# Preamble:

Pneumonia is a prevalent and potentially fatal respiratory infection affecting individuals globally. Early detection and accurate diagnosis are vital for effective treatment and management. Traditional diagnostic methods, such as clinical examination and laboratory tests, often require considerable time and specialized expertise. Deep learning models, such as Convolutional Neural Networks (CNNs) and Transfer Learning, have demonstrated promising results in medical image analysis for pneumonia detection. This study compares the performance of four deep learning models which are CNN, VGG16, ResNet152V2, and InceptionV3, in predicting pneumonia using a chest X-ray dataset. The methodology encompasses pre-processing of data set, defining each model's architecture, and training and evaluating the models using various metrics. Among the models, VGG16 achieved the highest validation accuracy of 97% and test accuracy of 89%. A subsequent ANOVA analysis revealed a statistically significant difference in classification performance among the model types, where VGG16 and InceptionV3 showed a significant difference in performance. This research provides insights into the comparative performance of these models in pneumonia classification and may inform the selection of suitable models for specific tasks. Ultimately, it may contribute to developing more accurate and efficient tools for early detection and treatment of pneumonia, significantly impacting public health.

# Research Question:

To investigate the performance of convolutional neural networks (CNNs) and transfer learning models for pneumonia detection in chest X-ray images, this study aims to answer the following research questions:

==> item What is the classification performance of Convolutional Neural Network (CNN) and Transfer Learning Models (ResNet152V2, InceptionV3, and VGG16) for pneumonia detection in chest X-ray images?

==>item Is there a statistically significant difference in classification performance between the CNN and the Transfer Learning Models for pneumonia detection in chest X-ray images?

==>item If there is a statistically significant difference in classification performance, which models differ significantly from each other?

# Proposed Workflow:

The effectiveness of four deep learning models, namely the convolutional neural network (CNN) which is the custom model along with three transfer learning models namely ResNet152V2, InceptionV3 and VGG16, for the prediction of pneumonia was examined through a research study. A large dataset of chest X-rays was collected from the open source called "Kaggle", which was pre-processed and divided into training and testing subsets. The architecture of each model was carefully defined, including the number and type of layers, activation functions, and hyperparameters. Subsequently, all models were trained using the same dataset and monitored for performance. Finally, a range of metrics, including accuracy, precision, recall, and F1 score, were used to evaluate the models. An Anova test for statistical analysis for one-way and post-hoc will be performed at the end.

```
## install D2l
!pip install d2l==1.0.0-beta0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Requirement already satisfied: d2l==1.0.0-beta0 in /usr/local/lib/python3.8/d
Requirement already satisfied: jupyter in /usr/local/lib/python3.8/dist-packa
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: gpytorch in /usr/local/lib/python3.8/dist-pack
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-package
Requirement already satisfied: gym==0.21.0 in /usr/local/lib/python3.8/dist-p
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-package
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packag
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.8/
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-pack
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.8
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-
Requirement already satisfied: linear-operator>=0.2.0 in /usr/local/lib/pytho
Requirement already satisfied: ipykernel in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: nbconvert in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.8/di
Requirement already satisfied: notebook in /usr/local/lib/python3.8/dist-pack
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: qtconsole in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.8/d
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.8/dist
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.8/
Requirement already satisfied: traitlets in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python
Requirement already satisfied: torch>=1.11 in /usr/local/lib/python3.8/dist-p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-pack
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.8/dis
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.8/dis
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.8/dist-
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/py
Requirement already satisfied: ipython-genutils~=0.2.0 in /usr/local/lib/pyth
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/py
Requirement already satisfied: pygments in /usr/local/lib/python3.8/dist-pack
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 i
```

```
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3
Requirement already satisfied: nbformat>=4.4 in /usr/local/lib/python3.8/dist-
Requirement already satisfied: testpath in /usr/local/lib/python3.8/dist-packa
Requirement already satisfied: defusedxml in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: jinja2>=2.4 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.8/
Requirement already satisfied: bleach in /usr/local/lib/python3.8/dist-package
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.8/dist-
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.8
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: Send2Trash>=1.5.0 in /usr/local/lib/python3.8/
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.8/
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.8/d
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
import pandas as pd
import matplotlib as mat
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
%matplotlib inline

pd.options.display.max_colwidth = 100

import random
import os

from numpy.random import seed
seed(42)

random.seed(42)
os.environ['PYTHONHASHSEED'] = str(42)
os.environ['TF_DETERMINISTIC_OPS'] = '1'

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import callbacks
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import glob
import cv2

from tensorflow.random import set_seed
set_seed(42)

import warnings
warnings.filterwarnings('ignore')


from IPython.utils.process import shutil
### unpack the data folders
import shutil
shutil.unpack_archive('/content/drive/MyDrive/chest_xray.zip')


IMG_SIZE = 224
BATCH = 32
SEED = 42
```

```
main_path = "/content"
train_path = os.path.join(main_path, "train")
test_path = os.path.join(main_path, "test")
train_path = os.path.join(main_path, "train")
test_path = os.path.join(main_path, "test")
train_normal = glob.glob(os.path.join(train_path, "NORMAL/*.jpeg"))
train_pneumonia = glob.glob(os.path.join(train_path, "PNEUMONIA/*.jpeg"))
test_normal = glob.glob(os.path.join(test_path, "NORMAL/*.jpeg"))
test_pneumonia = glob.glob(os.path.join(test_path, "PNEUMONIA/*.jpeg"))


train_list = [x for x in train_normal]
train_list.extend([x for x in train_pneumonia])

df_train = pd.DataFrame(np.concatenate([['Normal']*len(train_normal) , ['Pneumoni
df_train['image'] = [x for x in train_list]

test_list = [x for x in test_normal]
test_list.extend([x for x in test_pneumonia])

df_test = pd.DataFrame(np.concatenate([['Normal']*len(test_normal) , ['Pneumonia'
df_test['image'] = [x for x in test_list]
```

```
df_train.shape
```

```
(5232, 2)
```

```
df_test.shape
```

```
(624, 2)
```

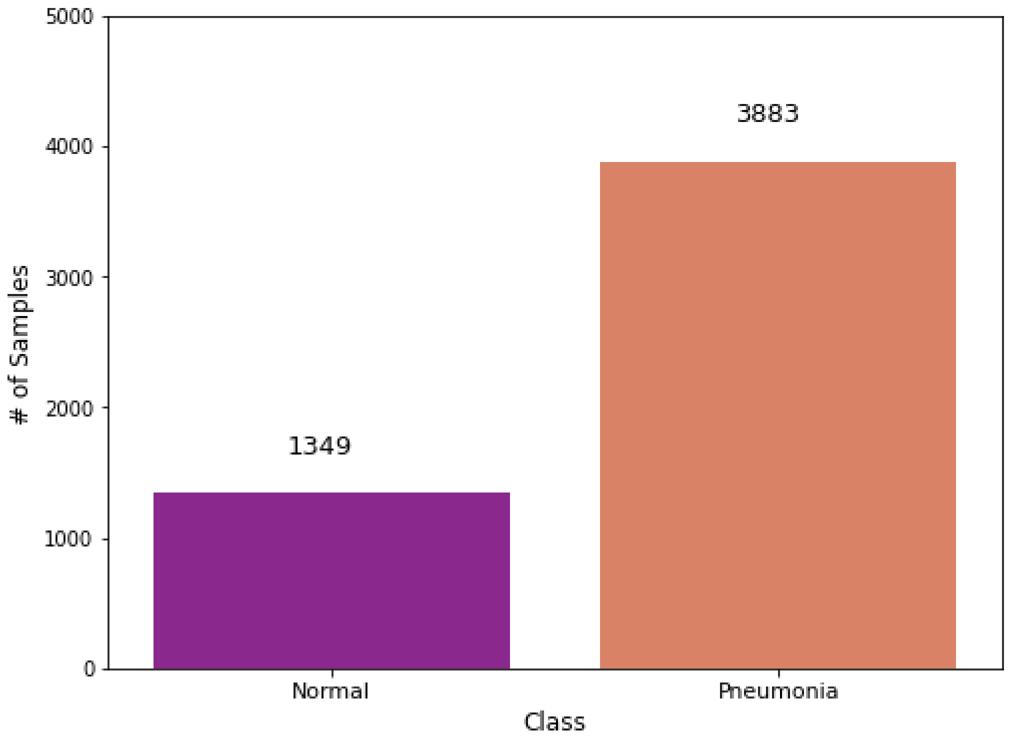
```
df_train.head()
```

| | class | image |
|---|---|---|
| **0** | Normal | /content/train/NORMAL/NORMAL-4656588-0001.jpeg |
| **1** | Normal | /content/train/NORMAL/NORMAL-705474-0001.jpeg |
| **2** | Normal | /content/train/NORMAL/NORMAL-9382452-0001.jpeg |
| **3** | Normal | /content/train/NORMAL/NORMAL-483610-0001.jpeg |
| **4** | Normal | /content/train/NORMAL/NORMAL-4093513-0001.jpeg |

