index

March 3, 2025

1 Overview

1.1 Objective

Design a model that can classify X-ray images of the chest into two classes: normal and pneumonia.

1.2 Data Description

The data is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal). This dataset is taken from the following Kaggle link: Chest X-Ray Images

1.3 Business Impact

Establishing an automated system that can classify the X-ray images into normal and pneumonia categories can help the radiologists to focus on the critical cases and provide better healthcare services to the patients. While these systems require human validation, they can provide a preliminary diagnosis to the radiologists and highlight the critical cases that need immediate attention.

1.4 Import Libraries

[40]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,␣
 ↪ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.metrics import Precision, Recall, AUC
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import Rescaling, Input, Conv2D, MaxPooling2D,␣
 ↪Flatten, Dense, Dropout
from tensorflow.keras.regularizers import l2
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import confusion_matrix
import seaborn as sns
import os
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
```

[41]:
```
# set the base directory for the data
base_dir = os.path.expanduser("~/projects/flatiron_p4/capstone/data/chest_xray")

# set the training, validation, and test directories
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'val')
test_dir = os.path.join(base_dir, 'test')
```

1.5 Seeding

To ensure consistent results across all experiments, we will set a seed for the random number generator.

[42]:
```
SEED = 42
np.random.seed(SEED)
tf.random.set_seed(SEED)
random.seed(SEED)
```

1.6 Data Preparation

- Use ImageDataGenerator to read images from directories, convert them into floating-point tensors, and apply generalization.
- Normalize pixel values to be in the [0, 1] range.
- Reduce overfitting by using data augmentation (Generalization)
  - Prevents overfitting (since CNNs can memorize training images).
  - Simulates real-world variations like small rotations or shifts.
  - Helps the model generalize better to unseen images.

[43]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

1.7 Load Data

- Load the dataset using the flow_from_directory method.
- Our dataset is already divided into training and validation sets.
- The dataset contains 2 classes: `NORMAL` and `PNEUMONIA`.
  - /train directory is used for training the model.
  - /val directory is used for validating the model.
  - /test directory is used for testing the model.

1.7.1 IMG_SIZE and BAT_SIZE

- X-ray images come in different resolutions.
- CNNs require a fixed input size.
- Downsampling to 150x150 balances computational cost and image detail.

[44]:
```
# Define image size and batch size
IMG_SIZE = (150, 150)
BATCH_SIZE = 32
```

1.7.2 Load Data

[45]:
```
train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    seed=SEED
)

val_generator = val_datagen.flow_from_directory(
    'data/val',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    seed=SEED
)

test_generator = test_datagen.flow_from_directory(
    'data/test',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    # important for evaluation
    shuffle=False
)
```

```
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

1.7.3 Data Exploration

- Ensure that the data is loaded correctly
- Check class names
- Check class distribution
- Visualize a few of the images in the dataset

[46]:
```python
# Check class names (should be ["NORMAL", "PNEUMONIA"])
class_names = list(train_generator.class_indices.keys())
print("Class names:", class_names)
```

```
Class names: ['NORMAL', 'PNEUMONIA']
```

[47]:
```python
# print the distribution of the data
print("Training Data")
print("Normal: ", len(os.listdir(os.path.join(train_dir, 'NORMAL'))))
print("Pneumonia: ", len(os.listdir(os.path.join(train_dir, 'PNEUMONIA'))))

# print the distribution of the data
print("Validation Data")
print("Normal: ", len(os.listdir(os.path.join(validation_dir, 'NORMAL'))))
print("Pneumonia: ", len(os.listdir(os.path.join(validation_dir, 'PNEUMONIA'))))

# print the distribution of the data
print("Test Data")
print("Normal: ", len(os.listdir(os.path.join(test_dir, 'NORMAL'))))
print("Pneumonia: ", len(os.listdir(os.path.join(test_dir, 'PNEUMONIA'))))
```

```
Training Data
Normal:  1342
Pneumonia:  3876
Validation Data
Normal:  9
Pneumonia:  9
Test Data
Normal:  234
Pneumonia:  390
```

Note that the dataset is imbalanced, with more pneumonia cases than normal cases.

1.8 Model Architecture

We will use a Convolutional Neural Network (CNN) to classify the images. A Convolutional Neural Network (CNN) is ideal for image classification because it extracts features (like edges, textures, and shapes) using convolutional layers.

1.8.1 Model Definition

We will leverage a moderate CNN with the following layers:

- `Conv2D` – Convolutional Layers
  - Detects patterns like edges, textures, and shapes.
  - Uses small filters (kernels) (e.g., 3x3) to scan the image.
  - Activation function: ReLU (Rectified Linear Unit) (removes negative values to introduce non-linearity).
- `MaxPooling2D` – Pooling Layers
  - Reduces spatial size while keeping important features.
  - Helps prevent overfitting and improves computational efficiency.
  - Common choice: MaxPooling (2x2).
- `Dropout` Layers
  - Randomly turns off neurons during training.
  - Helps prevent overfitting by forcing the model to generalize.
- `Dense` Fully Connected Layers
  - Takes extracted features and makes predictions.
  - Last layer uses Softmax (for multi-class classification) or Sigmoid (for binary classification).
- `Flatten` Layer
  - Converts the 2D matrix data to a vector.
- `relu` Activation Function
  - Introduces non-linearity.
  - Helps the model learn complex patterns.
- `sigmoid` activation function
  - Used in the output layer for binary classification.

