loss: 0.1071 - val_accuracy: 0.9386 - val_loss:
0.1750
Epoch 8/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.9661 - loss: 0.0956 - val_accuracy: 0.9386 - val_loss:
0.1714
Epoch 9/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.9640 - loss: 0.0926 - val_accuracy: 0.9505 - val_loss:
0.1752
Epoch 10/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9661 - loss: 0.0810 - val_accuracy: 0.9437 - val_loss:
0.1882
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9395 - loss: 0.1975
Model Accuracy: 94.37%
```

In [8]:
```
# Precision vs Recall

import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, classification_report, confusion_matrix
import numpy as np

# model training
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Predictions
y_pred = model.predict(X_test)

# Convert one-hot encoded labels to original labels
y_test_labels = np.argmax(y_test, axis=1)
y_pred_labels = np.argmax(y_pred, axis=1)

# Calculation of precision and recall
precision, recall, thresholds = precision_recall_curve(y_test_labels, y_pred_labels)

# Full report including precision, recall, f1-score and accuracy
print(classification_report(y_test_labels, y_pred_labels))

# Draw a precision vs recall graph
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='magenta')
plt.title('Precision vs Recall', fontsize=14)
plt.xlabel('Recall', fontsize=12)
plt.ylabel('Precision', fontsize=12)
plt.grid(True)
plt.show()

# Calculating the clutter matrix to see the number of True Positives, True Negatives, False Positives, False Neg
conf_matrix = confusion_matrix(y_test_labels, y_pred_labels)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Epoch 1/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.9713 - loss: 0.0815 - val_accuracy: 0.9471 - val_loss:
0.1863
Epoch 2/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.9764 - loss: 0.0599 - val_accuracy: 0.9462 - val_loss:
0.2145
Epoch 3/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.9706 - loss: 0.0760 - val_accuracy: 0.9445 - val_loss:
0.2042
Epoch 4/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.9791 - loss: 0.0595 - val_accuracy: 0.9514 - val_loss:
0.1638
Epoch 5/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.9853 - loss: 0.0444 - val_accuracy: 0.9317 - val_loss:
0.2396
Epoch 6/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.9622 - loss: 0.1045 - val_accuracy: 0.9514 - val_loss:
0.2055
Epoch 7/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.9743 - loss: 0.0694 - val_accuracy: 0.9420 - val_loss:
0.2358
Epoch 8/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.9785 - loss: 0.0622 - val_accuracy: 0.9522 - val_loss:
0.1909
Epoch 9/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.9843 - loss: 0.0445 - val_accuracy: 0.9428 - val_loss:
0.2292
Epoch 10/10
147/147 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.9823 - loss: 0.0437 - val_accuracy: 0.9343 - val_loss:
0.2508
37/37 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step
              precision    recall  f1-score   support

           0       0.89      0.88      0.89       339
           1       0.95      0.96      0.95       833

    accuracy                           0.93      1172
   macro avg       0.92      0.92      0.92      1172
weighted avg       0.93      0.93      0.93      1172
```

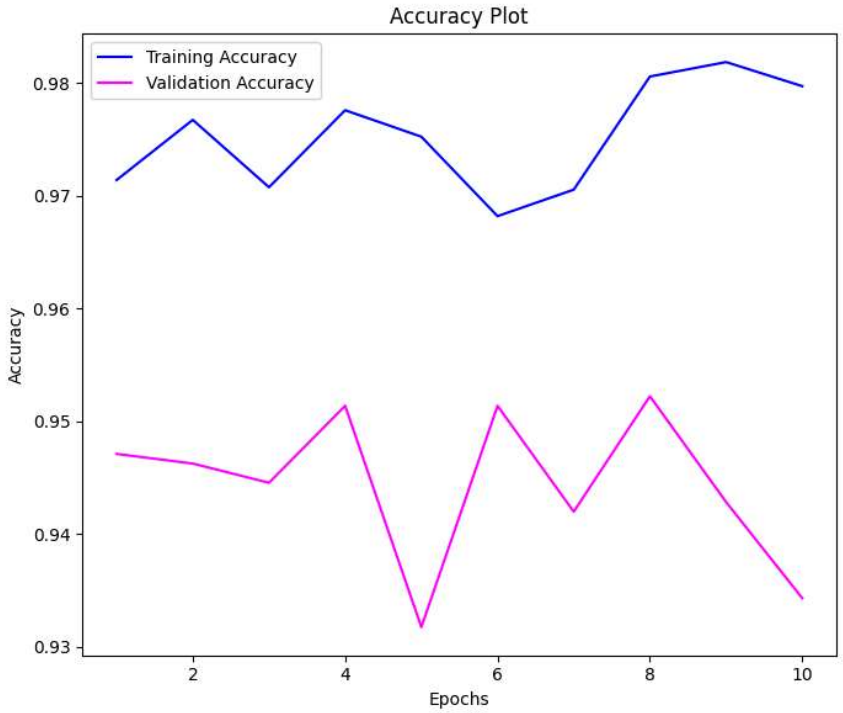


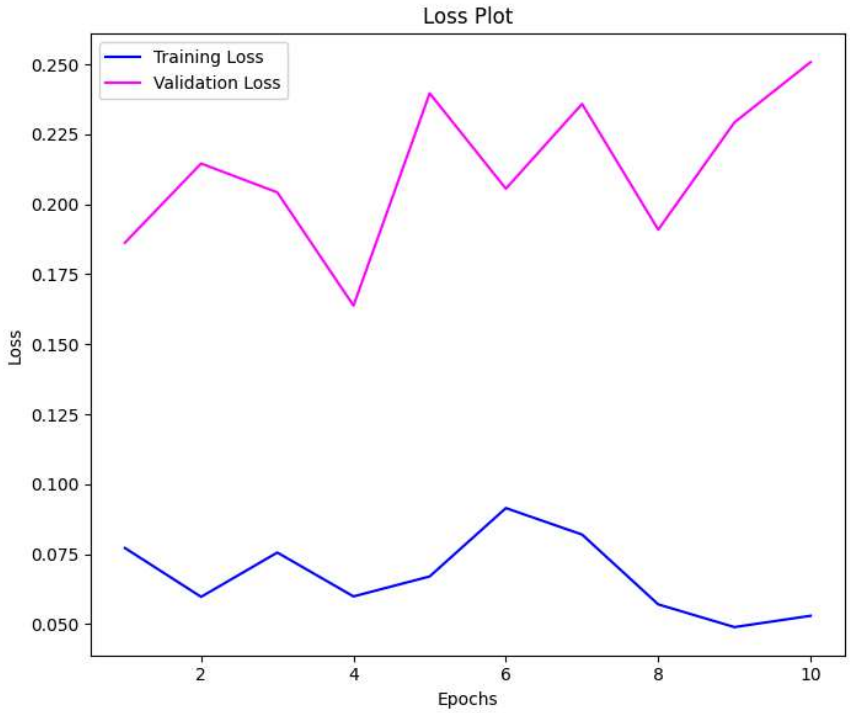