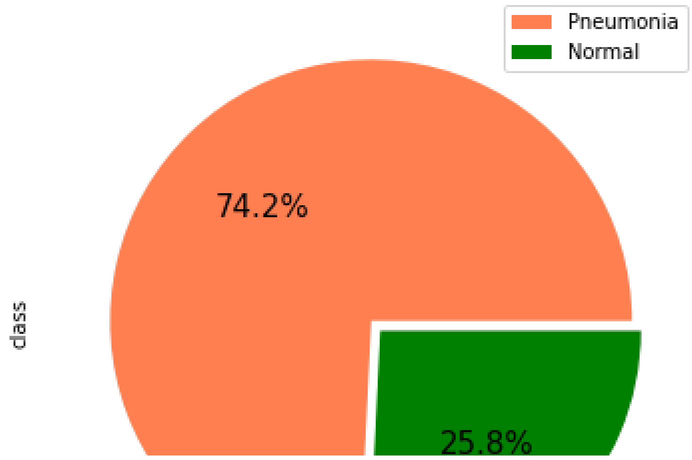
```
df_test.head()
```

| | class | image |
|---|---|---|
| **0** | Normal | /content/test/NORMAL/NORMAL-3267425-0001.jpeg |

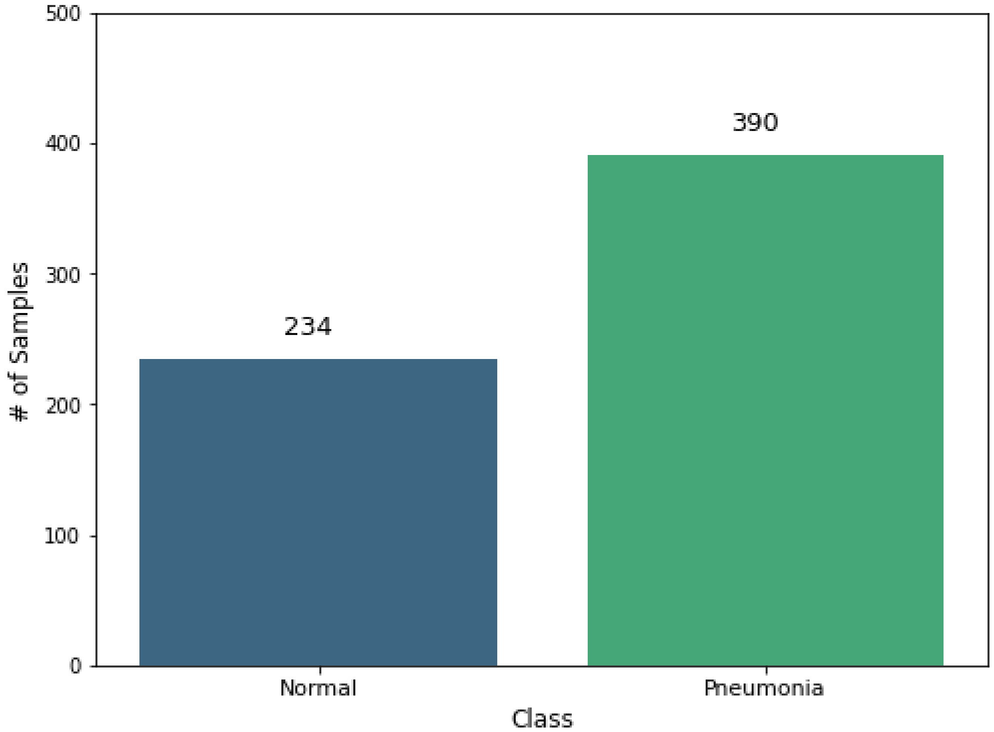
```
plt.figure(figsize=(8,6))

ax = sns.countplot(x='class', data=df_train, palette="plasma")

plt.xlabel("Class", fontsize= 12)
plt.ylabel("# of Samples", fontsize= 12)
plt.ylim(0,5000)
plt.xticks([0,1], ['Normal', 'Pneumonia'], fontsize = 11)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x()+0.30, p.get_height()+300), fontsize

plt.show()
```



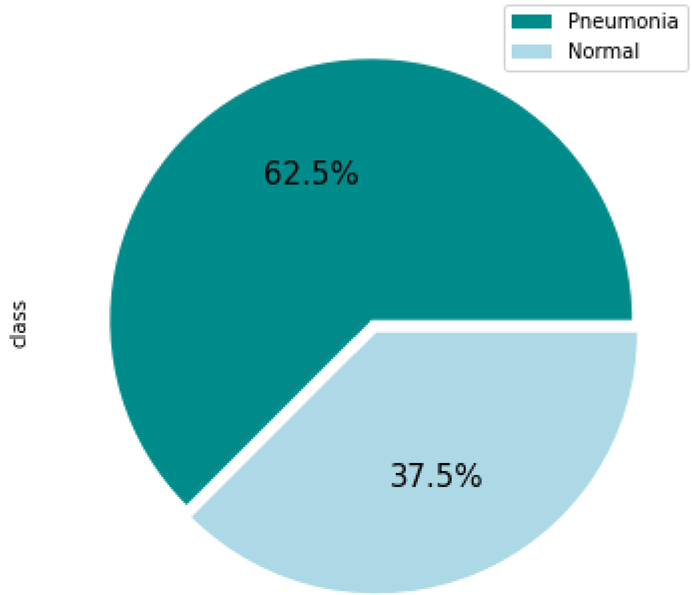

```
plt.figure(figsize=(9,6))

df_train['class'].value_counts().plot(kind='pie',labels = ['',''],
                                      autopct='%1.1f%%', colors = ['coral','green
                                      explode = [0,0.05], textprops = {"fontsize"

plt.legend(labels=['Pneumonia', 'Normal'])
plt.show()
```




```
plt.figure(figsize=(8,6))

ax = sns.countplot(x='class', data=df_test, palette="viridis")

plt.xlabel("Class", fontsize= 12)
plt.ylabel("# of Samples", fontsize= 12)
plt.ylim(0,500)
plt.xticks([0,1], ['Normal', 'Pneumonia'], fontsize = 11)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x()+0.32, p.get_height()+20), fontsize =

plt.show()
```

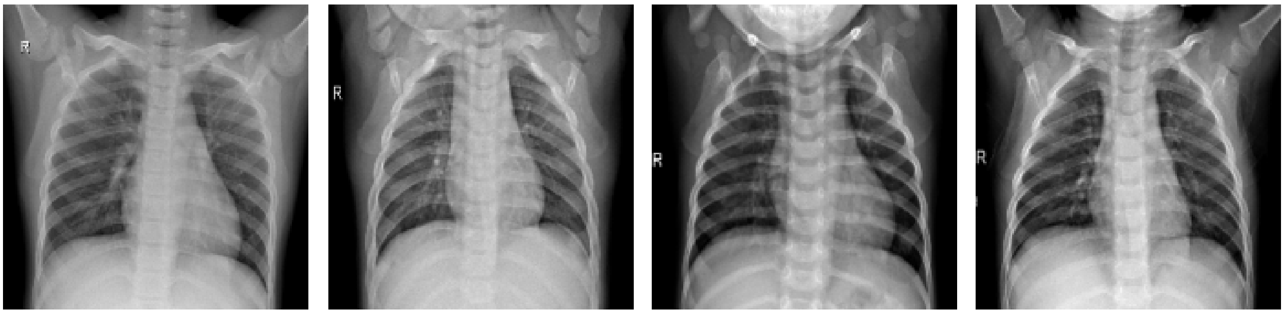


```
plt.figure(figsize=(9,6))

df_test['class'].value_counts().plot(kind='pie',labels = ['',''],
                                     autopct='%1.1f%%', colors = ['darkcyan','lig
                                     explode = [0,0.05], textprops = {"fontsize":

plt.legend(labels=['Pneumonia', 'Normal'])
plt.show()
```



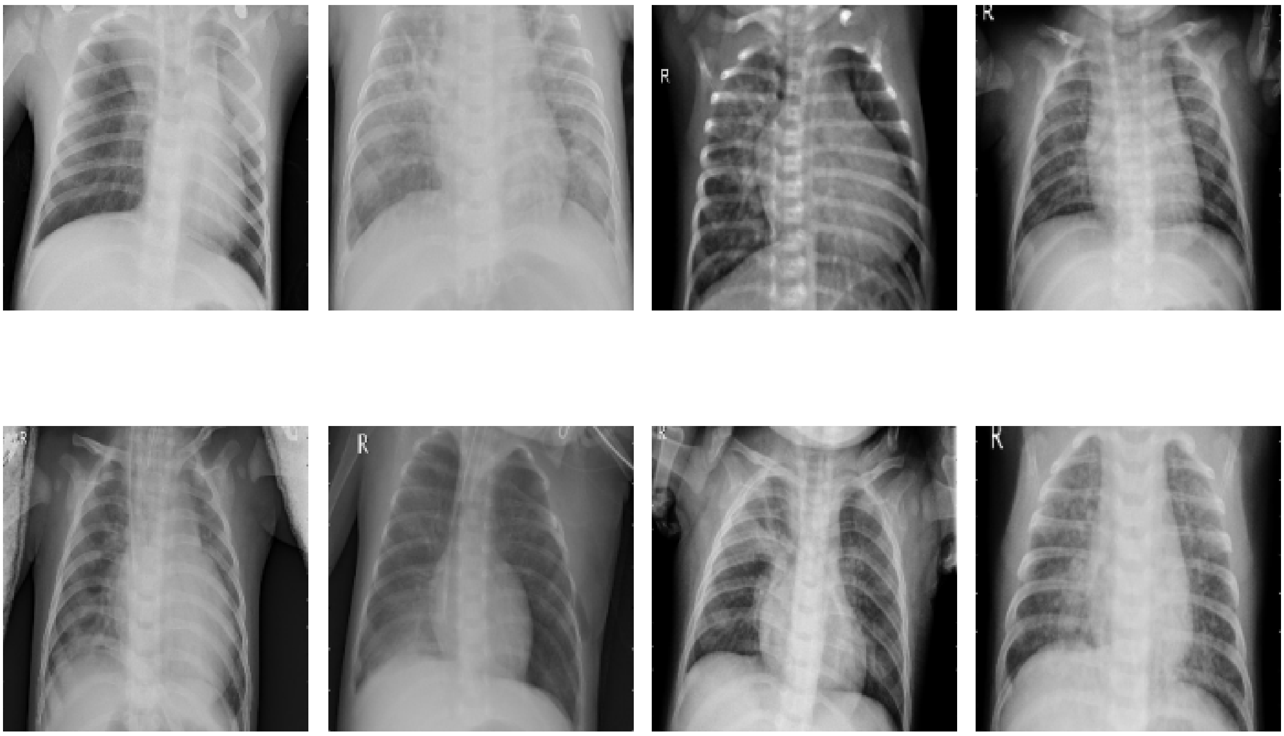

```
print('Train Set:Normal')

plt.figure(figsize=(10,10))

for i in range(0, 8):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(train_normal[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Train Set:Normal
```