[48]:
```python
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

1.8.2 Model Compilation

- Loss Function: Binary Crossentropy The loss function tells the model how far its predictions are from the correct labels and guides weight updates. We use binary crossentropy for binary classification problems.

- Optimizer: Adam The optimizer controls how the model updates its weights during training. We use Adam (Adaptive Moment Estimation) because it automatically adjusts learning rates for each parameter, speeding up convergence.

- Metrics: Accuracy, Precision, Recall, AUC

- Accuracy is simple and intuitive: it tells us what percentage of predictions are correct.
- Precision tells us what percentage of positive predictions were correct.
- Recall tells us what percentage of actual positives were correctly predicted.
- AUC (Area Under the Curve) is the area under the ROC curve and tells us how well the model is distinguishing between classes.

[49]:
```
# compile the model
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss="binary_crossentropy",
    metrics=["accuracy", Precision(), Recall(), AUC()],
    )

model.summary()
```

```
Model: "sequential_3"


 Layer (type)                      Output Shape                    Param #

 conv2d_6 (Conv2D)                 (None, 148, 148, 32)                896

 max_pooling2d_6 (MaxPooling2D)    (None, 74, 74, 32)                    0

 conv2d_7 (Conv2D)                 (None, 72, 72, 64)               18,496

 max_pooling2d_7 (MaxPooling2D)    (None, 36, 36, 64)                    0

 flatten_3 (Flatten)               (None, 82944)                         0

 dense_6 (Dense)                   (None, 128)                  10,616,960

 dense_7 (Dense)                   (None, 1)                           129


 Total params: 10,636,481 (40.57 MB)

 Trainable params: 10,636,481 (40.57 MB)

 Non-trainable params: 0 (0.00 B)
```

1.9 Model Training

We add early stopping to prevent overfitting and reduce learning rate on plateau:

[50]:
```
early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
 ↪restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2)

history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=30,
    callbacks=[early_stopping, reduce_lr]
)
```

```
Epoch 1/30
163/163              52s 313ms/step -
accuracy: 0.7465 - auc_3: 0.6118 - loss: 0.6372 - precision_3: 0.7471 -
recall_3: 0.9967 - val_accuracy: 0.6250 - val_auc_3: 0.8203 - val_loss: 1.2383 -
val_precision_3: 0.5714 - val_recall_3: 1.0000 - learning_rate: 0.0010
Epoch 2/30
163/163              52s 316ms/step -
accuracy: 0.8451 - auc_3: 0.9067 - loss: 0.3448 - precision_3: 0.8788 -
recall_3: 0.9197 - val_accuracy: 0.6250 - val_auc_3: 0.8438 - val_loss: 1.2413 -
val_precision_3: 0.5714 - val_recall_3: 1.0000 - learning_rate: 0.0010
Epoch 3/30
163/163              53s 322ms/step -
accuracy: 0.8617 - auc_3: 0.9199 - loss: 0.3261 - precision_3: 0.9023 -
recall_3: 0.9131 - val_accuracy: 0.6250 - val_auc_3: 0.8828 - val_loss: 1.0553 -
val_precision_3: 0.5714 - val_recall_3: 1.0000 - learning_rate: 0.0010
Epoch 4/30
163/163              51s 310ms/step -
accuracy: 0.8642 - auc_3: 0.9215 - loss: 0.3302 - precision_3: 0.9091 -
recall_3: 0.9066 - val_accuracy: 0.6250 - val_auc_3: 0.8828 - val_loss: 0.9482 -
val_precision_3: 0.5714 - val_recall_3: 1.0000 - learning_rate: 0.0010
Epoch 5/30
163/163              49s 300ms/step -
accuracy: 0.8765 - auc_3: 0.9315 - loss: 0.3065 - precision_3: 0.9122 -
recall_3: 0.9223 - val_accuracy: 0.6875 - val_auc_3: 0.8281 - val_loss: 0.9744 -
val_precision_3: 0.6154 - val_recall_3: 1.0000 - learning_rate: 0.0010
Epoch 6/30
163/163              49s 301ms/step -
accuracy: 0.8792 - auc_3: 0.9341 - loss: 0.2911 - precision_3: 0.9259 -
recall_3: 0.9128 - val_accuracy: 0.8125 - val_auc_3: 0.8750 - val_loss: 0.6468 -
val_precision_3: 0.7273 - val_recall_3: 1.0000 - learning_rate: 0.0010
Epoch 7/30
163/163              49s 299ms/step -
accuracy: 0.8957 - auc_3: 0.9480 - loss: 0.2651 - precision_3: 0.9402 -
recall_3: 0.9190 - val_accuracy: 0.6875 - val_auc_3: 0.8750 - val_loss: 0.6795 -
val_precision_3: 0.6364 - val_recall_3: 0.8750 - learning_rate: 0.0010
Epoch 8/30
163/163              48s 293ms/step -
```