```
Confusion Matrix:
[[298  41]
 [ 36 797]]
```

In [9]:
```
# Save the model
model.save('pneumonia_detection_model.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('m
y_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

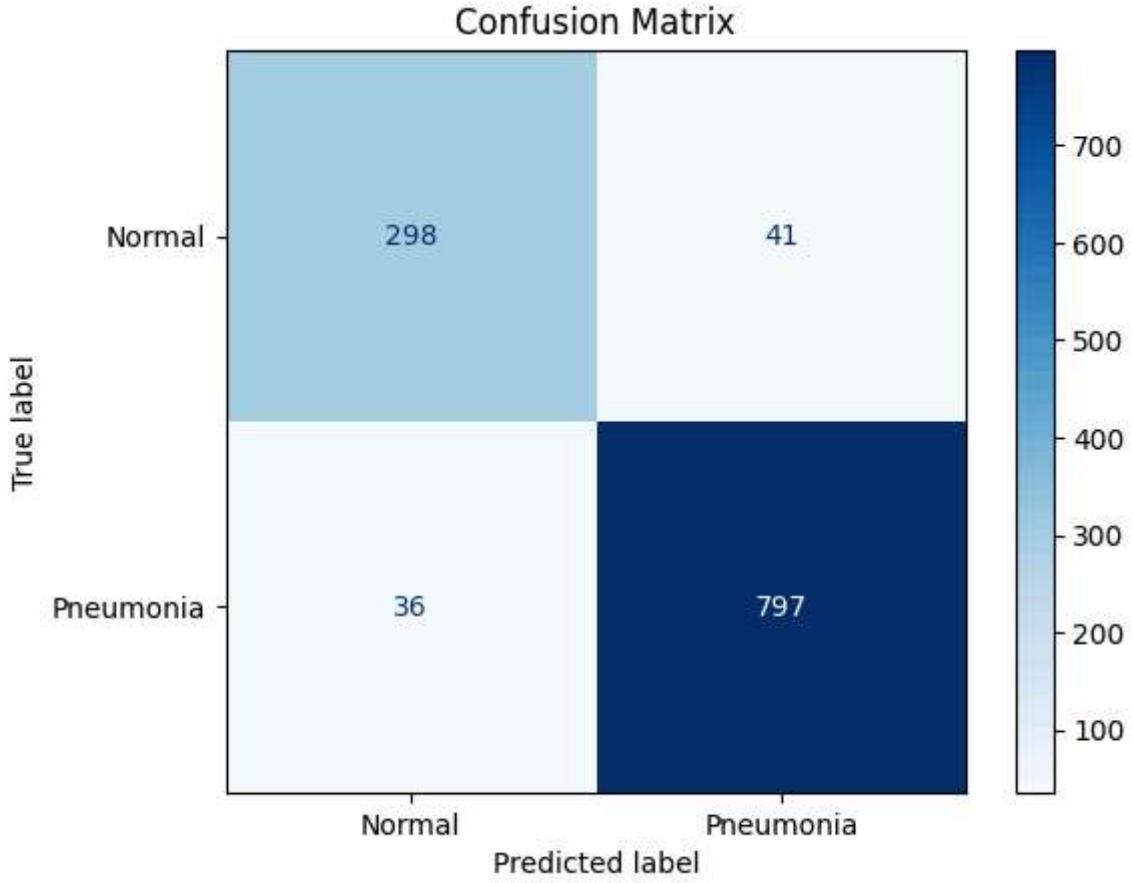
In [10]:
```
import matplotlib.pyplot as plt

# Suppose history contains the history of model training
# history = model.fit(...)

# Extract accuracy and error from history
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)

# Draw the accuracy graph
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, color='blue', label='Training Accuracy')
plt.plot(epochs, val_accuracy, color='magenta', label='Validation Accuracy')
plt.title('Accuracy Plot')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Draw an error diagram
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, color='blue', label='Training Loss')
plt.plot(epochs, val_loss, color='magenta', label='Validation Loss')
plt.title('Loss Plot')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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


Accuracy Plot
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.98
0.97
0.96
0.95
0.94
0.93
2
4
6
8
10

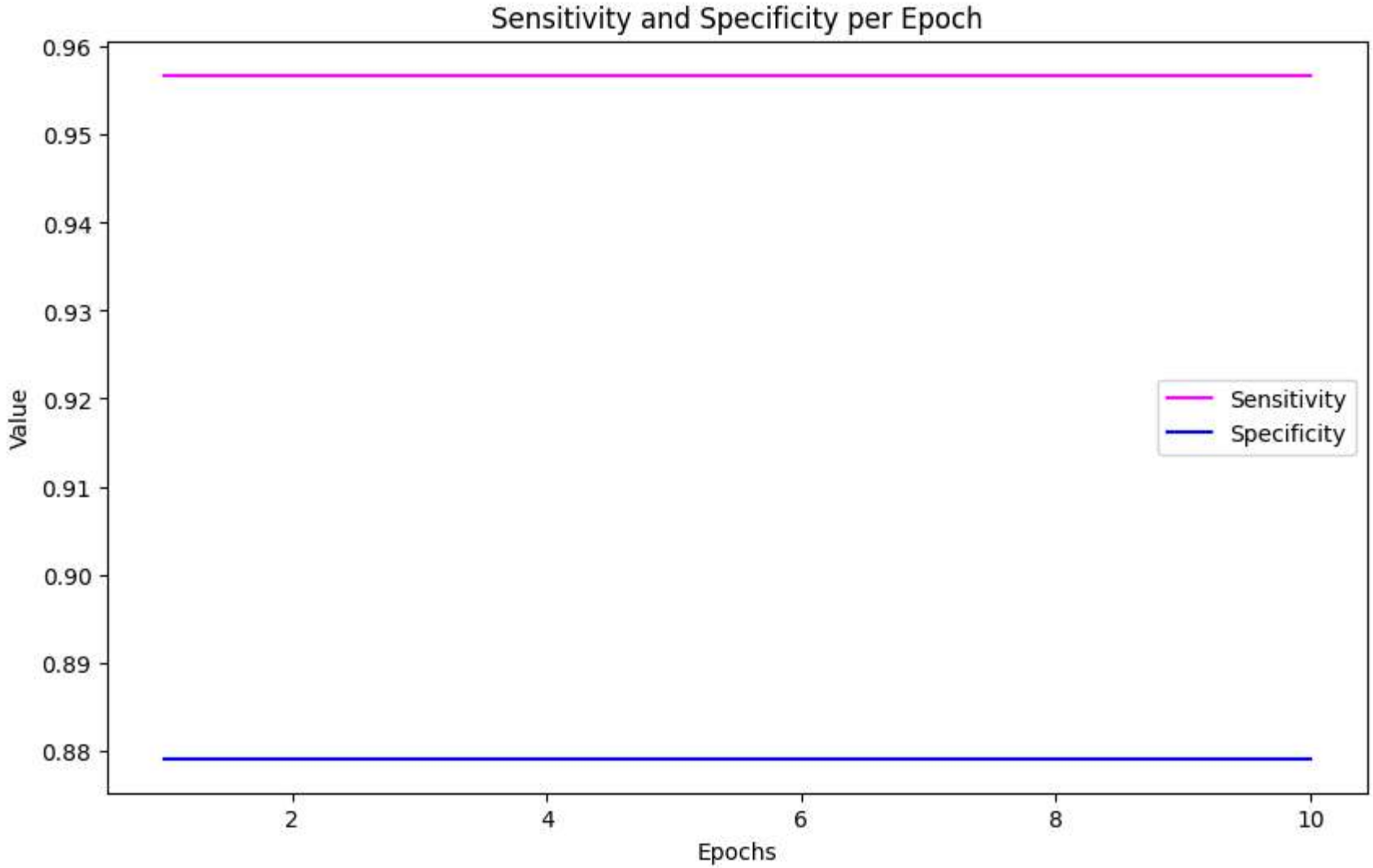

Loss Plot
Training Loss
Validation Loss
Loss
Epochs
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
2
4
6
8
10

In [11]:
```
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Predicting classes on test data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Calculation of confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)
tn, fp, fn, tp = cm.ravel()