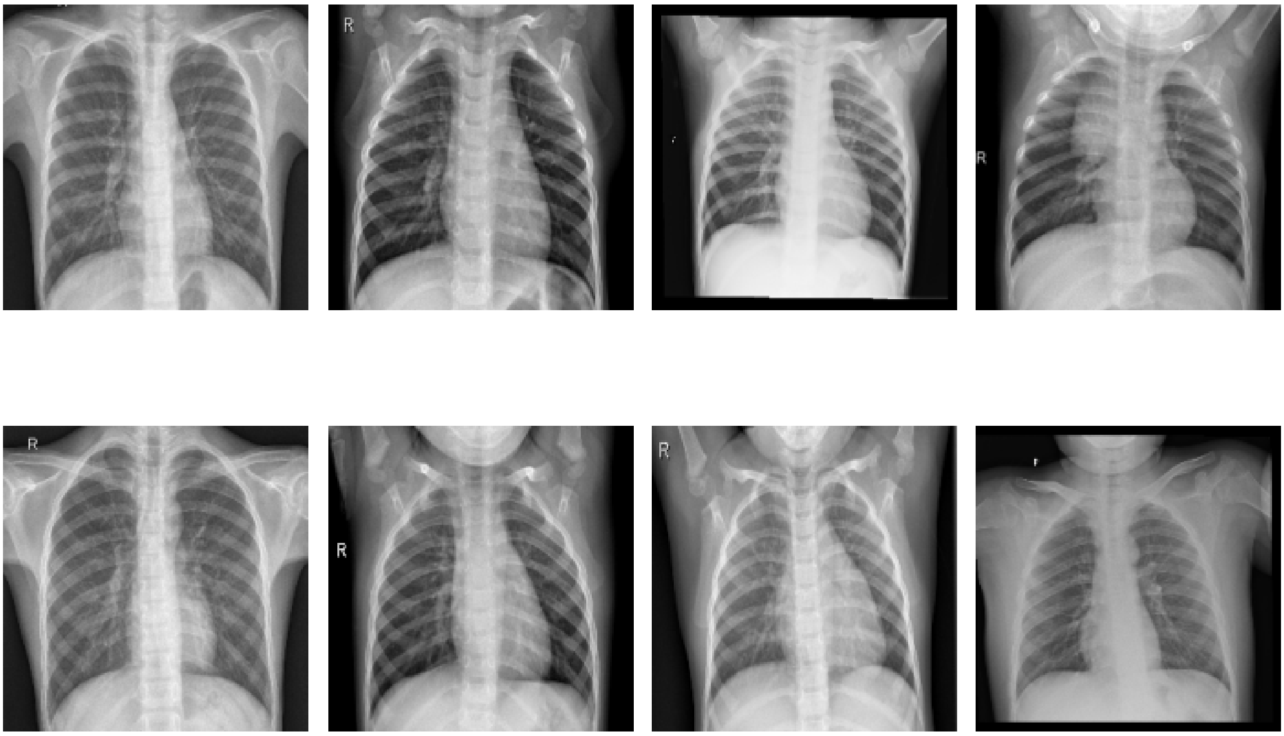
```
print('Train Set:Pneumonia')

plt.figure(figsize=(10,10))

for i in range(0, 8):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(train_pneumonia[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Train Set:Pneumonia
```



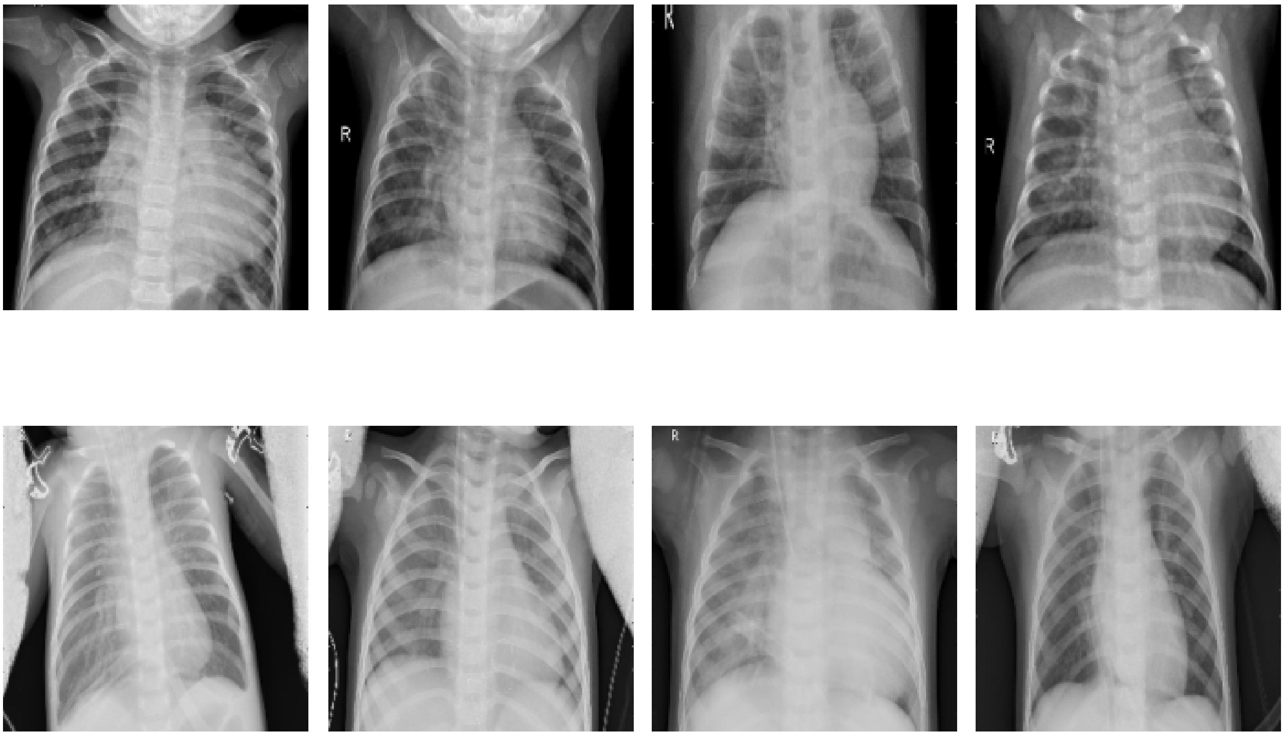
```
print('Test Set: Normal')

plt.figure(figsize=(10,10))

for i in range(0, 8):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(test_normal[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Test Set: Normal
```



```
print('Test Set: Pneumonia')

plt.figure(figsize=(10,10))

for i in range(0, 8):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(test_pneumonia[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Test Set: Pneumonia
```



## Data Preparation

```
print(type(df_train))
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
train_df, val_df = train_test_split(df_train, test_size = 0.20, random_state = SE
```

```
train_df.head()
```

| | class | image |
|---|---|---|
| **3566** | Pneumonia | /content/train/PNEUMONIA/BACTERIA-6678407-0001.jpeg |
| **2866** | Pneumonia | /content/train/PNEUMONIA/BACTERIA-7847892-0006.jpeg |
| **2681** | Pneumonia | /content/train/PNEUMONIA/BACTERIA-1982399-0002.jpeg |
| **1199** | Normal | /content/train/NORMAL/NORMAL-3688916-0002.jpeg |
| **4619** | Pneumonia | /content/train/PNEUMONIA/BACTERIA-1069837-0002.jpeg |

```
val_df.head()
```

| | class | image |
|---|---|---|
| **2945** | Pneumonia | /content/train/PNEUMONIA/BACTERIA-232309-0002.jpeg |
| **4878** | Pneumonia | /content/train/PNEUMONIA/VIRUS-8550709-0010.jpeg |
| **3177** | Pneumonia | /content/train/PNEUMONIA/BACTERIA-2603500-0001.jpeg |

```
print(val_df.shape)
```

```
(1047, 2)
```

load the images from the folders and prepare them to feed our models.

```
train_datagen = ImageDataGenerator(rescale=1/255.,
                                  zoom_range = 0.1,
                                  #rotation_range = 0.1,
                                  width_shift_range = 0.1,
                                  height_shift_range = 0.1)

val_datagen = ImageDataGenerator(rescale=1/255.)

ds_train = train_datagen.flow_from_dataframe(train_df,
                                             #directory=train_path, #dataframe co
                                             x_col = 'image',
                                             y_col = 'class',
                                             target_size = (IMG_SIZE, IMG_SIZE),
                                             class_mode = 'binary',
                                             batch_size = BATCH,
                                             seed = SEED)

ds_val = val_datagen.flow_from_dataframe(val_df,
                                            #directory=train_path,
                                            x_col = 'image',
                                            y_col = 'class',
                                            target_size = (IMG_SIZE, IMG_SIZE),
                                            class_mode = 'binary',
                                            batch_size = BATCH,
                                            seed = SEED)

ds_test = val_datagen.flow_from_dataframe(df_test,
                                            #directory=test_path,
                                            x_col = 'image',
                                            y_col = 'class',
                                            target_size = (IMG_SIZE, IMG_SIZE),
                                            class_mode = 'binary',
                                            batch_size = 1,
                                            shuffle = False)
```