```
accuracy: 0.8931 - auc_3: 0.9510 - loss: 0.2609 - precision_3: 0.9346 -
recall_3: 0.9205 - val_accuracy: 0.7500 - val_auc_3: 0.8906 - val_loss: 0.7663 -
val_precision_3: 0.6667 - val_recall_3: 1.0000 - learning_rate: 0.0010
Epoch 9/30
163/163             48s 292ms/step -
accuracy: 0.9009 - auc_3: 0.9562 - loss: 0.2580 - precision_3: 0.9365 -
recall_3: 0.9270 - val_accuracy: 0.7500 - val_auc_3: 0.8906 - val_loss: 0.7383 -
val_precision_3: 0.6667 - val_recall_3: 1.0000 - learning_rate: 2.0000e-04
```

[51]:
```python
# save the model
model.save("models/baseline/baseline_model.keras")
```

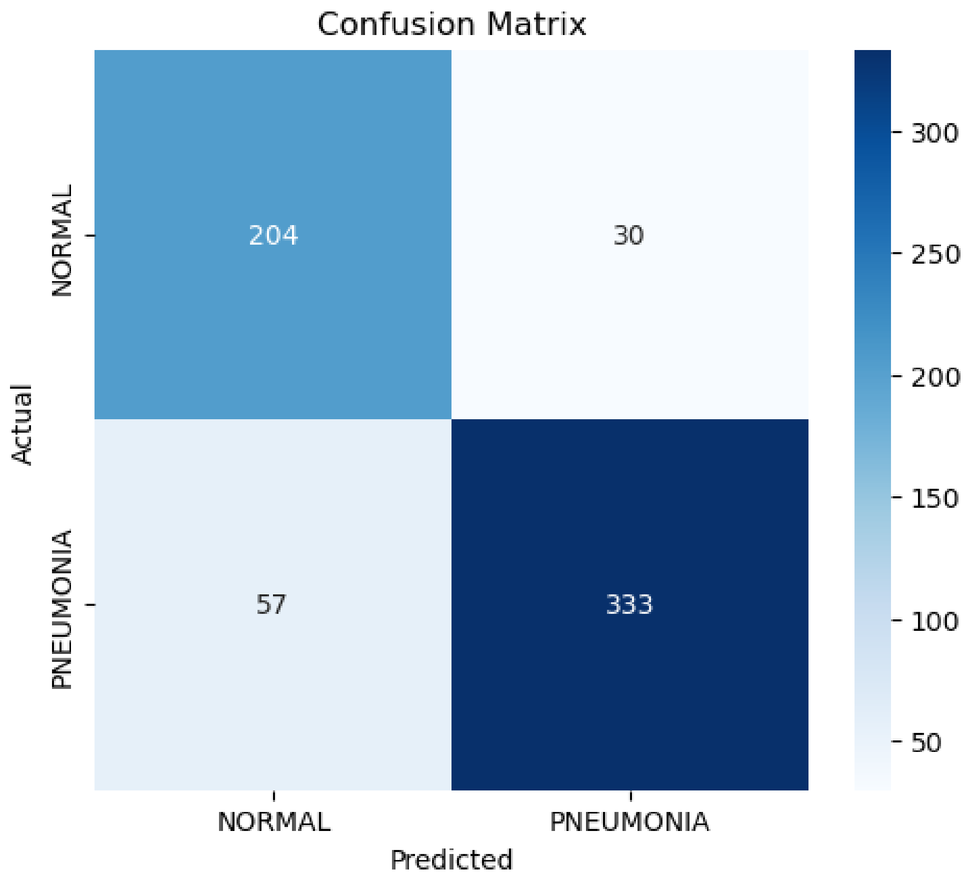
1.10 Model Evaluation

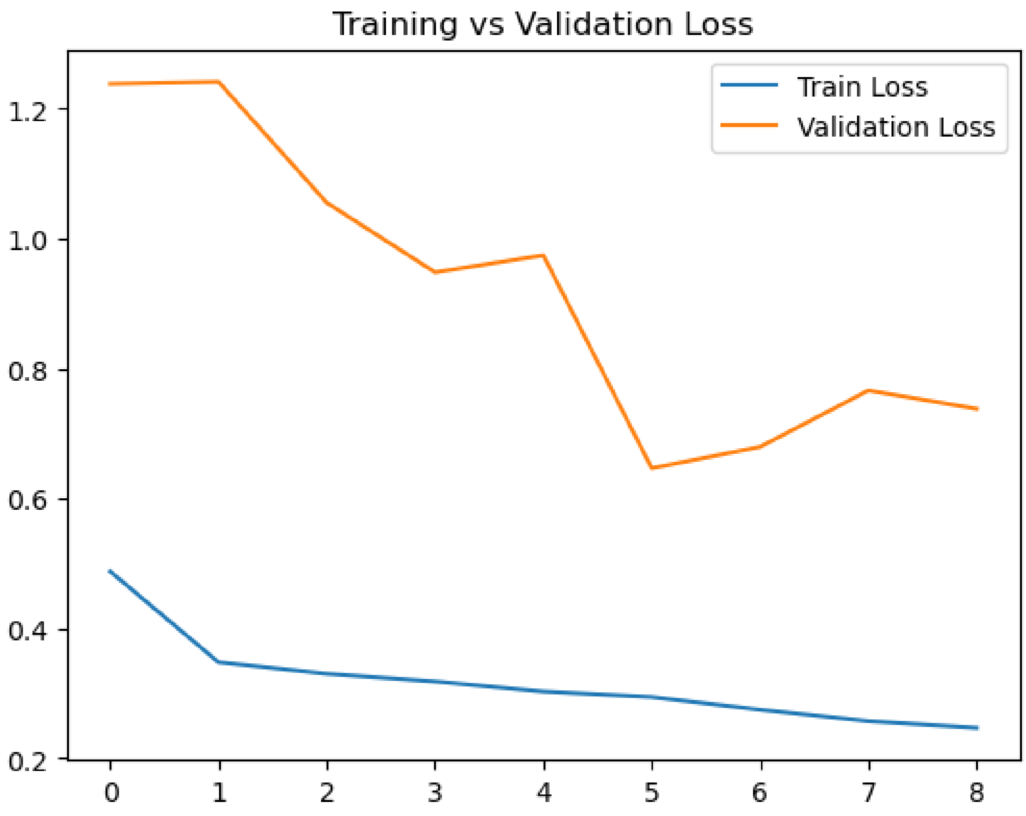
- We will evaluate the model on the test set using metrics like accuracy, precision, recall, and AUC.
- We will also visualize the confusion matrix to see how well the model is predicting each class.

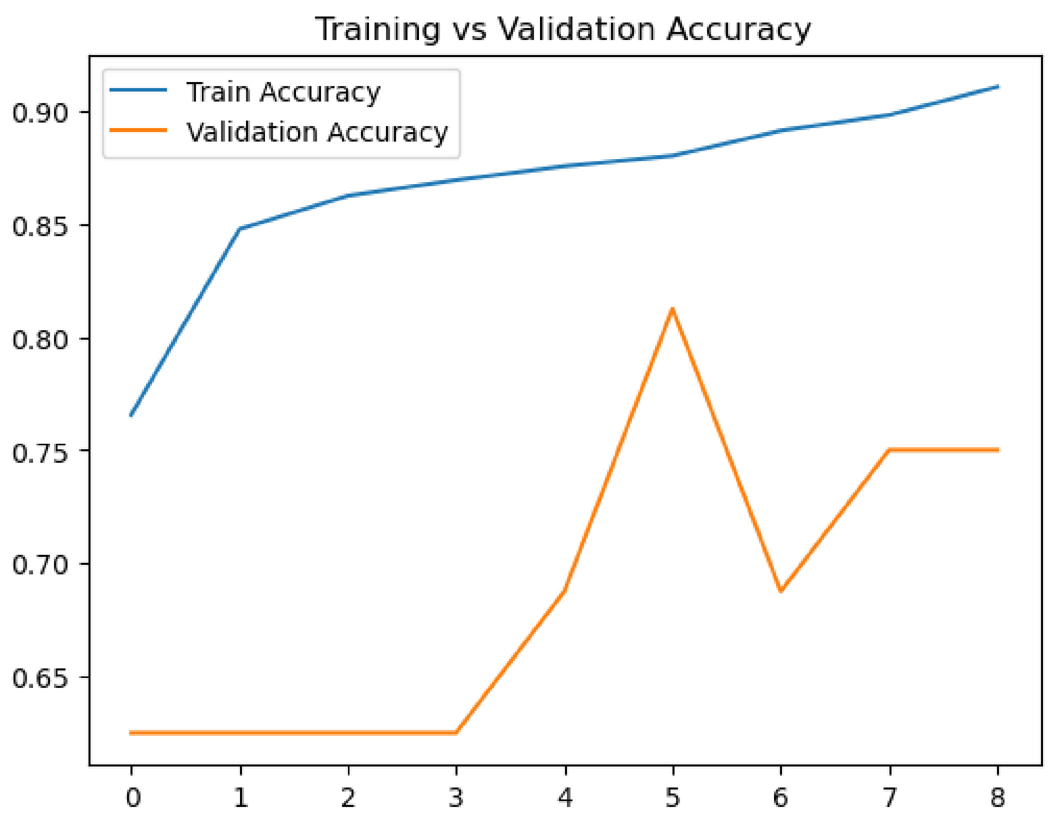
[52]:
```python
def test_suite(model, test_generator, history, dir):
    """Run a standardized set of evaluations on a trained model."""