# Calculation of sensitivity and specificity
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)

print(f'Sensitivity: {sensitivity:.2f}')
print(f'Specificity: {specificity:.2f}')

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Normal', 'Pneumonia'])
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()

# Draw a sensitivity and specificity diagram
epochs = range(1, len(history.history['accuracy']) + 1)

# Draw the diagram of sensitivity and specificity
plt.figure(figsize=(10, 6))
plt.plot(epochs, [sensitivity] * len(epochs), 'b', label='Sensitivity', color='magenta')
plt.plot(epochs, [specificity] * len(epochs), 'b', label='Specificity', color='blue')
plt.title('Sensitivity and Specificity per Epoch')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
plt.show()
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Sensitivity: 0.96
Specificity: 0.88
```



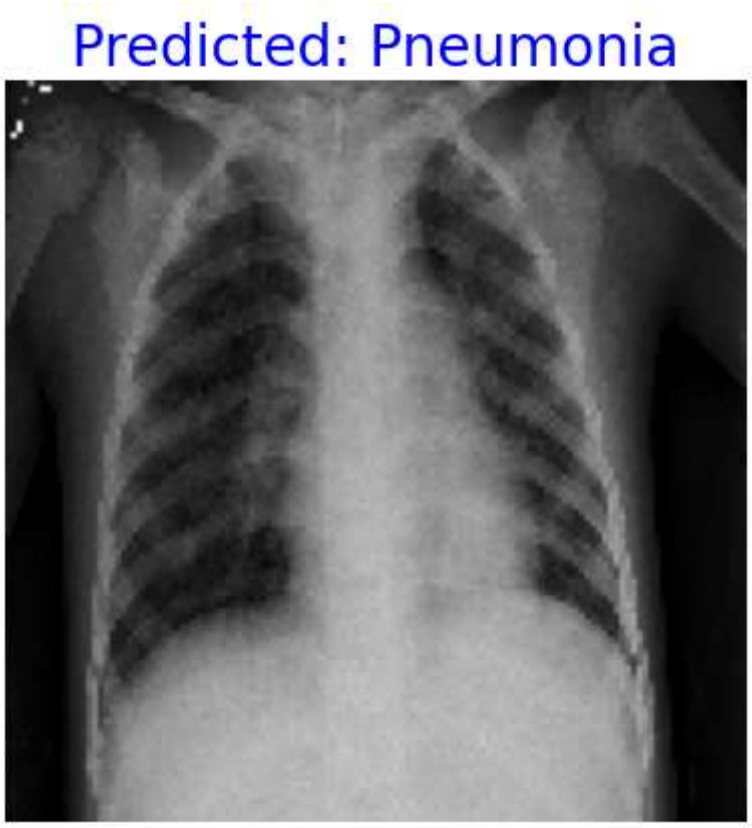