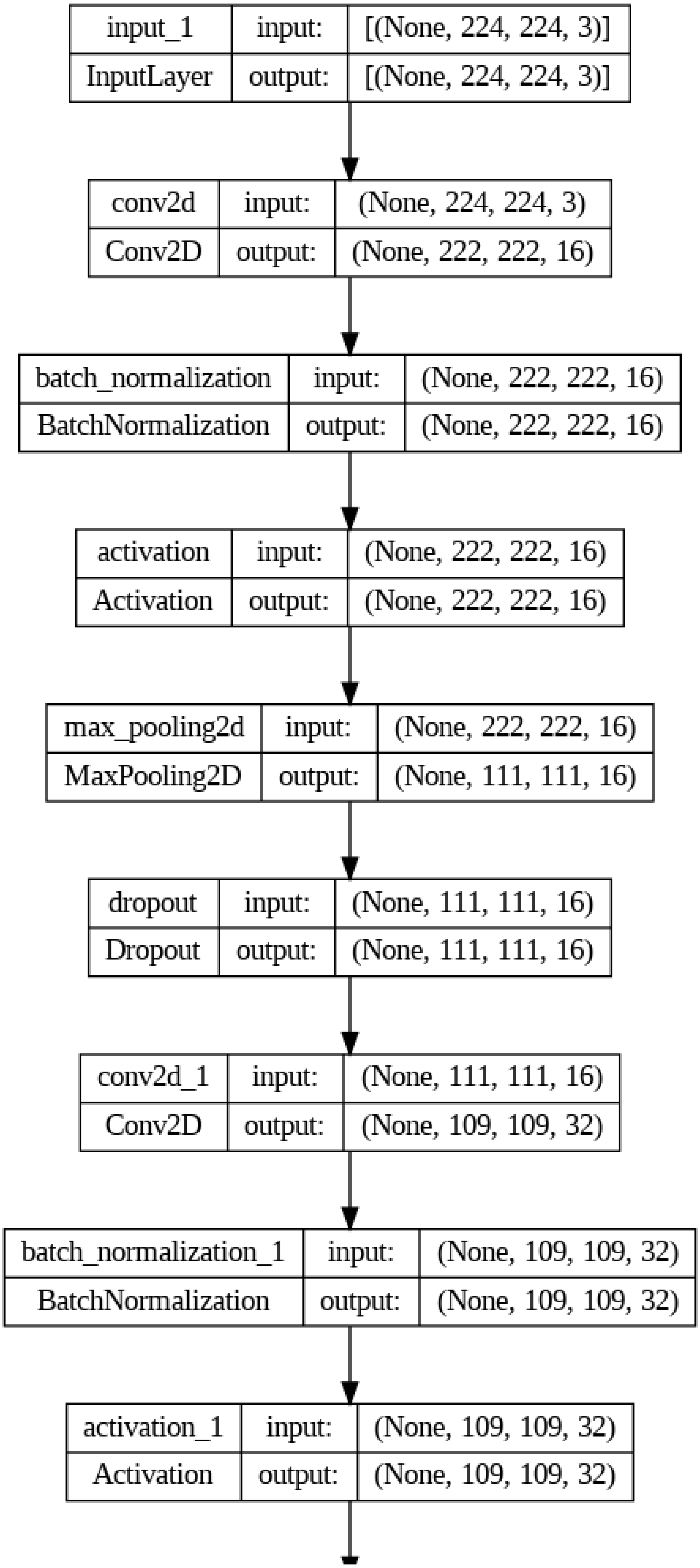
```
Found 4185 validated image filenames belonging to 2 classes.
Found 1047 validated image filenames belonging to 2 classes.
Found 624 validated image filenames belonging to 2 classes.
```

## CNN Building

```
#Setting callbakcs

early_stopping = callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    min_delta=1e-7,
    restore_best_weights=True,
)

plateau = callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor = 0.2,
    patience = 2,
    min_delt = 1e-7,
    cooldown = 0,
    verbose = 1
)
```

```
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 = keras.Model(inputs=[inputs], outputs=output)

    return model
```

```
from keras.utils import plot_model

model = get_model()
plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```


input_1 | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
conv2d | input: | (None, 224, 224, 3)
Conv2D | output: | (None, 222, 222, 16)
batch_normalization | input: | (None, 222, 222, 16)
BatchNormalization | output: | (None, 222, 222, 16)
activation | input: | (None, 222, 222, 16)
Activation | output: | (None, 222, 222, 16)
max_pooling2d | input: | (None, 222, 222, 16)
MaxPooling2D | output: | (None, 111, 111, 16)
dropout | input: | (None, 111, 111, 16)
Dropout | output: | (None, 111, 111, 16)
conv2d_1 | input: | (None, 111, 111, 16)
Conv2D | output: | (None, 109, 109, 32)
batch_normalization_1 | input: | (None, 109, 109, 32)
BatchNormalization | output: | (None, 109, 109, 32)
activation_1 | input: | (None, 109, 109, 32)
Activation | output: | (None, 109, 109, 32)

```
keras.backend.clear_session()

model = get_model()
model.compile(loss='binary_crossentropy'
              , optimizer = keras.optimizers.Adam(learning_rate=3e-5), metrics='b

model.summary()
```

```
    Model: "model"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     input_1 (InputLayer)        [(None, 224, 224, 3)]     0

     conv2d (Conv2D)             (None, 222, 222, 16)      448

     batch_normalization (BatchN  (None, 222, 222, 16)     64
     ormalization)

     activation (Activation)     (None, 222, 222, 16)      0

     max_pooling2d (MaxPooling2D  (None, 111, 111, 16)     0
     )

     dropout (Dropout)           (None, 111, 111, 16)      0

     conv2d_1 (Conv2D)           (None, 109, 109, 32)      4640

     batch_normalization_1 (Batc  (None, 109, 109, 32)     128
     hNormalization)

     activation_1 (Activation)   (None, 109, 109, 32)      0

     max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0
     2D)

     dropout_1 (Dropout)         (None, 54, 54, 32)        0

     conv2d_2 (Conv2D)           (None, 52, 52, 64)        18496

     conv2d_3 (Conv2D)           (None, 50, 50, 64)        36928

     batch_normalization_2 (Batc  (None, 50, 50, 64)       256
     hNormalization)

     activation_2 (Activation)   (None, 50, 50, 64)        0

     max_pooling2d_2 (MaxPooling  (None, 25, 25, 64)       0
     2D)