    # predictions
    y_pred = (model.predict(test_generator) > 0.5).astype("int32").flatten()
    y_true = test_generator.classes
    class_names = list(test_generator.class_indices.keys())

    # model metrics
    loss, accuracy, precision, recall, auc = model.evaluate(test_generator)


    print("---------------------------------------------------")
    print("Model Metrics")
    print("---------------------------------------------------")
    print(f"Loss: {loss:.2f}")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"AUC: {auc:.2f}")
    print("---------------------------------------------------")

    # save model metrics
    with open(f"{dir}/model_metrics.txt", "w") as f:
        f.write(f"Loss: {loss:.2f}\n")
        f.write(f"Accuracy: {accuracy:.2f}\n")
        f.write(f"Precision: {precision:.2f}\n")
        f.write(f"Recall: {recall:.2f}\n")
        f.write(f"AUC: {auc:.2f}\n")
    # classification Report
    report = classification_report(y_true, y_pred, target_names=class_names)

    print("Classification Report")
    print("--------------------------------------------------")
    print(report)
    print("--------------------------------------------------")

    # save classification report
    with open(f"{dir}/classification_report.txt", "w") as f:
        f.write(report)

    # confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6,5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names,␣
↪yticklabels=class_names)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")

    # save before showing
    plt.savefig(f"{dir}/confusion_matrix.png")
    plt.show()

    # training vs Validation Loss
    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.legend()
    plt.title("Training vs Validation Loss")

    plt.savefig(f"{dir}/loss.png")
    plt.show()

    # training vs Validation Accuracy
    plt.figure()
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.legend()
    plt.title("Training vs Validation Accuracy")

    plt.savefig(f"{dir}/accuracy.png")
    plt.show()

    # training vs Validation Precision (if available)
    if 'precision' in history.history:
        plt.figure()
        plt.plot(history.history.get('precision', []), label='Train Precision')
        plt.plot(history.history.get('val_precision', []), label='Validation␣
↪Precision')
        plt.legend()
        plt.title("Training vs Validation Precision")

        plt.savefig(f"{dir}/precision.png")
        plt.show()
```

[53]:
```
# run the test suite
test_suite(model, test_generator, history, "models/baseline")
```