```
<ipython-input-11-705c81530d1c>:32: UserWarning: color is redundantly defined by the 'color' keyword argument
and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argument will take precedence.
  plt.plot(epochs, [sensitivity] * len(epochs), 'b', label='Sensitivity', color='magenta')
<ipython-input-11-705c81530d1c>:33: UserWarning: color is redundantly defined by the 'color' keyword argument
and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argument will take precedence.
  plt.plot(epochs, [specificity] * len(epochs), 'b', label='Specificity', color='blue')
```


Sensitivity and Specificity per Epoch
0.96
0.95
0.94
0.93
0.92
0.91
0.90
0.89
0.88
Value
2
4
6
8
10
Epochs
Sensitivity
Specificity

In [12]:
```
import matplotlib.pyplot as plt
import cv2
import numpy as np
import random

# Randomly select an image from the dataset
index = random.randint(0, len(images) - 1)  # 'images' is a list of images
sample_image = images[index]
sample_label = labels[index]  # True label related to the image

# Reshape the sample image for prediction
sample_image_for_prediction = np.expand_dims(sample_image, axis=0)  # Expand dimensions to match model input

# Step 1: Model prediction for the selected image
predicted_proba = model.predict(sample_image_for_prediction)
predicted_label = np.argmax(predicted_proba, axis=1)[0]  # Get predicted class label

# Determine disease based on predicted label
if predicted_label == 0:
    predicted_disease = "Normal"
else:
    predicted_disease = "Pneumonia"

# Display the image and the predicted disease
plt.imshow(cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB))
plt.axis('off')  # Disable image axes
plt.title(f'Predicted: {predicted_disease}', fontsize=20, color='blue')
plt.show()

# Step 2: Now, check the true label and compare
if sample_label == 0:
    actual_disease = "Normal"
else:
    actual_disease = "Pneumonia"

print(f"Actual label: {actual_disease}")

# Step 3: Check if prediction is correct
if predicted_label == sample_label:
    print("Prediction is correct.")
else:
```

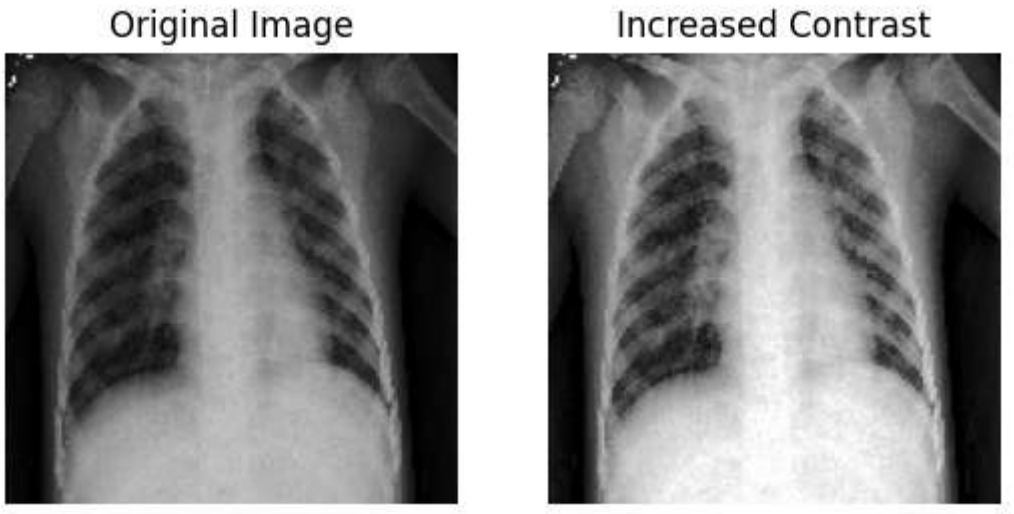
```
    print("Prediction is incorrect.")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 550ms/step
```

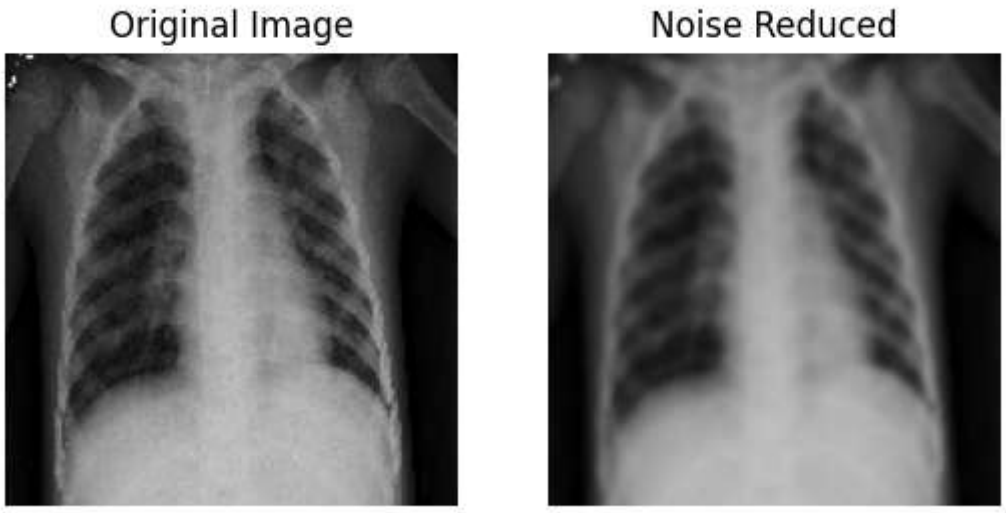



```
Actual label: Pneumonia
Prediction is correct.
```

In [13]:
```
# Increase image contrast
def increase_contrast(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    equalized_image = cv2.equalizeHist(gray_image)
    return equalized_image

