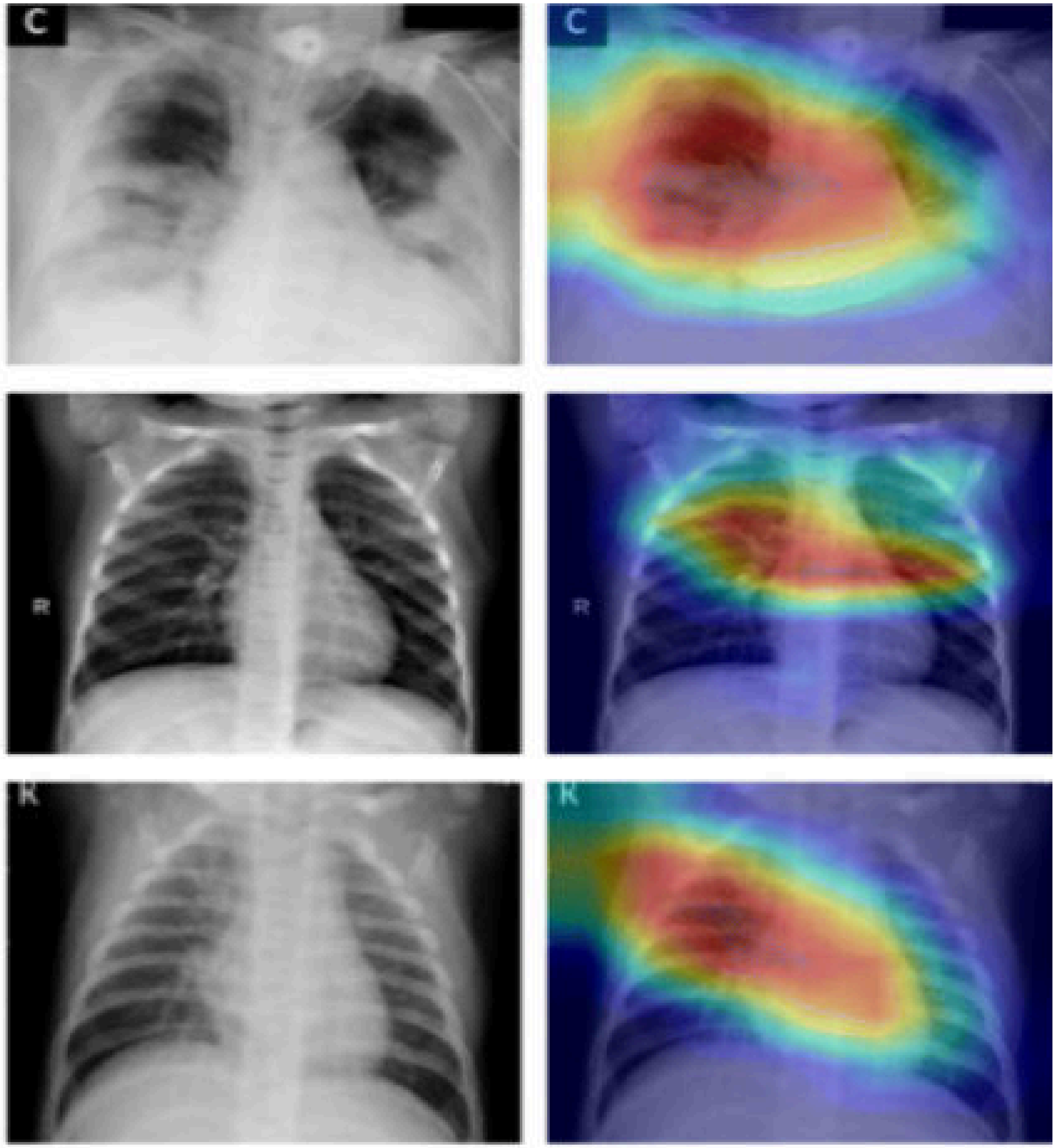