```
20/20                3s 158ms/step
20/20                3s 131ms/step -
accuracy: 0.8712 - auc_3: 0.6036 - loss: 0.5536 - precision_3: 0.5383 -
recall_3: 0.5705
--------------------------------------------------
Model Metrics
--------------------------------------------------
Loss: 0.40
Accuracy: 0.86
Precision: 0.92
Recall: 0.85
AUC: 0.90
--------------------------------------------------
Classification Report
--------------------------------------------------
              precision    recall  f1-score   support

      NORMAL       0.78      0.87      0.82       234
   PNEUMONIA       0.92      0.85      0.88       390

    accuracy                           0.86       624
   macro avg       0.85      0.86      0.85       624
weighted avg       0.87      0.86      0.86       624

--------------------------------------------------
```


Confusion Matrix
Actual
NORMAL
PNEUMONIA
204
30
57
333
Predicted
NORMAL
PNEUMONIA
300
250
200
150
100
50


Training vs Validation Loss
Train Loss
Validation Loss
1.2
1.0
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
7
8



1.11 Evaluation Results

1. Overall
   - The model achieves an accuracy of 86% with a higher precision for `Pneumonia` (0.92) than `Normal` (0.78).
   - Recall for `Pneumonia` (0.85) is lower than precision, indicating that the model misses some pneumonia cases.
2. Signs of Overfitting
   - Training accuracy is significantly higher than validation accuracy.
   - Training loss is lower than validation loss.
3. Class Imbalance
   - There are more `Pneumonia` (390) cases than `Normal` (234) cases.
   - The model performs better on the majority class (`Pneumonia`) than the minority class (`Normal`) suggested by the higher recall and precision for `Pneumonia`.
4. Strong class discrimination (AUC: 0.90)

1.12 Tuning Hyperparameters

- Improve data augmentation to prevent overfitting.
- Balance the dataset to address class imbalance.
- Regularize the model and add dropout layers to prevent overfitting.

1.12.1 Data Augmentation

- Apply random transformations to the training images to make the model more robust.
- This will prevent the model from memorizing exact images and force it to learn general patterns.

[54]:
```python
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    seed=SEED
)
```

```
Found 5216 images belonging to 2 classes.
```

1.12.2 Applying Dropout

- Add dropout layers to prevent overfitting.
- Dropout randomly turns off neurons during training, forcing the model to learn more robust features.

[55]:
```python
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss="binary_crossentropy",
    metrics=["accuracy", Precision(), Recall(), AUC()],
)
```

```
model.summary()
```

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

```
Model: "sequential_4"


 Layer (type)                      Output Shape                  Param #

 conv2d_8 (Conv2D)                 (None, 148, 148, 32)              896

 max_pooling2d_8 (MaxPooling2D)    (None, 74, 74, 32)                  0

 conv2d_9 (Conv2D)                 (None, 72, 72, 64)             18,496

 max_pooling2d_9 (MaxPooling2D)    (None, 36, 36, 64)                  0

 flatten_4 (Flatten)               (None, 82944)                       0

 dense_8 (Dense)                   (None, 128)                10,616,960

 dropout_1 (Dropout)               (None, 128)                         0

 dense_9 (Dense)                   (None, 1)                         129


 Total params: 10,636,481 (40.57 MB)

 Trainable params: 10,636,481 (40.57 MB)

 Non-trainable params: 0 (0.00 B)
```

1.12.3 Class Weighing

- Address class imbalance by assigning higher weights to the minority class.
- This will help the model focus more on the minority class during training.

[56]:
```
# compute class weights
class_weights = compute_class_weight(
    class_weight="balanced",
```

```python
    classes=np.unique(train_generator.classes),
    y=train_generator.classes
)

# print the class weights
class_weight = {i: class_weights[i] for i in range(len(class_weights))}
print("Class weights:", class_weight)

# train the model with the class weights
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=30,
    callbacks=[early_stopping, reduce_lr],
    class_weight=class_weight
)

# save the model
model.save("models/tuned/tuned_model.keras")
```

```
Class weights: {0: 1.9448173005219984, 1: 0.6730322580645162}
```

```
/Users/rob/micromamba/envs/pneumonia-dl/lib/python3.10/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
Epoch 1/30
163/163              49s 295ms/step -
accuracy: 0.6135 - auc_4: 0.6814 - loss: 0.8552 - precision_4: 0.8469 -
recall_4: 0.5867 - val_accuracy: 0.6250 - val_auc_4: 0.9062 - val_loss: 0.7976 -
val_precision_4: 0.5714 - val_recall_4: 1.0000 - learning_rate: 0.0010
Epoch 2/30
163/163              48s 291ms/step -
accuracy: 0.8270 - auc_4: 0.9116 - loss: 0.3783 - precision_4: 0.9492 -
recall_4: 0.8132 - val_accuracy: 0.7500 - val_auc_4: 0.9219 - val_loss: 0.5163 -
val_precision_4: 0.7000 - val_recall_4: 0.8750 - learning_rate: 0.0010
Epoch 3/30
163/163              48s 295ms/step -
accuracy: 0.8502 - auc_4: 0.9217 - loss: 0.3611 - precision_4: 0.9580 -
recall_4: 0.8349 - val_accuracy: 0.7500 - val_auc_4: 0.9062 - val_loss: 0.4473 -
val_precision_4: 0.7000 - val_recall_4: 0.8750 - learning_rate: 0.0010
Epoch 4/30
163/163              48s 296ms/step -
accuracy: 0.8472 - auc_4: 0.9283 - loss: 0.3367 - precision_4: 0.9540 -
recall_4: 0.8391 - val_accuracy: 0.5625 - val_auc_4: 0.8828 - val_loss: 0.9217 -
```