     dropout_2 (Dropout)         (None, 25, 25, 64)        0

     flatten (Flatten)           (None, 40000)             0

     dense (Dense)               (None, 64)                2560064
```

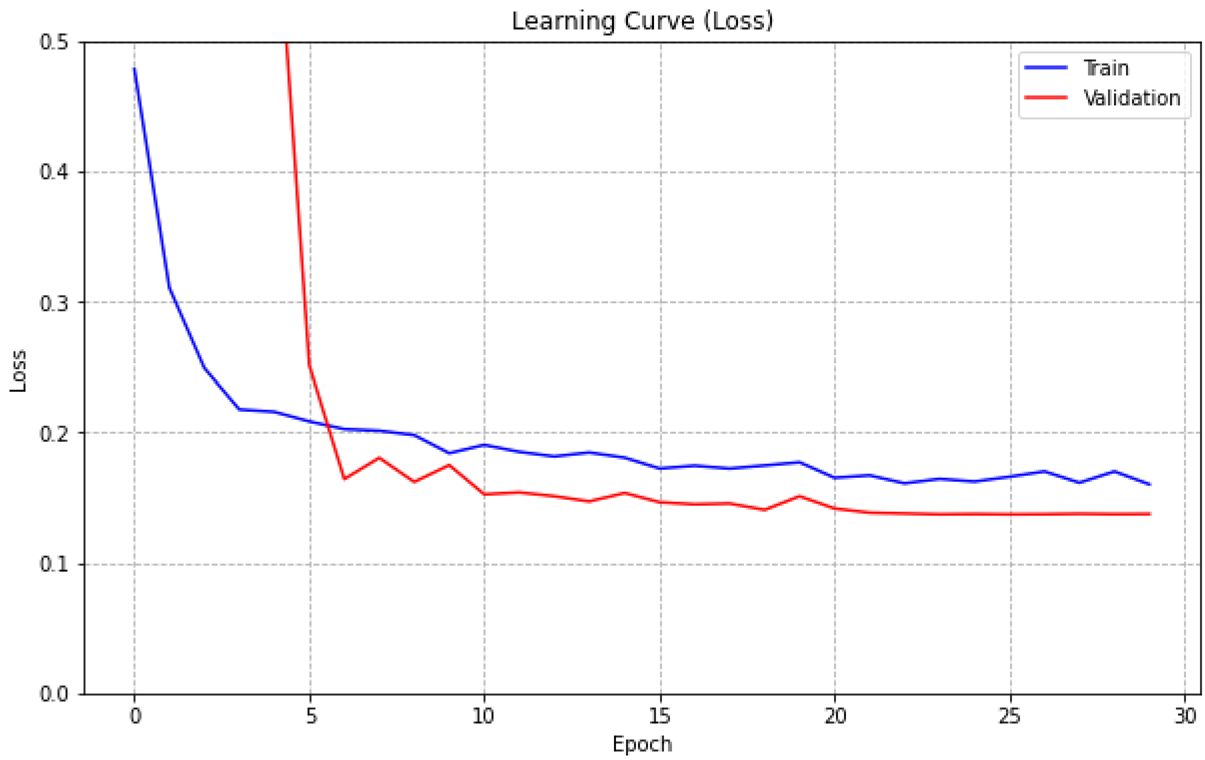
```
 dropout_3 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,621,089
Trainable params: 2,620,865
Non-trainable params: 224
_________________________________________________________________
```

```
history = model.fit(ds_train,
          batch_size = BATCH, epochs = 30,
          validation_data=ds_val,
          callbacks=[early_stopping, plateau],
          steps_per_epoch=(len(train_df)/BATCH),
          validation_steps=(len(val_df)/BATCH));
```

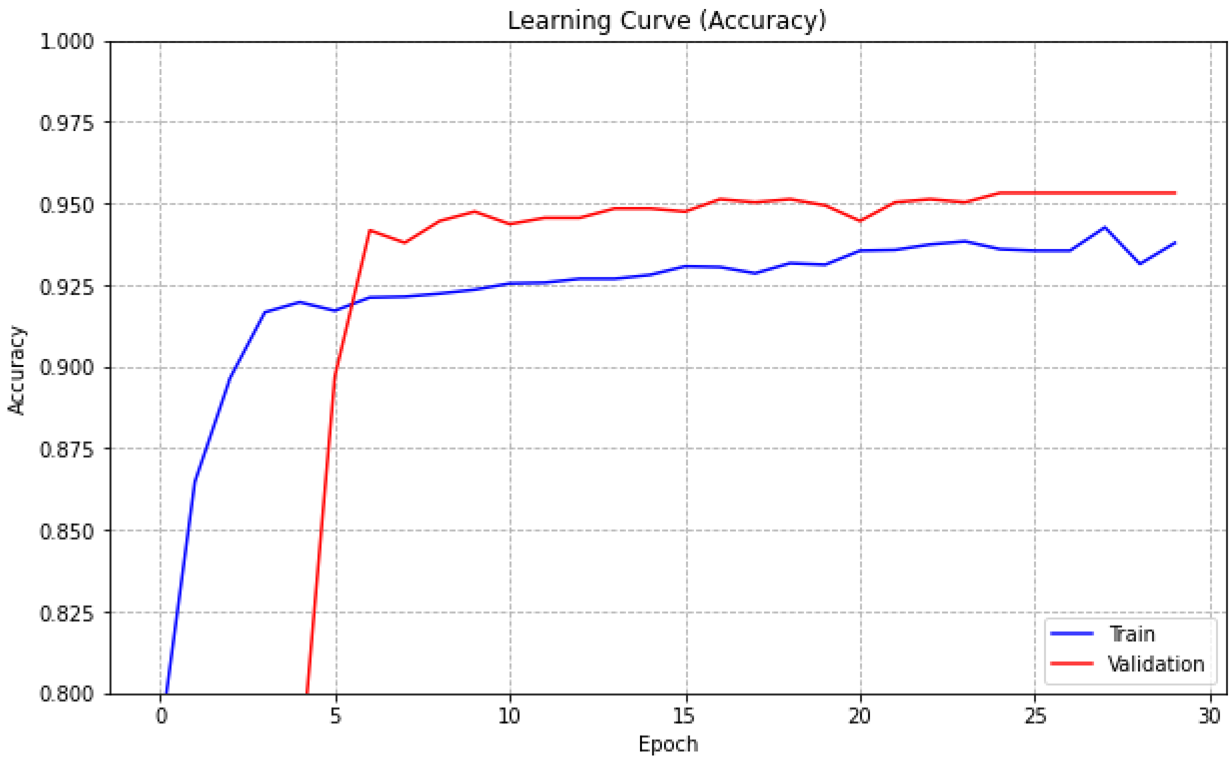
```
Epoch 1/30
130/130 [==============================] - 138s 941ms/step - loss: 0.4782 - b
Epoch 2/30
130/130 [==============================] - 119s 909ms/step - loss: 0.3106 - b
Epoch 3/30
131/130 [==============================] - ETA: 0s - loss: 0.2491 - binary_ac
Epoch 3: ReduceLROnPlateau reducing learning rate to 5.9999998484272515e-06.
130/130 [==============================] - 113s 863ms/step - loss: 0.2491 - b
Epoch 4/30
130/130 [==============================] - 112s 859ms/step - loss: 0.2175 - b
Epoch 5/30
130/130 [==============================] - 112s 861ms/step - loss: 0.2157 - b
Epoch 6/30
130/130 [==============================] - 122s 933ms/step - loss: 0.2084 - b
Epoch 7/30
130/130 [==============================] - 122s 932ms/step - loss: 0.2025 - b
Epoch 8/30
130/130 [==============================] - 115s 880ms/step - loss: 0.2013 - b
Epoch 9/30
130/130 [==============================] - 121s 930ms/step - loss: 0.1979 - b
Epoch 10/30
130/130 [==============================] - 123s 938ms/step - loss: 0.1840 - b
Epoch 11/30
130/130 [==============================] - 122s 934ms/step - loss: 0.1904 - b
Epoch 12/30
130/130 [==============================] - 113s 862ms/step - loss: 0.1851 - b
Epoch 13/30
130/130 [==============================] - 126s 961ms/step - loss: 0.1816 - b
Epoch 14/30
130/130 [==============================] - 122s 931ms/step - loss: 0.1846 - b
Epoch 15/30
130/130 [==============================] - 123s 940ms/step - loss: 0.1806 - b
Epoch 16/30
130/130 [==============================] - 123s 938ms/step - loss: 0.1723 - b
Epoch 17/30
130/130 [==============================] - 121s 926ms/step - loss: 0.1745 - b
Epoch 18/30
130/130 [==============================] - 124s 947ms/step - loss: 0.1723 - b
Epoch 19/30
130/130 [==============================] - 122s 933ms/step - loss: 0.1746 - b
Epoch 20/30
```

```
130/130 [==============================] - 113s 864ms/step - loss: 0.1771 - b
Epoch 21/30
131/130 [==============================] - ETA: 0s - loss: 0.1652 - binary_ac
Epoch 21: ReduceLROnPlateau reducing learning rate to 1.1999999514955563e-06.
130/130 [==============================] - 121s 927ms/step - loss: 0.1652 - b
Epoch 22/30
130/130 [==============================] - 112s 855ms/step - loss: 0.1670 - b
Epoch 23/30
130/130 [==============================] - 123s 939ms/step - loss: 0.1609 - b
Epoch 24/30
130/130 [==============================] - 120s 916ms/step - loss: 0.1643 - b
Epoch 25/30
130/130 [==============================] - 121s 927ms/step - loss: 0.1623 - b
Epoch 26/30
131/130 [==============================] - ETA: 0s - loss: 0.1660 - binary_ac
Epoch 26: ReduceLROnPlateau reducing learning rate to 2.3999998575163774e-07.
```

```
fig, ax = plt.subplots(figsize=(10,6))
sns.lineplot(x=history.epoch, y=history.history['loss'], color='blue', label='Tra
sns.lineplot(x=history.epoch, y=history.history['val_loss'], color='red', label='
ax.set_title('Learning Curve (Loss)')
ax.set_ylabel('Loss')
ax.set_xlabel('Epoch')
ax.set_ylim(0, 0.5)
ax.legend(loc='upper right')
plt.grid(True, linestyle='--')
plt.show()
```



```
fig, ax = plt.subplots(figsize=(10,6))
sns.lineplot(x=history.epoch, y=history.history['binary_accuracy'], color='blue',
sns.lineplot(x=history.epoch, y=history.history['val_binary_accuracy'], color='re
ax.set_title('Learning Curve (Accuracy)')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.set_ylim(0.80, 1.0)
ax.legend(loc='lower right')
plt.grid(True, linestyle='--')
plt.show()
```




```
# Evaluate the model on the validation set
score = model.evaluate(ds_val, steps=len(val_df)//BATCH, verbose=1)

# Print the validation loss and accuracy
print('Val loss:', score[0])
print('Val accuracy:', score[1])
```

```
32/32 [==============================] - 13s 398ms/step - loss: 0.1378 - bina
Val loss: 0.13775357604026794
Val accuracy: 0.953125
```