# Select an image
sample_image = images[index]
contrast_image = increase_contrast(sample_image)

# Display original image and image with high contrast
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(contrast_image, cmap='gray')
plt.title('Increased Contrast')
plt.axis('off')

plt.show()
```



In [14]:
```
# Reduce image noise
def reduce_noise(image):
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
    return blurred_image

# noise reduction
noise_reduced_image = reduce_noise(sample_image)

# Display the original image and the image with noise reduction
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(noise_reduced_image, cv2.COLOR_BGR2RGB))
plt.title('Noise Reduced')
plt.axis('off')

plt.show()
```



In [15]:
```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# 1. Upload images with pre-processing
def load_images(data_path):
    images = []
    labels = []

    for label in ['NORMAL', 'PNEUMONIA']:
        label_path = os.path.join(data_path, label)
        for img_file in os.listdir(label_path):
            img_path = os.path.join(label_path, img_file)
            img = cv2.imread(img_path)

            # Convert to gray image
            img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            # noise reduction
            img_denoised = cv2.GaussianBlur(img_gray, (5, 5), 0)

            # Increase contrast
            img_eq = cv2.equalizeHist(img_denoised)

            # Resize to 128x128
            img_resized = cv2.resize(img_eq, (128, 128))
            images.append(img_resized)
            labels.append(int(label == 'PNEUMONIA'))

    return np.array(images), np.array(labels)

# Upload images
data_path = '/content/chest-xray-pneumonia/chest_xray/train'
images, labels = load_images(data_path)

# 2. Dividing data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)
```

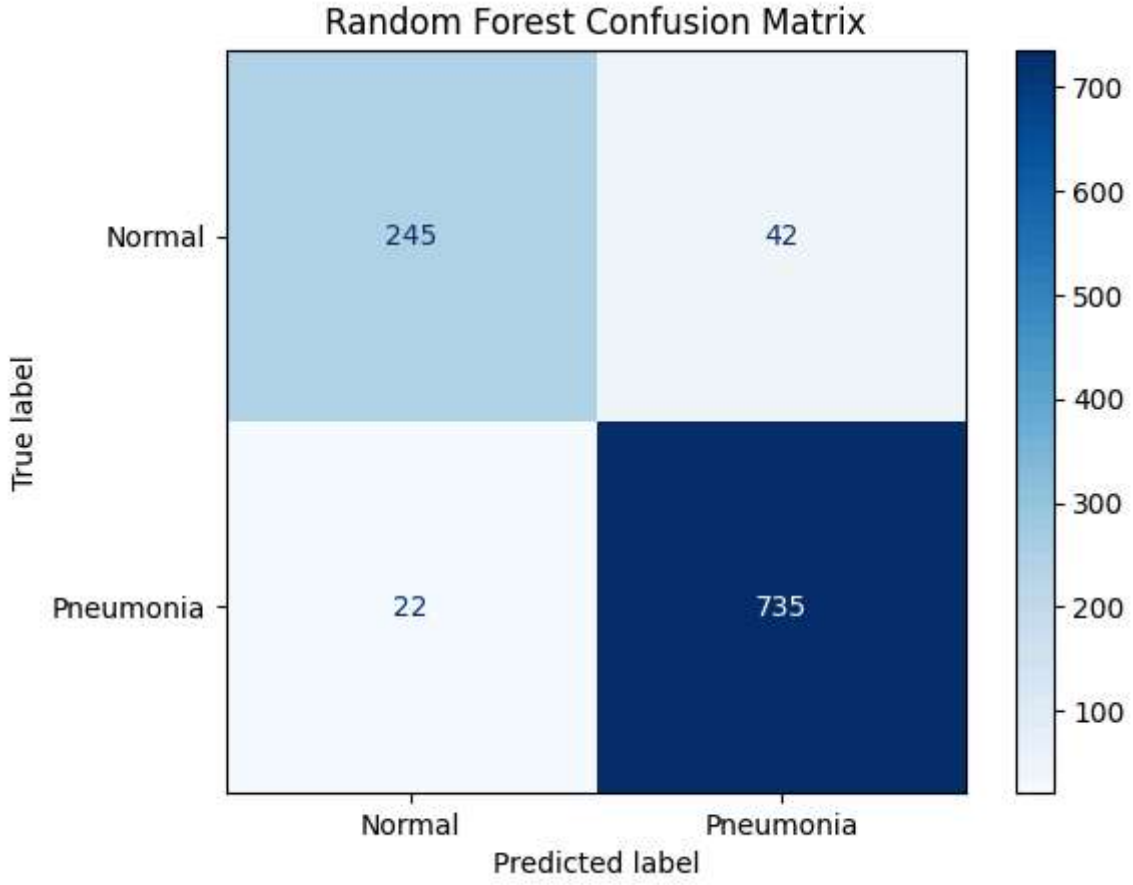
```
# 3. Data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True)

# 4.Create a model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(2, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# 5. Model training
history = model.fit(datagen.flow(X_train[..., np.newaxis], y_train, batch_size=32),
                    validation_data=(X_val[..., np.newaxis], y_val),
                    epochs=10)