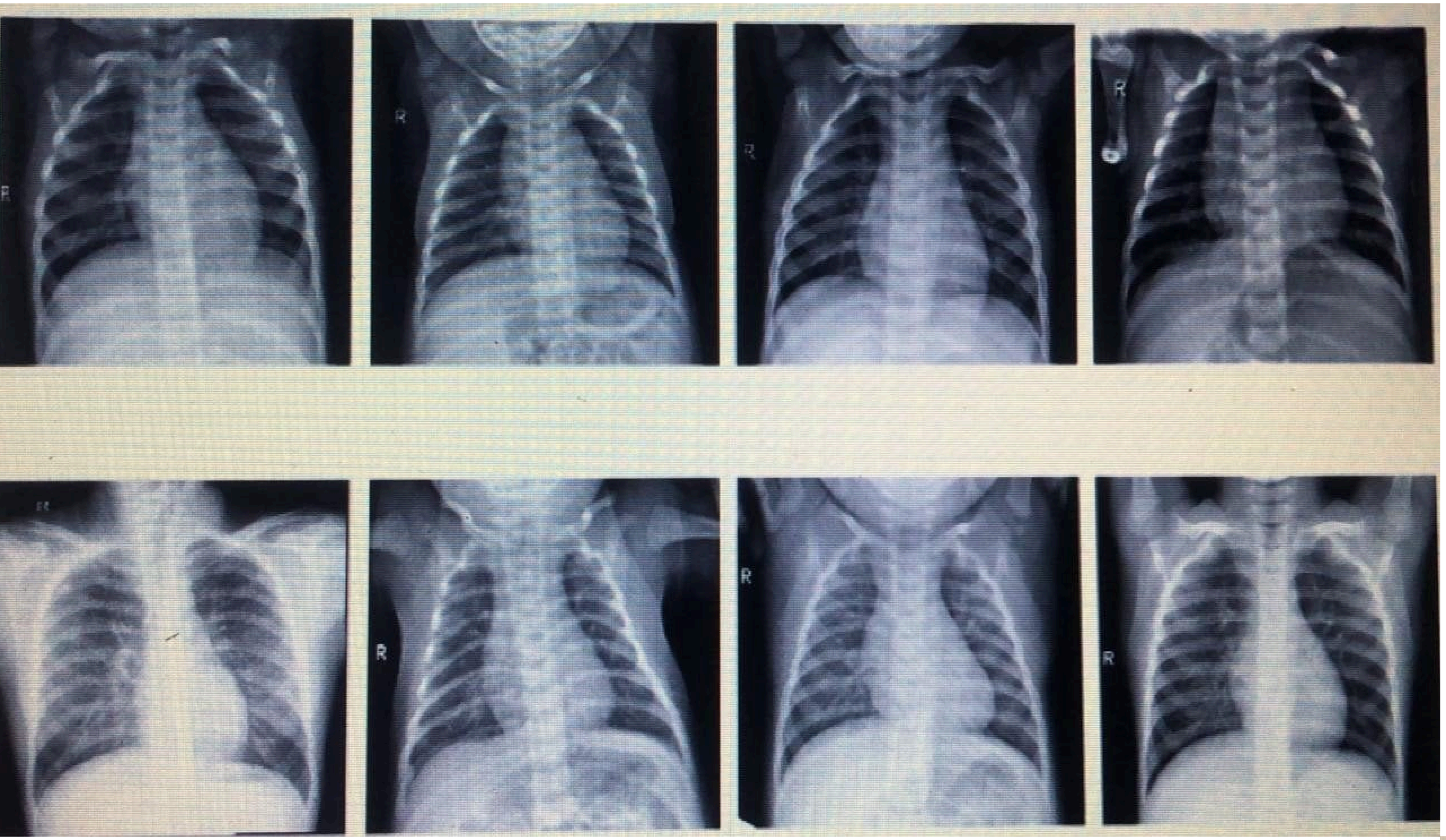