```
# Evaluate the model on the Test set
score = model.evaluate(ds_test, steps = len(df_test), verbose = 1)

# Print the Test loss and accuracy
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

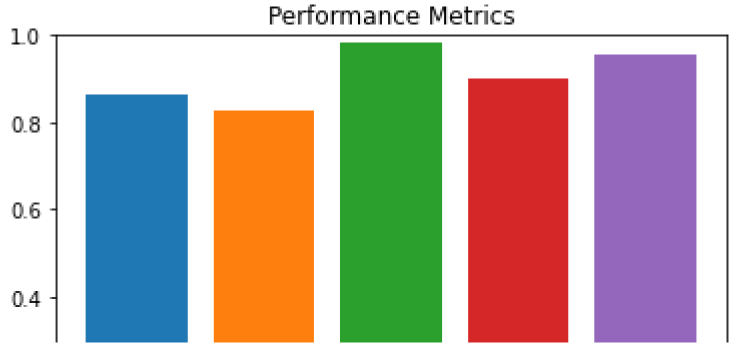
```
624/624 [==============================] - 12s 18ms/step - loss: 0.4482 - bin
Test loss: 0.44821447134017944
Test accuracy: 0.8621794581413269
```

## Performance Evaluation for CNN

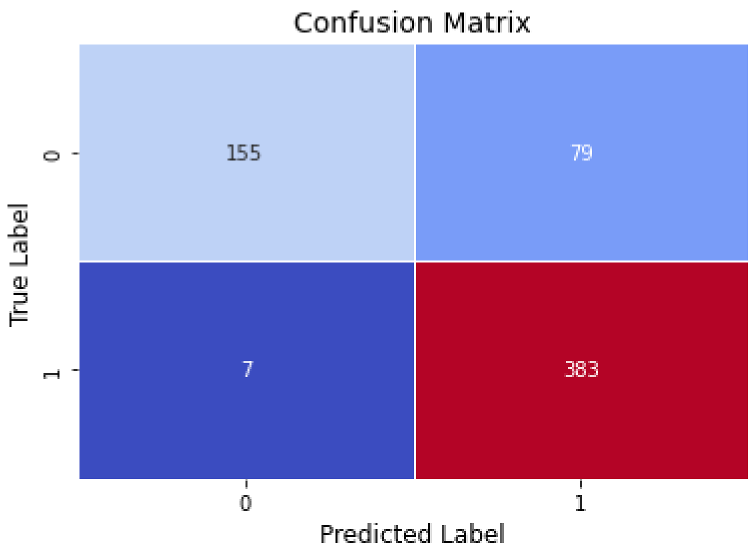
```
# Load the test data and labels
num_label = {'Normal': 0, 'Pneumonia': 1}
Y_test = df_test['class'].copy().map(num_label).astype('int')

# Make predictions on the test data
ds_test.reset()
predictions = model.predict(ds_test, steps=len(ds_test), verbose=0)
pred_labels = np.where(predictions>0.5, 1, 0)

# Compute the performance metrics
accuracy = metrics.accuracy_score(Y_test, pred_labels)
precision, recall, f1_score, _ = metrics.precision_recall_fscore_support(Y_test,
roc_auc = metrics.roc_auc_score(Y_test, predictions)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1_score))
print("ROC AUC: {:.2f}".format(roc_auc))
```

```
Accuracy: 0.86
Precision: 0.83
Recall: 0.98
F1 Score: 0.90
ROC AUC: 0.95
```

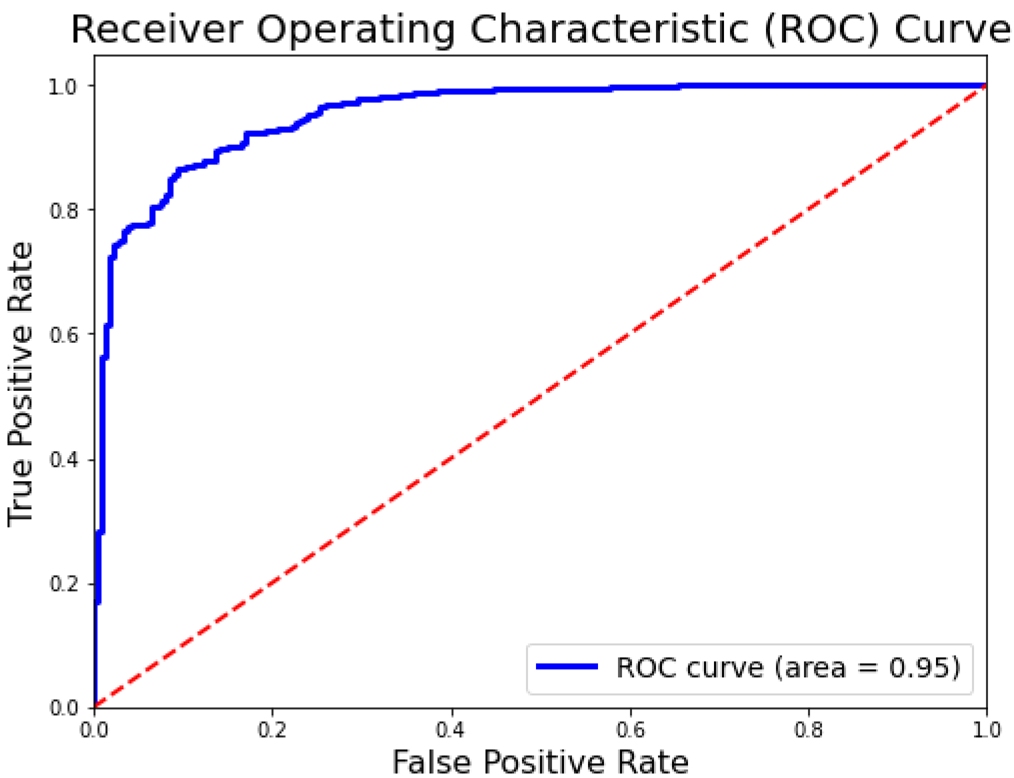
```
import matplotlib.pyplot as plt

# Define the performance metrics
performance_metrics = {
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1_score,
    'ROC AUC': roc_auc
}

# Plot the performance metrics as a bar chart
plt.bar(performance_metrics.keys(), performance_metrics.values(), color=['#1f77b4
plt.title('Performance Metrics')
plt.ylim([0, 1])
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the confusion matrix
conf_mat = metrics.confusion_matrix(Y_test, pred_labels)

# Plot the confusion matrix using seaborn
sns.heatmap(conf_mat, annot=True, cmap='coolwarm', fmt="d", linewidths=.5, cbar=F

# Set the axis labels and title
plt.xlabel("Predicted Label", fontsize= 12)
plt.ylabel("True Label", fontsize= 12)
plt.title("Confusion Matrix", fontsize= 14)

# Show the plot
plt.show()
```



```
import matplotlib.pyplot as plt
from sklearn import metrics

# Make predictions on the test data
ds_test.reset()
predictions = model.predict(ds_test, steps=len(ds_test), verbose=0)

# Compute the fpr and tpr for all thresholds of the classification
fpr, tpr, thresholds = metrics.roc_curve(Y_test, predictions)

# Compute the area under the ROC curve
roc_auc = metrics.auc(fpr, tpr)

# Plot ROC curve
fig, ax = plt.subplots(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=3, label='ROC curve (area = %0.2f)' % roc_auc
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('Receiver Operating Characteristic (ROC) Curve', fontsize=20)
plt.legend(loc="lower right", fontsize=14)
plt.show()
```




```
model.save('my_model.h5')
```

## Prediction with test data/images

```
img_width, img_height = 224, 224

# Load the image
img_path = '/content/test/NORMAL/NORMAL-1049278-0001.jpeg'
img = image.load_img(img_path, target_size=(img_width, img_height))

# Convert the image to a numpy array
img_array = image.img_to_array(img)

# Normalize the image array
img_array = img_array / 255.

# Reshape the array
img_array = np.expand_dims(img_array, axis=0)

# Prediction
prediction = model.predict(img_array)
```

```
1/1 [==============================] - 0s 220ms/step
```

```
prediction = model.predict(img_array)
predicted_class = np.argmax(prediction)
if predicted_class == 0:
    print("The image is classified as NORMAL.")
else:
    print("The image is classified as PNEUMONIA.")
```

```
1/1 [==============================] - 0s 70ms/step
The image is classified as NORMAL.
```

```
img_width, img_height = 224, 224

# Load the image
img_path = '/content/test/PNEUMONIA/BACTERIA-1135262-0001.jpeg'
img = image.load_img(img_path, target_size=(img_width, img_height))

# Convert the image to a numpy array
img_array = image.img_to_array(img)

# Normalize the image array
img_array = img_array / 255.

# Reshape the array
img_array = np.expand_dims(img_array, axis=0)

prediction = model.predict(img_array)
```

```
1/1 [==============================] - 0s 43ms/step
```

```
prediction = model.predict(img_array)
predicted_class = np.argmax(prediction)
if predicted_class == 1:
    print("The image is classified as NORMAL.")
else:
    print("The image is classified as PNEUMONIA.")
```

```
1/1 [==============================] - 0s 29ms/step
The image is classified as PNEUMONIA.
```