# 6. Model evaluation
test_loss, test_accuracy = model.evaluate(X_val[..., np.newaxis], y_val)
print(f'Test accuracy: {test_accuracy:.2f}')
```

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

Epoch 1/10

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarnin
g: 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 ig
nored.
  self._warn_if_super_not_called()
```

```
131/131 ━━━━━━━━━━━━━━━━━━━━ 13s 70ms/step - accuracy: 0.7097 - loss: 53.2976 - val_accuracy: 0.8621 - val_los
s: 0.3833
Epoch 2/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 8s 56ms/step - accuracy: 0.8798 - loss: 0.3126 - val_accuracy: 0.8946 - val_loss:
0.2969
Epoch 3/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 7s 49ms/step - accuracy: 0.8796 - loss: 0.3061 - val_accuracy: 0.9109 - val_loss:
0.2462
Epoch 4/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 7s 51ms/step - accuracy: 0.8975 - loss: 0.2567 - val_accuracy: 0.9071 - val_loss:
0.2360
Epoch 5/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 7s 54ms/step - accuracy: 0.9108 - loss: 0.2287 - val_accuracy: 0.9080 - val_loss:
0.2894
Epoch 6/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 7s 46ms/step - accuracy: 0.9166 - loss: 0.2208 - val_accuracy: 0.9195 - val_loss:
0.2195
Epoch 7/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 8s 59ms/step - accuracy: 0.9226 - loss: 0.1984 - val_accuracy: 0.9186 - val_loss:
0.2075
Epoch 8/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 6s 46ms/step - accuracy: 0.9213 - loss: 0.2040 - val_accuracy: 0.9310 - val_loss:
0.1953
Epoch 9/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 8s 56ms/step - accuracy: 0.9249 - loss: 0.1934 - val_accuracy: 0.9387 - val_loss:
0.1755
Epoch 10/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 47ms/step - accuracy: 0.9296 - loss: 0.1752 - val_accuracy: 0.9033 - val_loss:
0.2520
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9009 - loss: 0.2465
Test accuracy: 0.90
```

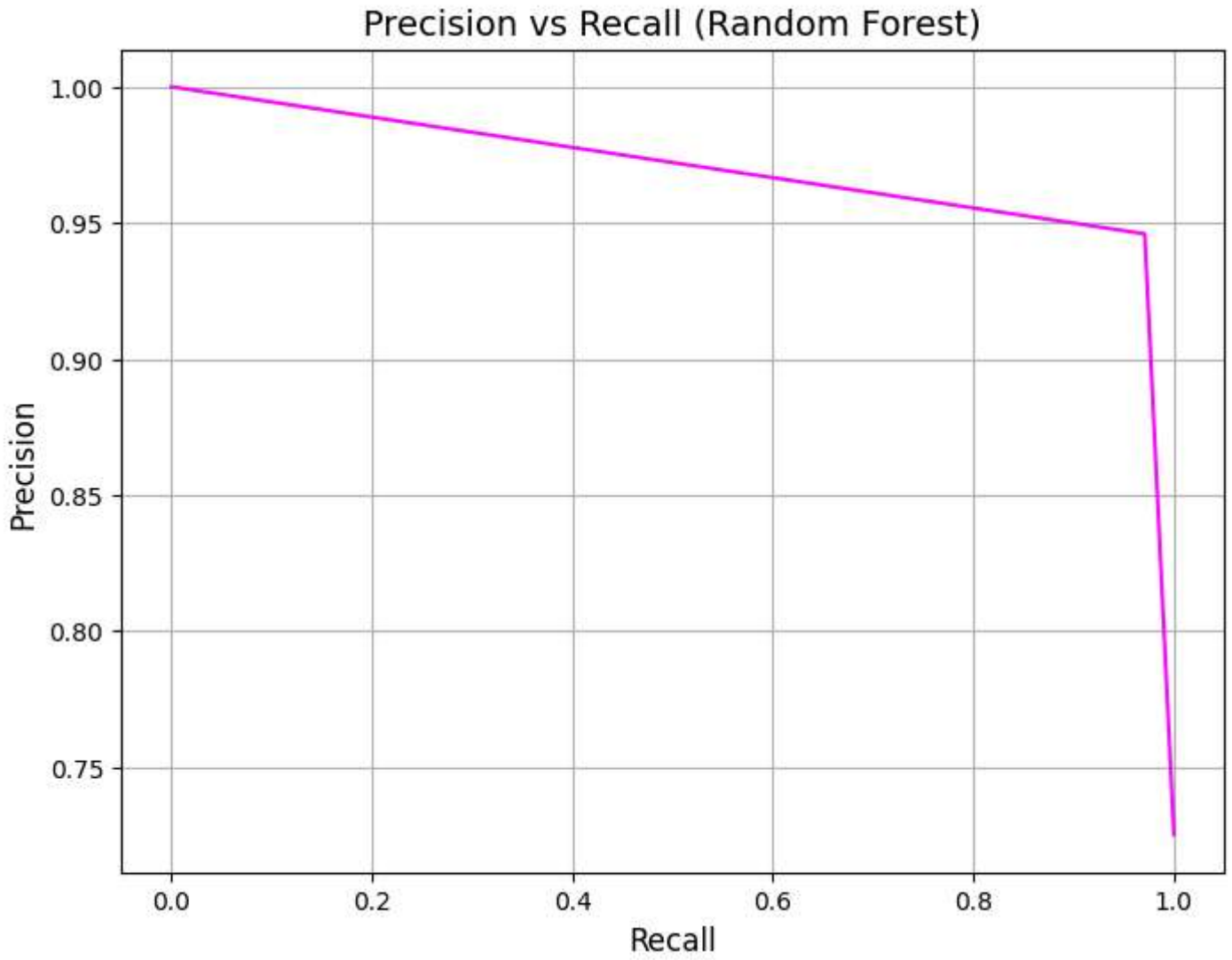
In [16]:
```
# Random forest

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split

# Dividing data into training and testing sets (pre-processing as before)
X_train, X_test, y_train, y_test = train_test_split(images.reshape(len(images), -1), labels, test_size=0.2, rand

# Construction of Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Model training
rf_model.fit(X_train, y_train)

# Prediction on test data
y_pred_rf = rf_model.predict(X_test)

# Calculate accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Random Forest Validation Accuracy: {accuracy_rf * 100:.2f}%')

# Calculation of confusion matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
tn_rf, fp_rf, fn_rf, tp_rf = cm_rf.ravel()

# Calculation of sensitivity and specificity
sensitivity_rf = tp_rf / (tp_rf + fn_rf)
specificity_rf = tn_rf / (tn_rf + fp_rf)

print(f'Random Forest Sensitivity: {sensitivity_rf:.2f}')
print(f'Random Forest Specificity: {specificity_rf:.2f}')

# Display the confusion matrix
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=['Normal', 'Pneumonia'])
disp_rf.plot(cmap='Blues')
plt.title('Random Forest Confusion Matrix')
plt.show()
```

```
Random Forest Validation Accuracy: 93.87%
Random Forest Sensitivity: 0.97
Random Forest Specificity: 0.85
```