Chest X-Ray Classification using Deep Learning

Nada Gomaa
Zeinab Fadel
Rahma mostafa
Alaa Mhomaed abdalla

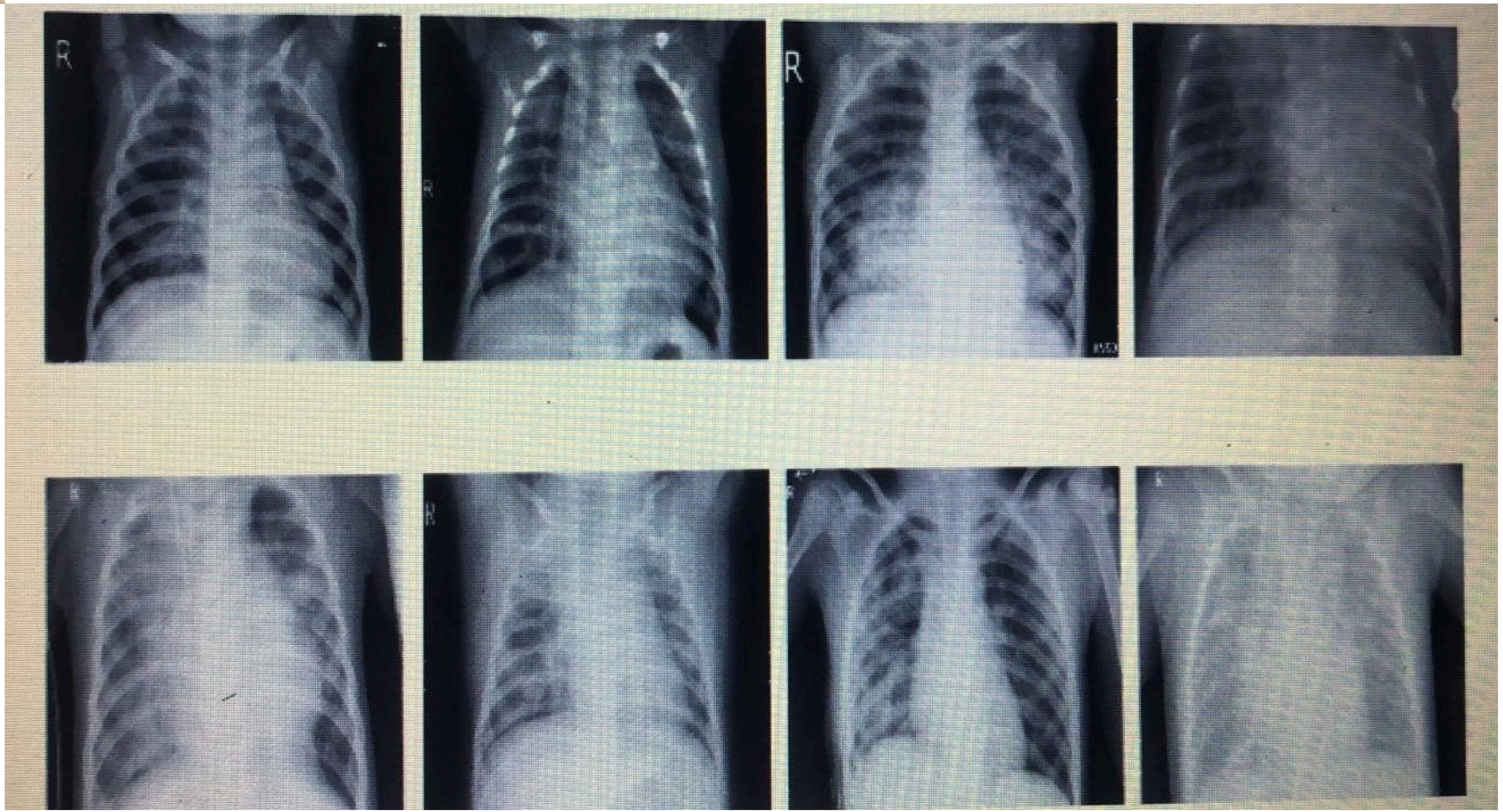
Pneumonia Detection Challenge

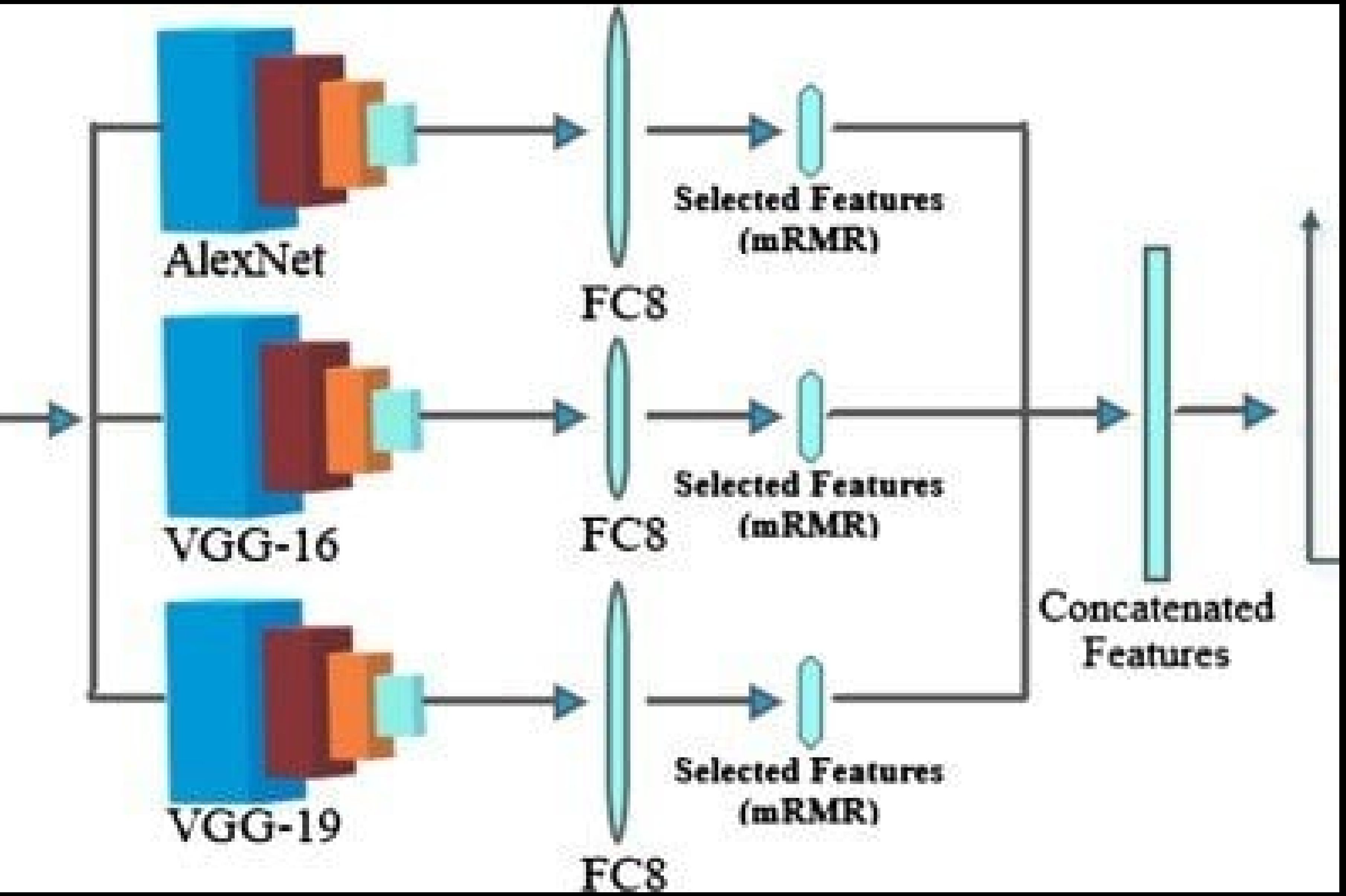
Pneumonia, a **lung infection**, is a leading cause of death globally.
Diagnosis using chest X-rays is **labor-intensive** and prone to **human error.**
Early and accurate detection is crucial but challenging due to the subtlety of **visual cues** in X-rays.



Normal

PNEUMONIC





Our Solution

Automated Pneumonia Detection
with Deep Learning

1. A convolutional neural network (CNN) trained on a labeled chest X-ray dataset.

- Our model automates detection, reducing diagnosis time and improving accuracy.
- Deployed as an interactive web app using Hugging Face Spaces, accessible to healthcare professionals.