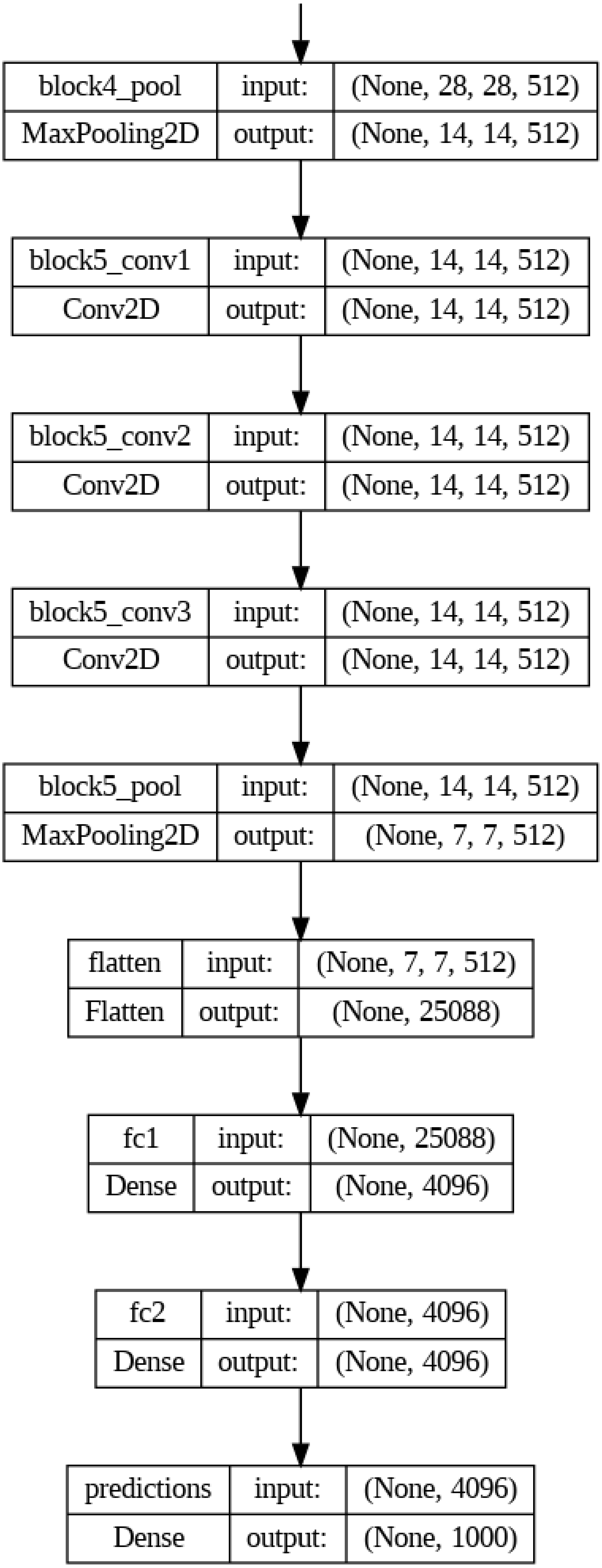
## Transfer learning

# VGG16

```
base_model = tf.keras.applications.VGG16(
    weights='imagenet',
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    include_top=False)

base_model.trainable = False

def get_pretrained():

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

    x = base_model(inputs)

    # Head
    x = layers.Flatten()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.1)(x)

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

    model = keras.Model(inputs=[inputs], outputs=output)

    return model
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applica
58889256/58889256 [==============================] - 0s 0us/step
```

```
from keras.utils.vis_utils import plot_model
from keras.applications.vgg16 import VGG16

# VGG16 model
model = VGG16()

# Model architecture
plot_model(model, show_shapes=True, show_layer_names=True)
```


block4_pool
MaxPooling2D
input:
output:
(None, 28, 28, 512)
(None, 14, 14, 512)
block5_conv1
Conv2D
input:
output:
(None, 14, 14, 512)
(None, 14, 14, 512)
block5_conv2
Conv2D
input:
output:
(None, 14, 14, 512)
(None, 14, 14, 512)
block5_conv3
Conv2D
input:
output:
(None, 14, 14, 512)
(None, 14, 14, 512)
block5_pool
MaxPooling2D
input:
output:
(None, 14, 14, 512)
(None, 7, 7, 512)
flatten
Flatten
input:
output:
(None, 7, 7, 512)
(None, 25088)
fc1
Dense
input:
output:
(None, 25088)
(None, 4096)
fc2
Dense
input:
output:
(None, 4096)
(None, 4096)
predictions
Dense
input:
output:
(None, 4096)
(None, 1000)

```
keras.backend.clear_session()
model_pretrained = get_pretrained()
model_pretrained.compile(loss='binary_crossentropy',
                         optimizer=keras.optimizers.Adam(learning_rate=5e-5),
                         metrics='binary_accuracy')
model_pretrained.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dropout (Dropout)           (None, 128)               0
```

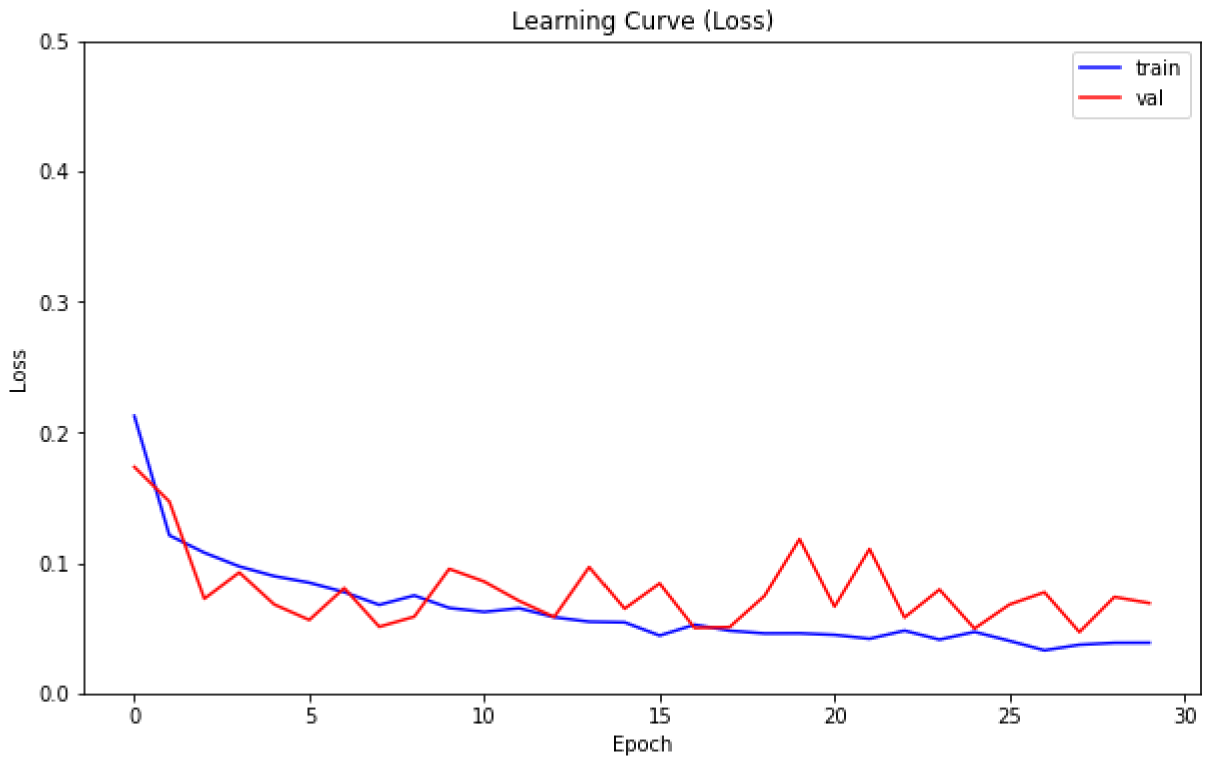
```
 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 17,926,209
Trainable params: 3,211,521
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
EPOCHS = 30
history = model_pretrained.fit(
    ds_train,
    validation_data=ds_val,
    epochs=EPOCHS,
    verbose=1
)
```