Random Forest Confusion Matrix
Normal
Pneumonia
True label
245
42
22
735
Normal
Pneumonia
Predicted label
700
600
500
400
300
200
100

In [17]:
```
# Precision vs Recall (Random Forest)

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_recall_curve, classification_report
import matplotlib.pyplot as plt
import numpy as np

# Random Forest model training
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Prediction of tags
y_pred_rf = rf_model.predict(X_test)

# Calculation of precision and recall
precision_rf, recall_rf, _ = precision_recall_curve(y_test, y_pred_rf)

# Full report
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))

# Draw a Precision vs Recall graph
plt.figure(figsize=(8, 6))
plt.plot(recall_rf, precision_rf, color='magenta')
plt.title('Precision vs Recall (Random Forest)', fontsize=14)
plt.xlabel('Recall', fontsize=12)
plt.ylabel('Precision', fontsize=12)
plt.grid(True)
plt.show()
```

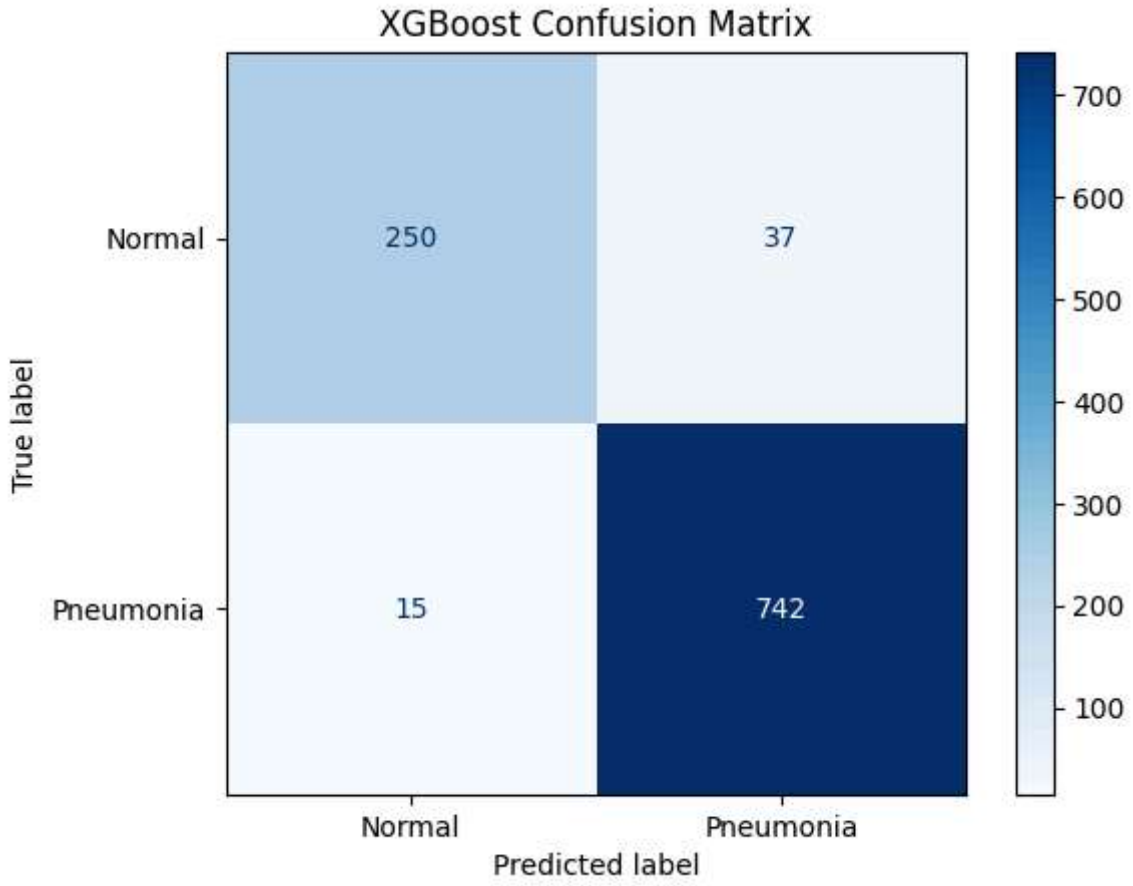
```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.85      0.88       287
           1       0.95      0.97      0.96       757

    accuracy                           0.94      1044
   macro avg       0.93      0.91      0.92      1044
weighted avg       0.94      0.94      0.94      1044
```


Precision vs Recall (Random Forest)
Precision
Recall
1.00
0.95
0.90
0.85
0.80
0.75
0.0
0.2
0.4
0.6
0.8
1.0

In [18]:
```
# XGBOOST
!pip install xgboost
import xgboost as xgb
from sklearn.metrics import accuracy_score, confusion_matrix

# Building the XGBoost model
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

# Model training
xgb_model.fit(X_train, y_train)

# Prediction on test data
y_pred_xgb = xgb_model.predict(X_test)

# Calculate accuracy
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print(f'XGBoost Validation Accuracy: {accuracy_xgb * 100:.2f}%')

# Calculation of confusion matrix
cm_xgb = confusion_matrix(y_test, y_pred_xgb)
tn_xgb, fp_xgb, fn_xgb, tp_xgb = cm_xgb.ravel()

# Calculation of sensitivity and specificity
sensitivity_xgb = tp_xgb / (tp_xgb + fn_xgb)
specificity_xgb = tn_xgb / (tn_xgb + fp_xgb)

print(f'XGBoost Sensitivity: {sensitivity_xgb:.2f}')
print(f'XGBoost Specificity: {specificity_xgb:.2f}')

# Display the confusion matrix
disp_xgb = ConfusionMatrixDisplay(confusion_matrix=cm_xgb, display_labels=['Normal', 'Pneumonia'])
disp_xgb.plot(cmap='Blues')
plt.title('XGBoost Confusion Matrix')
plt.show()
```