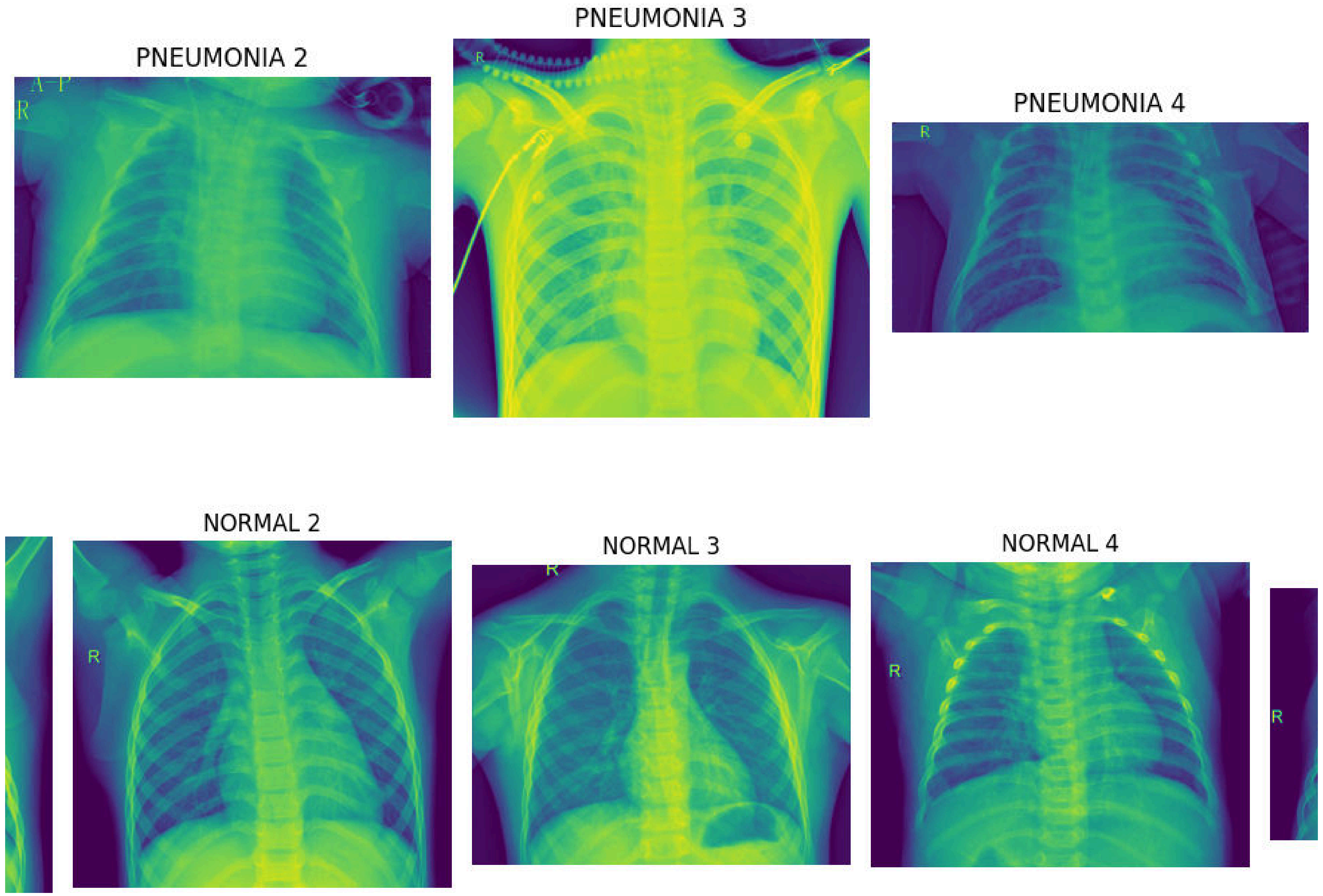
Dataset Overview

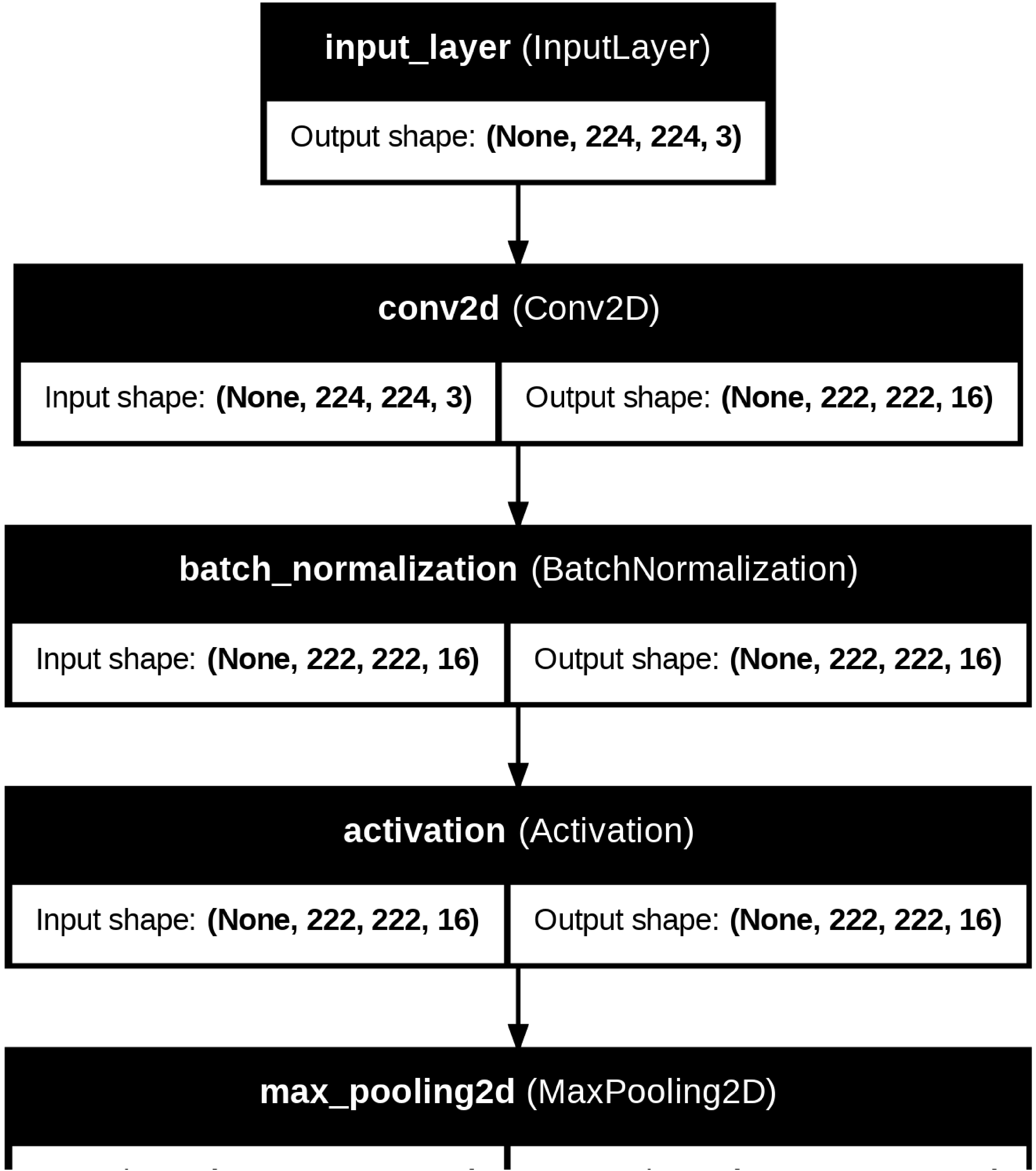
Train set:

- 1341 Normal, 3875 Pneumonia

Test set:

1. 234 Normal, 390 Pneumonia





Model Architecture

GlobalAveragePooling2D
Dense layers with ReLU activation
Dropout for regularization
Final layer with sigmoid activation for binary classification