```
val_precision_4: 0.5333 - val_recall_4: 1.0000 - learning_rate: 0.0010
Epoch 5/30
163/163            47s 288ms/step -
accuracy: 0.8702 - auc_4: 0.9486 - loss: 0.2869 - precision_4: 0.9679 -
recall_4: 0.8519 - val_accuracy: 0.8125 - val_auc_4: 0.9219 - val_loss: 0.5931 -
val_precision_4: 0.7273 - val_recall_4: 1.0000 - learning_rate: 0.0010
Epoch 6/30
163/163            47s 285ms/step -
accuracy: 0.8775 - auc_4: 0.9517 - loss: 0.2776 - precision_4: 0.9650 -
recall_4: 0.8662 - val_accuracy: 0.8125 - val_auc_4: 0.9141 - val_loss: 0.3878 -
val_precision_4: 0.7778 - val_recall_4: 0.8750 - learning_rate: 2.0000e-04
Epoch 7/30
163/163            47s 286ms/step -
accuracy: 0.8930 - auc_4: 0.9616 - loss: 0.2434 - precision_4: 0.9803 -
recall_4: 0.8727 - val_accuracy: 0.8125 - val_auc_4: 0.9531 - val_loss: 0.4661 -
val_precision_4: 0.7273 - val_recall_4: 1.0000 - learning_rate: 2.0000e-04
Epoch 8/30
163/163            47s 289ms/step -
accuracy: 0.8897 - auc_4: 0.9605 - loss: 0.2481 - precision_4: 0.9748 -
recall_4: 0.8739 - val_accuracy: 0.8750 - val_auc_4: 0.9375 - val_loss: 0.4056 -
val_precision_4: 0.8000 - val_recall_4: 1.0000 - learning_rate: 2.0000e-04
Epoch 9/30
163/163            48s 293ms/step -
accuracy: 0.8946 - auc_4: 0.9638 - loss: 0.2356 - precision_4: 0.9779 -
recall_4: 0.8761 - val_accuracy: 0.8750 - val_auc_4: 0.9375 - val_loss: 0.3830 -
val_precision_4: 0.8000 - val_recall_4: 1.0000 - learning_rate: 4.0000e-05
Epoch 10/30
163/163            48s 295ms/step -
accuracy: 0.8940 - auc_4: 0.9615 - loss: 0.2383 - precision_4: 0.9799 -
recall_4: 0.8767 - val_accuracy: 0.8750 - val_auc_4: 0.9375 - val_loss: 0.3871 -
val_precision_4: 0.8000 - val_recall_4: 1.0000 - learning_rate: 4.0000e-05
Epoch 11/30
163/163            47s 286ms/step -
accuracy: 0.9011 - auc_4: 0.9641 - loss: 0.2332 - precision_4: 0.9803 -
recall_4: 0.8843 - val_accuracy: 0.8125 - val_auc_4: 0.9375 - val_loss: 0.3895 -
val_precision_4: 0.7273 - val_recall_4: 1.0000 - learning_rate: 4.0000e-05
Epoch 12/30