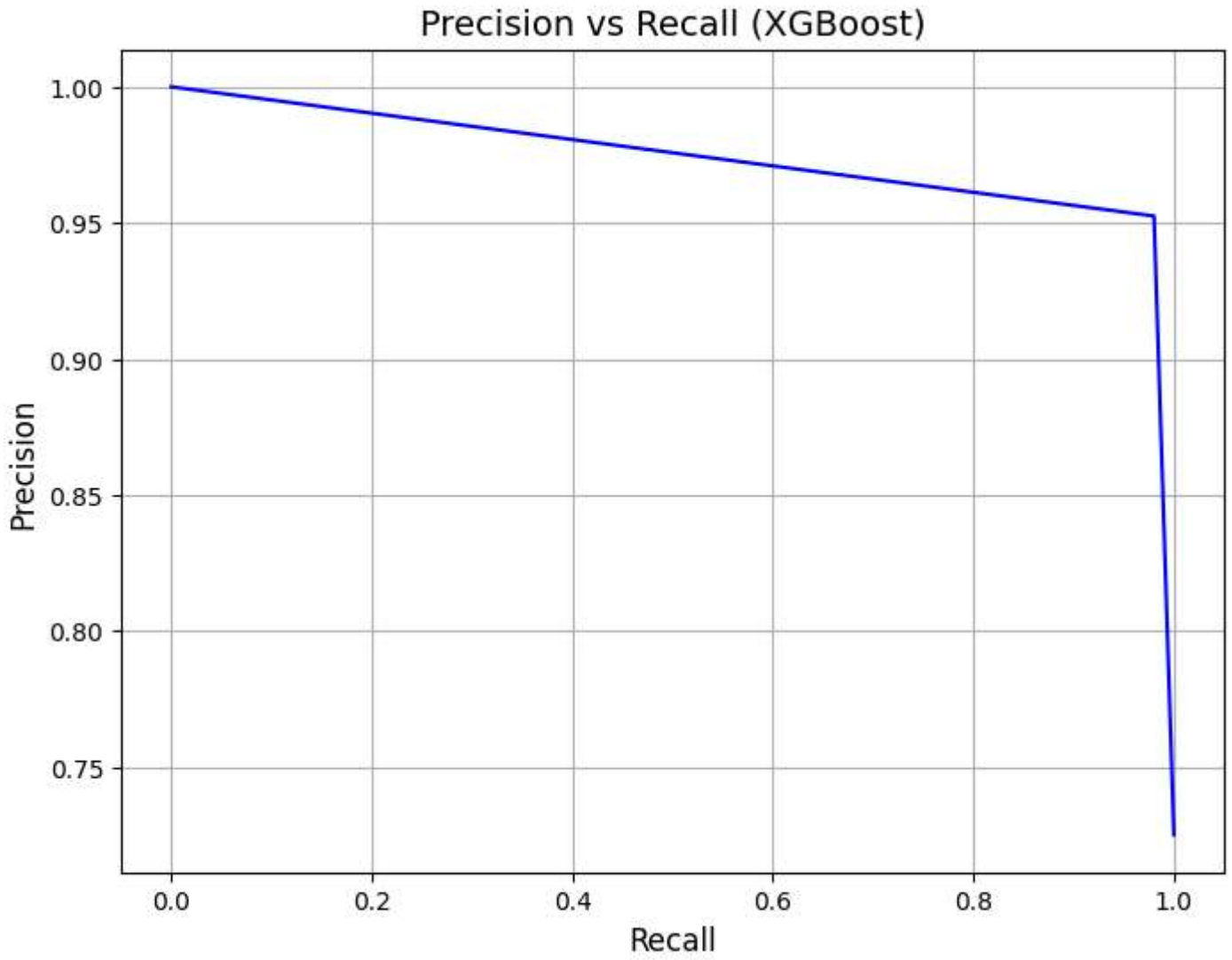
```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.2
3.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [17:18:46] WARNING: /workspace/src/l
earner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)

XGBoost Validation Accuracy: 95.02%
XGBoost Sensitivity: 0.98
XGBoost Specificity: 0.87
```



In [19]:
```
# Precision vs Recall (XGBOOST)

from xgboost import XGBClassifier
from sklearn.metrics import precision_recall_curve, classification_report
import matplotlib.pyplot as plt
import numpy as np

# XGBoost model training
xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(X_train, y_train)

# Prediction of tags
y_pred_xgb = xgb_model.predict(X_test)

# Calculation of precision and recall
precision_xgb, recall_xgb, _ = precision_recall_curve(y_test, y_pred_xgb)

# Full report
print("XGBoost Classification Report:")
print(classification_report(y_test, y_pred_xgb))

# Draw a Precision vs Recall graph
plt.figure(figsize=(8, 6))
plt.plot(recall_xgb, precision_xgb, color='blue')
plt.title('Precision vs Recall (XGBoost)', fontsize=14)
plt.xlabel('Recall', fontsize=12)
plt.ylabel('Precision', fontsize=12)
plt.grid(True)
plt.show()
```

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

           0       0.94      0.87      0.91       287
           1       0.95      0.98      0.97       757

    accuracy                           0.95      1044
   macro avg       0.95      0.93      0.94      1044
weighted avg       0.95      0.95      0.95      1044
```


Precision vs Recall (XGBoost)
Precision
Recall
1.00
0.95
0.90
0.85
0.80
0.75
0.0
0.2
0.4
0.6
0.8
1.0