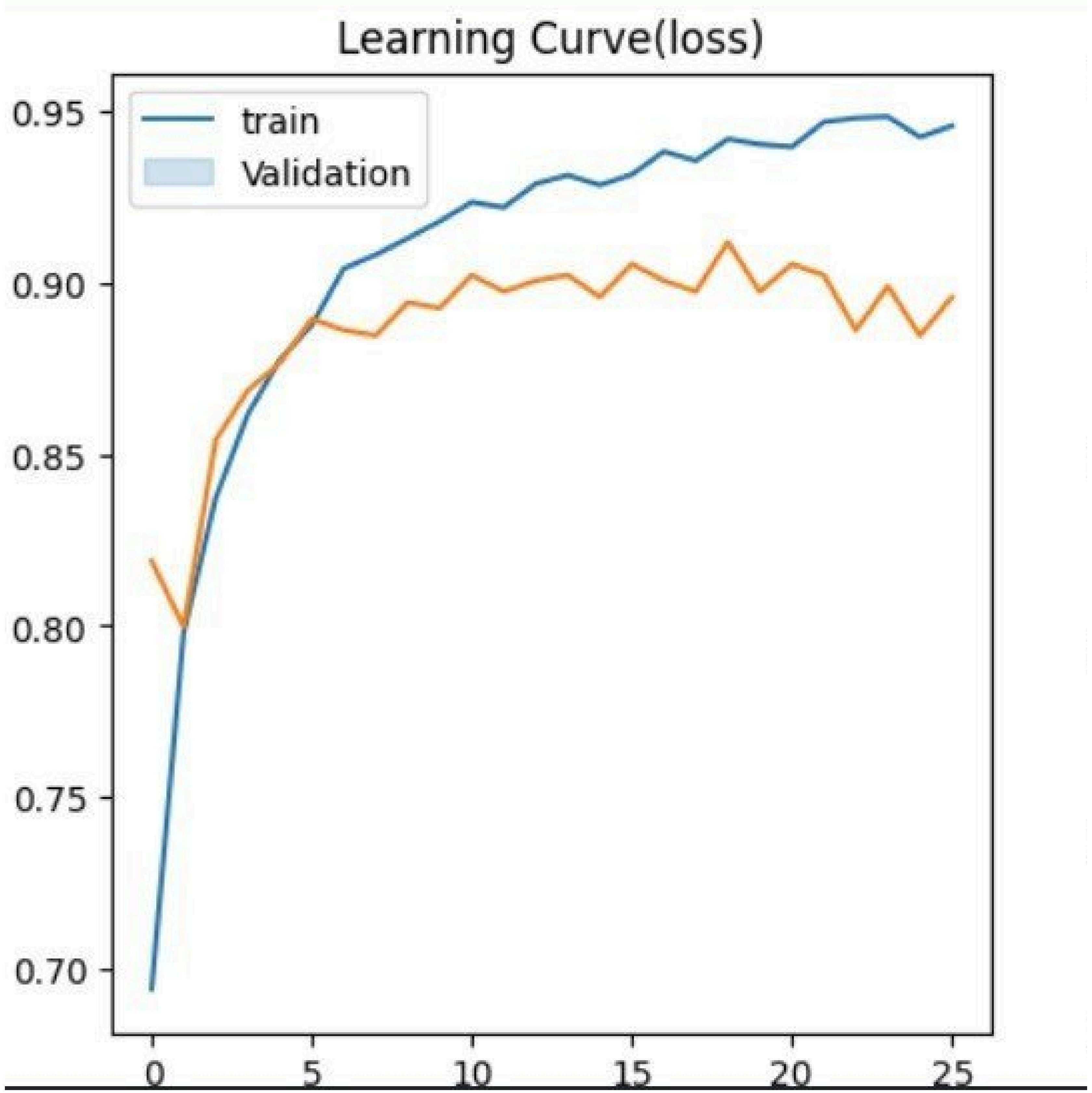
Training Process

Hyperparameters:

Optimizer: Adam

Loss: Binary Cross-Entropy

Learning rate: 5×10^{-5}



Model Evaluation

Evaluation metrics used:

Binary Accuracy, Confusion Matrix.