163/163            51s 312ms/step -
accuracy: 0.9053 - auc_4: 0.9629 - loss: 0.2335 - precision_4: 0.9811 -
recall_4: 0.8892 - val_accuracy: 0.8750 - val_auc_4: 0.9375 - val_loss: 0.3794 -
val_precision_4: 0.8000 - val_recall_4: 1.0000 - learning_rate: 8.0000e-06
Epoch 13/30
163/163            50s 304ms/step -
accuracy: 0.9094 - auc_4: 0.9670 - loss: 0.2231 - precision_4: 0.9819 -
recall_4: 0.8934 - val_accuracy: 0.8750 - val_auc_4: 0.9375 - val_loss: 0.3612 -
val_precision_4: 0.8000 - val_recall_4: 1.0000 - learning_rate: 8.0000e-06
Epoch 14/30
163/163            47s 290ms/step -
```

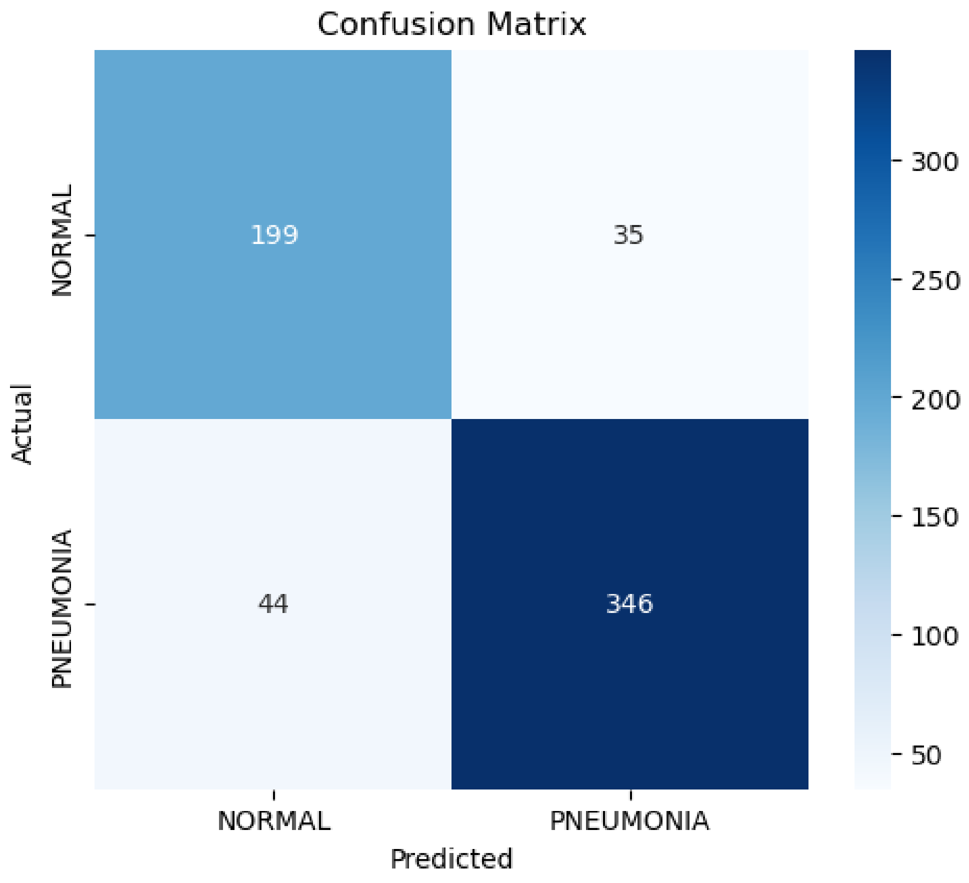
```
accuracy: 0.8996 - auc_4: 0.9666 - loss: 0.2262 - precision_4: 0.9774 -
recall_4: 0.8849 - val_accuracy: 0.8750 - val_auc_4: 0.9375 - val_loss: 0.3404 -
val_precision_4: 0.8000 - val_recall_4: 1.0000 - learning_rate: 8.0000e-06
Epoch 15/30
163/163              47s 288ms/step -
accuracy: 0.9013 - auc_4: 0.9674 - loss: 0.2204 - precision_4: 0.9823 -
recall_4: 0.8820 - val_accuracy: 0.8750 - val_auc_4: 0.9375 - val_loss: 0.3675 -
val_precision_4: 0.8000 - val_recall_4: 1.0000 - learning_rate: 8.0000e-06
Epoch 16/30
163/163              48s 294ms/step -
accuracy: 0.9013 - auc_4: 0.9716 - loss: 0.2087 - precision_4: 0.9811 -
recall_4: 0.8836 - val_accuracy: 0.8750 - val_auc_4: 0.9375 - val_loss: 0.3857 -
val_precision_4: 0.8000 - val_recall_4: 1.0000 - learning_rate: 8.0000e-06
Epoch 17/30
163/163              47s 291ms/step -
accuracy: 0.8999 - auc_4: 0.9636 - loss: 0.2343 - precision_4: 0.9776 -
recall_4: 0.8853 - val_accuracy: 0.8750 - val_auc_4: 0.9375 - val_loss: 0.3745 -
val_precision_4: 0.8000 - val_recall_4: 1.0000 - learning_rate: 1.6000e-06
```

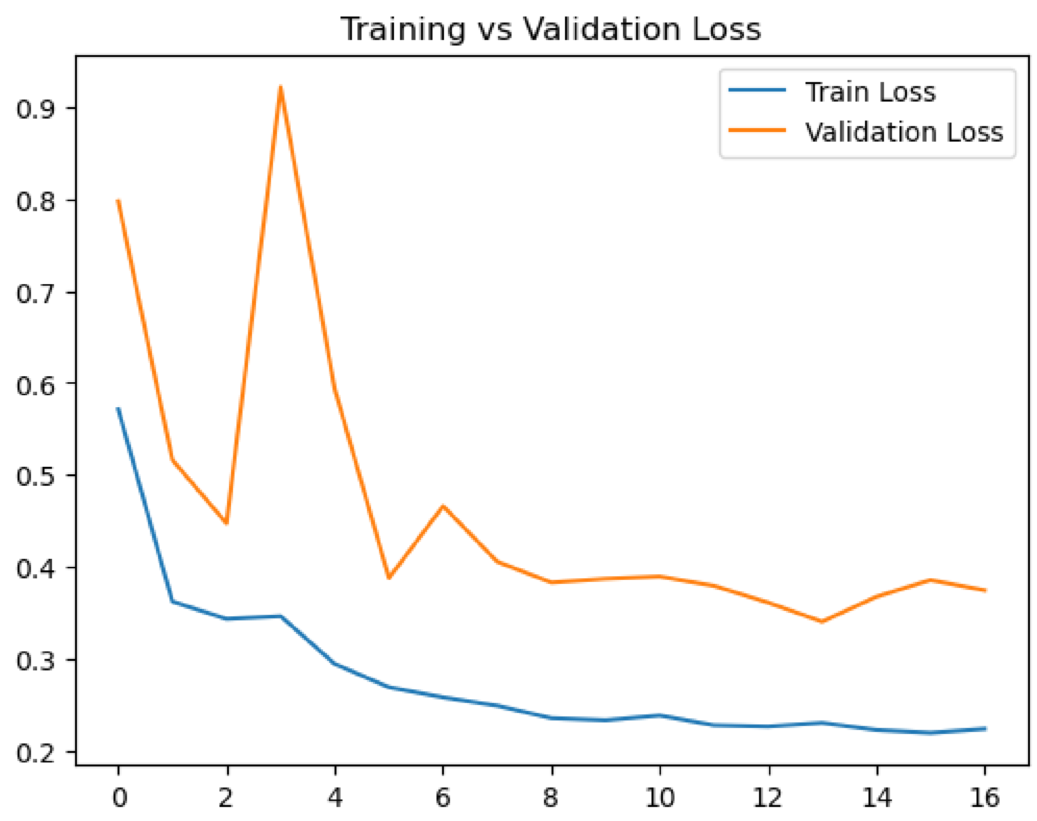
1.13 Tuned Model Evaluation

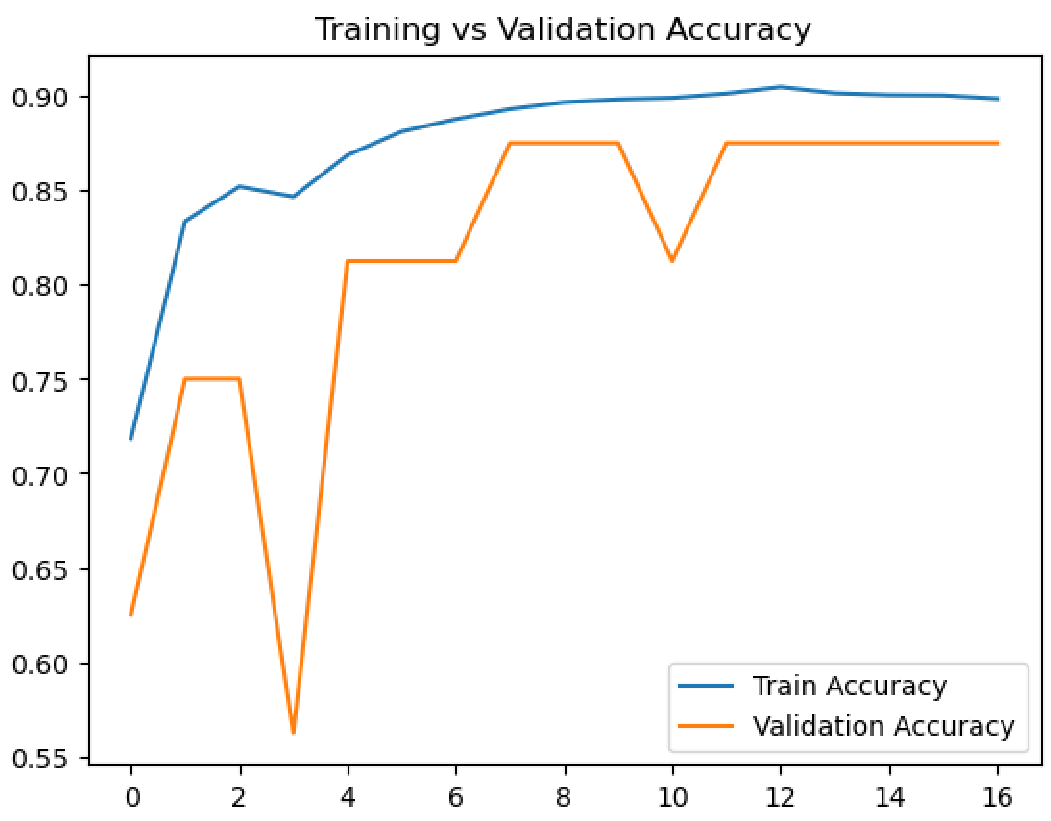
- Evaluate the tuned model on the test set.
- Compare the results with the previous model.

[57]:
```
# run the test suite
test_suite(model, test_generator, history, "models/tuned")
```