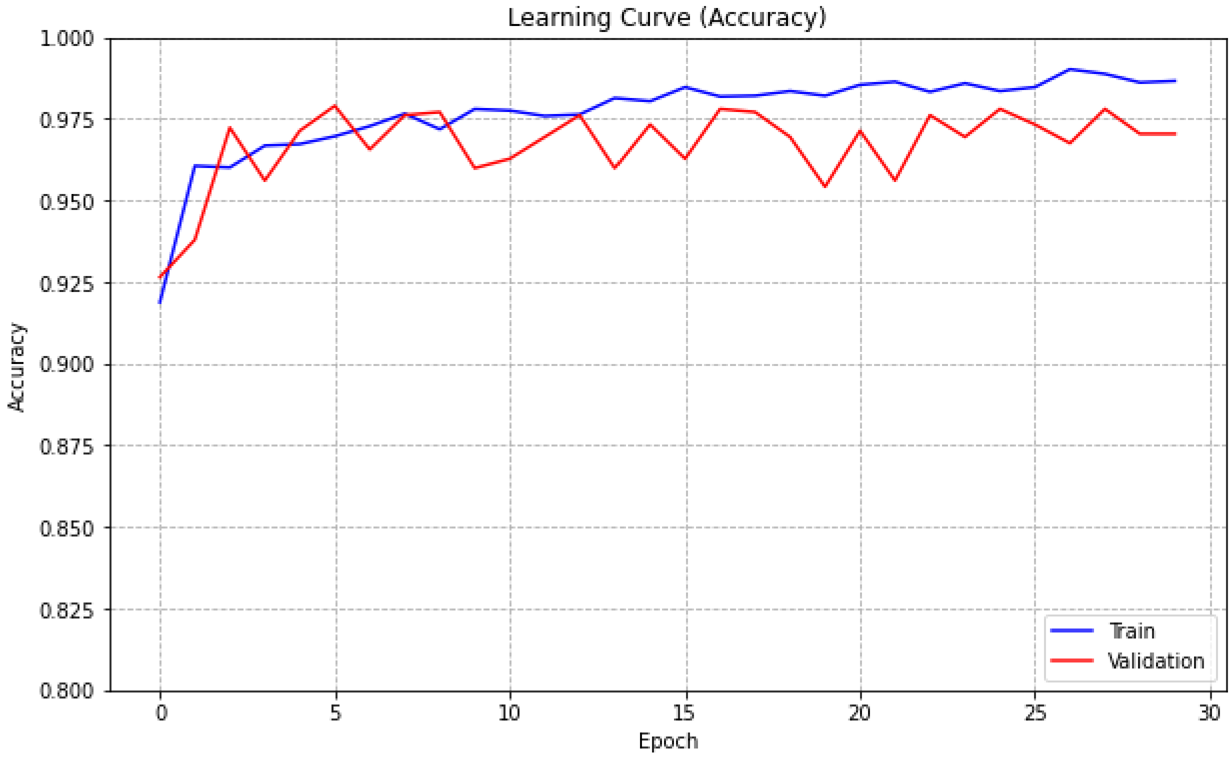
```
Epoch 2/30
131/131 [==============================] - 137s 1s/step - loss: 0.1212 - bina
Epoch 3/30
131/131 [==============================] - 140s 1s/step - loss: 0.1078 - bina
Epoch 4/30
131/131 [==============================] - 140s 1s/step - loss: 0.0973 - bina
Epoch 5/30
131/131 [==============================] - 139s 1s/step - loss: 0.0898 - bina
Epoch 6/30
131/131 [==============================] - 139s 1s/step - loss: 0.0848 - bina
Epoch 7/30
131/131 [==============================] - 139s 1s/step - loss: 0.0776 - bina
Epoch 8/30
131/131 [==============================] - 138s 1s/step - loss: 0.0678 - bina
Epoch 9/30
131/131 [==============================] - 137s 1s/step - loss: 0.0750 - bina
Epoch 10/30
131/131 [==============================] - 138s 1s/step - loss: 0.0654 - bina
Epoch 11/30
131/131 [==============================] - 138s 1s/step - loss: 0.0624 - bina
Epoch 12/30
131/131 [==============================] - 139s 1s/step - loss: 0.0652 - bina
Epoch 13/30
131/131 [==============================] - 137s 1s/step - loss: 0.0583 - bina
Epoch 14/30
131/131 [==============================] - 138s 1s/step - loss: 0.0549 - bina
Epoch 15/30
131/131 [==============================] - 137s 1s/step - loss: 0.0544 - bina
Epoch 16/30
131/131 [==============================] - 139s 1s/step - loss: 0.0442 - bina
Epoch 17/30
131/131 [==============================] - 137s 1s/step - loss: 0.0526 - bina
Epoch 18/30
131/131 [==============================] - 138s 1s/step - loss: 0.0480 - bina
Epoch 19/30
131/131 [==============================] - 139s 1s/step - loss: 0.0459 - bina
Epoch 20/30
131/131 [==============================] - 138s 1s/step - loss: 0.0459 - bina
Epoch 21/30
131/131 [==============================] - 138s 1s/step - loss: 0.0448 - bina
```

```
Epoch 23/30
131/131 [==============================] - 138s 1s/step - loss: 0.0480 - bina
Epoch 24/30
131/131 [==============================] - 140s 1s/step - loss: 0.0411 - bina
Epoch 25/30
131/131 [==============================] - 138s 1s/step - loss: 0.0471 - bina
Epoch 26/30
131/131 [==============================] - 139s 1s/step - loss: 0.0402 - bina
Epoch 27/30
131/131 [==============================] - 139s 1s/step - loss: 0.0329 - bina
Epoch 28/30
131/131 [==============================] - 138s 1s/step - loss: 0.0372 - bina
Epoch 29/30
131/131 [==============================] - 137s 1s/step - loss: 0.0386 - bina
Epoch 30/30
131/131 [==============================] - 139s 1s/step - loss: 0.0387 - bina
```

```
fig, ax = plt.subplots(figsize=(10,6))
sns.lineplot(x=history.epoch, y=history.history['loss'], color='blue')
sns.lineplot(x=history.epoch, y=history.history['val_loss'], color='red')
ax.set_title('Learning Curve (Loss)')
ax.set_ylabel('Loss')
ax.set_xlabel('Epoch')
ax.set_ylim(0, 0.5)
ax.legend(['train', 'val'], loc='best')
plt.show()
```



```
fig, ax = plt.subplots(figsize=(10, 6))
sns.lineplot(x=history.epoch, y=history.history['binary_accuracy'], color='blue',
sns.lineplot(x=history.epoch, y=history.history['val_binary_accuracy'], color='re
ax.set_title('Learning Curve (Accuracy)')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.set_ylim(0.80, 1.0)
ax.legend(loc='lower right')
plt.grid(True, linestyle='--')
plt.show()
```




## Performance Evaluation for VGG16

```
score = model_pretrained.evaluate(ds_val, steps = len(val_df)/BATCH, verbose = 1)
print('Val loss:', score[0])
print('Val accuracy:', score[1])
```

```
32/32 [==============================] - 19s 581ms/step - loss: 0.0691 - bina
Val loss: 0.06911761313676834
Val accuracy: 0.970391571521759
```

```
score = model_pretrained.evaluate(ds_test, steps = len(df_test), verbose = 1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
624/624 [==============================] - 18s 27ms/step - loss: 0.1632 - bin
Test loss: 0.16316260397434235
Test accuracy: 0.9471153616905212
```

## Fine Tuning

```
base_model.trainable = True

# Freeze all layers except for the
for layer in base_model.layers[:-13]:
    layer.trainable = False


for layer_number, layer in enumerate(base_model.layers):
    print(layer_number, layer.name, layer.trainable)
```

```
0 input_2 False
1 block1_conv1 False
2 block1_conv2 False
3 block1_pool False
4 block2_conv1 False
5 block2_conv2 False
6 block2_pool True
7 block3_conv1 True
8 block3_conv2 True
9 block3_conv3 True
10 block3_pool True
11 block4_conv1 True
12 block4_conv2 True
13 block4_conv3 True
14 block4_pool True
15 block5_conv1 True
16 block5_conv2 True
17 block5_conv3 True
18 block5_pool True
```

```
model_pretrained.compile(loss='binary_crossentropy', optimizer=keras.optimizers.A

model_pretrained.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 17,926,209
Trainable params: 17,666,049
```

```
Non-trainable params: 260,160
_________________________________________________________________
```

```
score = model_pretrained.evaluate(ds_val, steps = len(val_df)/BATCH, verbose = 1)
print('Val loss:', score[0])
print('Val accuracy:', score[1])
```

```
32/32 [==============================] - 21s 607ms/step - loss: 0.0691 - bina
Val loss: 0.06911761313676834
Val accuracy: 0.970391571521759
```

```
score = model_pretrained.evaluate(ds_test, steps = len(df_test), verbose = 1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
624/624 [==============================] - 17s 28ms/step - loss: 0.1632 - bin
Test loss: 0.16316260397434235
Test accuracy: 0.9471153616905212
```

## Performance Evaluation

```
# Load the test data and labels
num_label = {'Normal': 0, 'Pneumonia': 1}
Y_test = df_test['class'].copy().map(num_label).astype('int')

# Make predictions on the test data
ds_test.reset()
predictions = model_pretrained.predict(ds_test, steps=len(ds_test), verbose=0)
pred_labels = np.where(predictions>0.5, 1, 0)

# Compute the performance metrics
accuracy = metrics.accuracy_score(Y_test, pred_labels)
precision, recall, f1_score, _ = metrics.precision_recall_fscore_support(Y_test,
roc_auc = metrics.roc_auc_score(Y_test, predictions)

print('Accuracy: ', accuracy)
print('Precision: ', precision)
print('Recall: ', recall)
print('F1 score: ', f1_score)
print('ROC AUC: ', roc_auc)
```

```
Accuracy:  0.9471153846153846
Precision:  0.9301204819277108
Recall:  0.9897435897435898
F1 score:  0.9590062111801243
ROC AUC:  0.9881328073635766
```

```
# Evaluation metrics and corresponding colors
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 score', 'ROC AUC']
values = [0.9471153846153846, 0.9301204819277108, 0.9897435897435898, 0.959006211
colors = ['red', 'blue', 'green', 'orange', 'purple']

plt.bar(metrics, values, color=colors)

# Add axis labels and title
plt.xlabel('Evaluation Metric')
plt.ylabel('Value')
plt.title('Evaluation Metrics for VGG16 on Pneumonia Dataset')

# Display the plot
plt.show()
```