Accuracy on the validation set: 94%

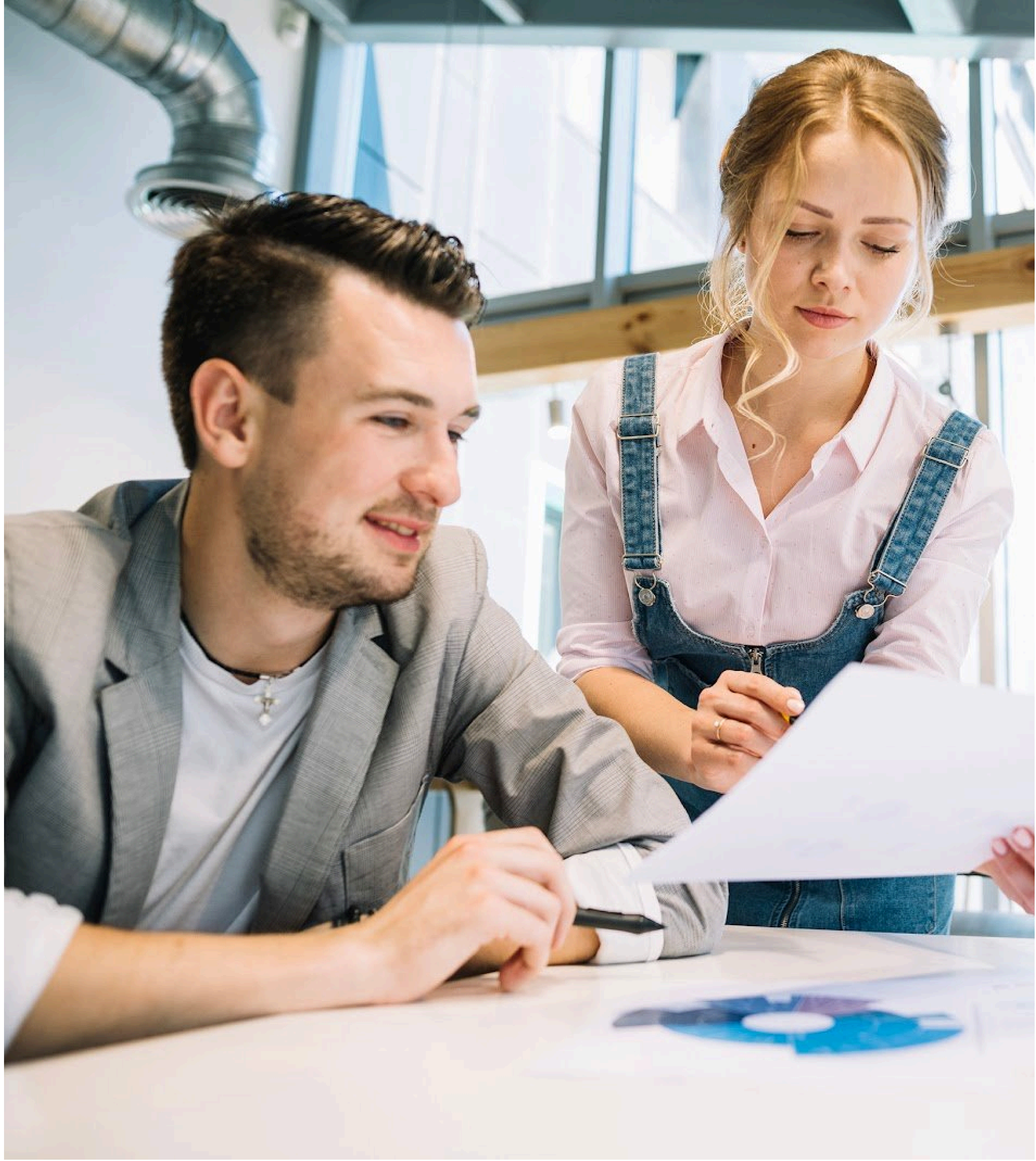
Overfitting: 4%

Loss decreased over epochs, improving performance.

Model Deployment

Save the trained model. Upload to Hugging Face repository.

Set up an interface using Gradio for real-time image classification.



Code Implementation

Model creation

(Pre_trained_model())Training code

block (fit method with callbacks)Model saving (save method for deployment)

```
def get_model():

    #Input shape = [width, height, color channels]
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

    # Block One
    x = layers.Conv2D(filters=16, kernel_size=3, padding='valid')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Dropout(0.2)(x)

    # Block Two
    x = layers.Conv2D(filters=32, kernel_size=3, padding='valid')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Dropout(0.2)(x)

    # Block Three
    x = layers.Conv2D(filters=64, kernel_size=3, padding='valid')(x)
    x = layers.Conv2D(filters=64, kernel_size=3, padding='valid')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Dropout(0.4)(x)

    # Head
    #x = layers.BatchNormalization()(x)
    x = layers.Flatten()(x)
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    #Final Layer (Output)
    output = layers.Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[inputs], outputs=output)
```

Code Implementation

```
] model_pretrained = Pre_trained_model()
  model_pretrained.compile(loss='binary_crossentropy'
                   , optimizer =optimizers.Adam(learning_rate=5e-5), metrics=['binar

  model_pretrained.summary()
```

```
] ##1 perfect
  history_pretrained=model_pretrained.fit(
      train_generator,
      epochs=20,
      validation_data=val_generator,
      callbacks=[model_checkpoint_callback,Early_Stopping,reduce_lr]
  )
```

```
] ##2
  history_pretrained=model_pretrained.fit(
      train_generator,
      epochs=20,
      validation_data=val_generator,
      callbacks=[model_checkpoint_callback,Early_Stopping,reduce_lr]
  )
```

Code Implementation

Model creation

(Pre_trained_model())Training code

block (fit method with callbacks)Model saving (save method for deployment)

```python
def get_model():

    #Input shape = [width, height, color channels]
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

    # Block One
    x = layers.Conv2D(filters=16, kernel_size=3, padding='valid')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Dropout(0.2)(x)

    # Block Two
    x = layers.Conv2D(filters=32, kernel_size=3, padding='valid')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Dropout(0.2)(x)

    # Block Three
    x = layers.Conv2D(filters=64, kernel_size=3, padding='valid')(x)
    x = layers.Conv2D(filters=64, kernel_size=3, padding='valid')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Dropout(0.4)(x)

    # Head
    #x = layers.BatchNormalization()(x)
    x = layers.Flatten()(x)
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    #Final Layer (Output)
    output = layers.Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[inputs], outputs=output)
```

Results & Conclusion

Thanks!