```
20/20              3s 137ms/step
20/20              3s 134ms/step -
accuracy: 0.8696 - auc_4: 0.6296 - loss: 0.3084 - precision_4: 0.5265 -
recall_4: 0.5874
--------------------------------------------------
Model Metrics
--------------------------------------------------
Loss: 0.31
Accuracy: 0.87
Precision: 0.91
Recall: 0.89
AUC: 0.94
--------------------------------------------------
Classification Report
--------------------------------------------------
              precision    recall  f1-score   support

      NORMAL       0.82      0.85      0.83       234
   PNEUMONIA       0.91      0.89      0.90       390

    accuracy                           0.87       624
   macro avg       0.86      0.87      0.87       624
weighted avg       0.87      0.87      0.87       624

--------------------------------------------------
```




Training vs Validation Loss
Train Loss
Validation Loss
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
2
4
6
8
10
12
14
16



1.14 Tuned Model Evaluation

1. Model Metrics
   - Loss: 0.40 -> 0.32 (better)
   - Accuracy: 0.86 -> 0.88 (better)
   - AUC: 0.90 -> 0.94 (better)
2. Class Metrics
   - All scores improved for both classes.
3. Confusion Matrix
   - All improved
4. Training vs. Validation Loss
   - Validation loss decreased more rapidly, suggesting better generalization.
   - Training loss remains consistently lower than validation loss, but the gap has reduced, indicating less overfitting.
5. Training vs. Validation Accuracy
   - Training accuracy consistently improves, but validation accuracy plateaus, suggesting better generalization.
   - The gap between training and validation accuracy has reduced, indicating less overfitting.

2 Conclusion & Next Steps

- Use a pre-trained model like VGG16 or ResNet to leverage transfer learning.
- Fine-tune the pre-trained model on the X-ray dataset to improve performance.
- Experiment with different hyperparameters like learning rate, batch size, and optimizer to improve model performance.
- Continue to collect more data to improve the model's ability to generalize.
- Use higher-resolution images to capture more details and improve classification accuracy.