```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'chest-xray-pneumonia:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F17810%2F23812%2Fbundle%2Farchive.zip%3F

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading chest-xray-pneumonia, 2463365435 bytes compressed
[==================================================] 2463365435 bytes downloaded
```

```
Downloaded and uncompressed: chest-xray-pneumonia
Data source import complete.
```

AI for Medical Diagnosis

Computer Vision (CV) has a lot of applications in medical diagnosis:

- Dermatology
- Ophthakmology
- Histopathology.

X-rays images are critical for the detection of lung cancer, pneumenia ... In this notebook you will learn:

- Data pre-processing
- Preprocess images properly for the train, validation and test sets.
- Set-up a pre-trained neural network to make disease predictions on chest X-rays.

In this notebook you will work with chest X-ray images taken from the public ChestX-ray8 dataset.

What is Pneumonia ?

From Mayo Clinic's Article on pneumonia

Pneumonia is an infection that inflames the air sacs in one or both lungs. The air sacs may fill with fluid or pus (purulent material), causing cough with phlegm or pus, fever, chills, and difficulty breathing. A variety of organisms, including bacteria, viruses and fungi, can cause pneumonia.

Pneumonia can range in seriousness from mild to life-threatening. It is most serious for infants and young children, people older than age 65, and people with health problems or weakened immune systems.

Computer Vision

Computer vision is an interdisciplinary scientific field that deals with how computers can gain a high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do. We can use Computer Vision to determine whether a person is affected by pneumonia or not.

Pneumonia Detection with Convolutional Neural Networks

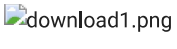
Computer Vision can be realized using Convolutional neural networks (CNN) They are neural networks making features extraction over an image before classifying it. The feature extraction performed consists of three basic operations:

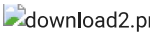
- Filter an image for a particular feature (convolution)
- Detect that feature within the filtered image (using the ReLU activation)
- Condense the image to enhance the features (maximum pooling)

The convolution process is illustrated below

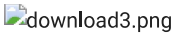
Using convolution filters with different dimensions or values results in differents features extracted

Features are then detected using the reLu activation on each destination pixel.



Features are the enhanced with MaxPool layers 

The stride parameters determines the distance between each filters. The padding one determines if we ignore the borderline pixels or not (adding zeros helps the neural network to get information on the border)



The outputs are then concatened in Dense layers

By using a sigmoid activation, the neural network determines which class the image belongs to 

Import Packages and Functions

We'll make use of the following packages:

- numpy and pandas is what we'll use to manipulate our data
- matplotlib.pyplot and seaborn will be used to produce plots for visualization
- util will provide the locally defined utility functions that have been provided for this assignment We will also use several modules from the keras framework for building deep learning models.

Run the next cell to import all the necessary packages.

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras

os.listdir("../input/chest-xray-pneumonia/chest_xray")
```

```
['test', 'chest_xray', 'val', 'train', '__MACOSX']
```

```
len(os.listdir("../input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA"))
```

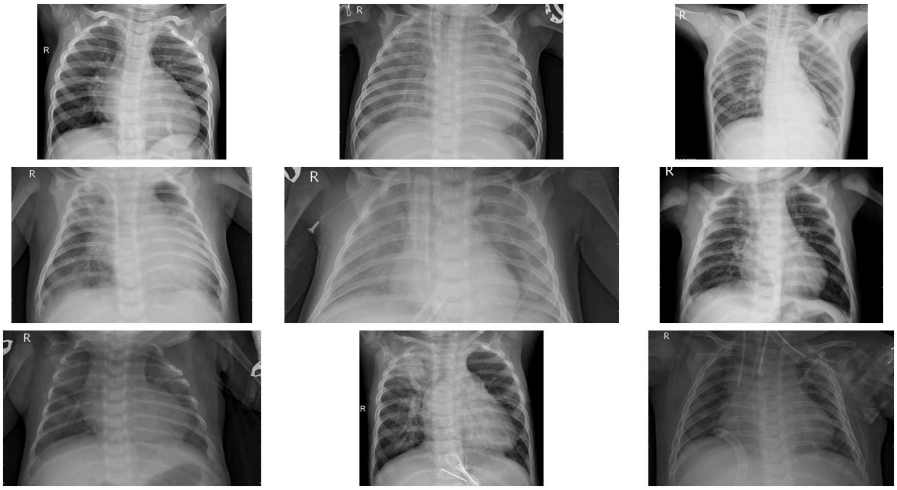
```
3875
```

The dataset is divided into three sets: 1) Train set 2) Validation set and 3) Test set.

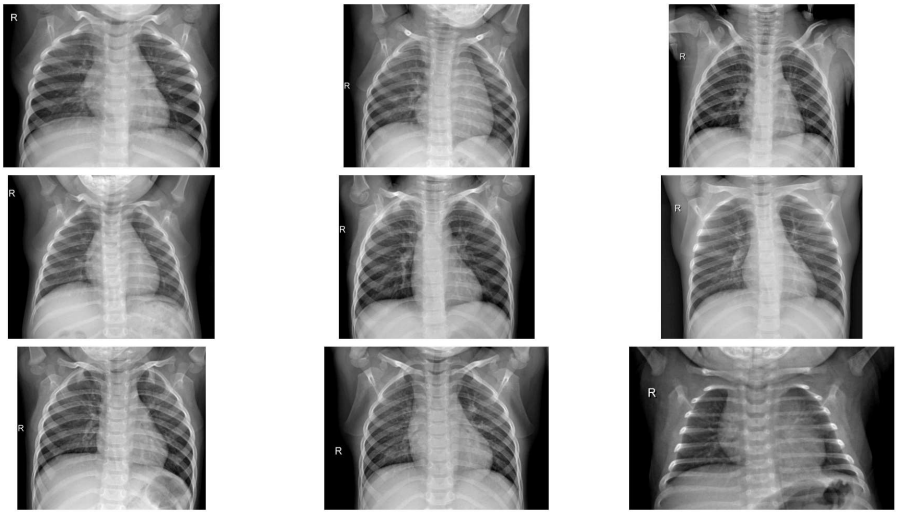
Data Visualization

```
train_dir = "../input/chest-xray-pneumonia/chest_xray/train"
test_dir = "../input/chest-xray-pneumonia/chest_xray/test"
val_dir = "../input/chest-xray-pneumonia/chest_xray/val"

print("Train set:\n========================================")
num_pneumonia = len(os.listdir(os.path.join(train_dir, 'PNEUMONIA')))
num_normal = len(os.listdir(os.path.join(train_dir, 'NORMAL')))
print(f"PNEUMONIA={num_pneumonia}")
print(f"NORMAL={num_normal}")

print("Test set:\n========================================")
print(f"PNEUMONIA={len(os.listdir(os.path.join(test_dir, 'PNEUMONIA')))}")
print(f"NORMAL={len(os.listdir(os.path.join(test_dir, 'NORMAL')))}")

print("Validation set:\n========================================")
print(f"PNEUMONIA={len(os.listdir(os.path.join(val_dir, 'PNEUMONIA')))}")
print(f"NORMAL={len(os.listdir(os.path.join(val_dir, 'NORMAL')))}")

pneumonia = os.listdir("../input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA")
pneumonia_dir = "../input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA"

plt.figure(figsize=(20, 10))

for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = plt.imread(os.path.join(pneumonia_dir, pneumonia[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.tight_layout()
```

```
Train set:
========================================
PNEUMONIA=3875
NORMAL=1341
Test set:
========================================
PNEUMONIA=390
NORMAL=234
Validation set:
========================================
PNEUMONIA=8
NORMAL=8
```

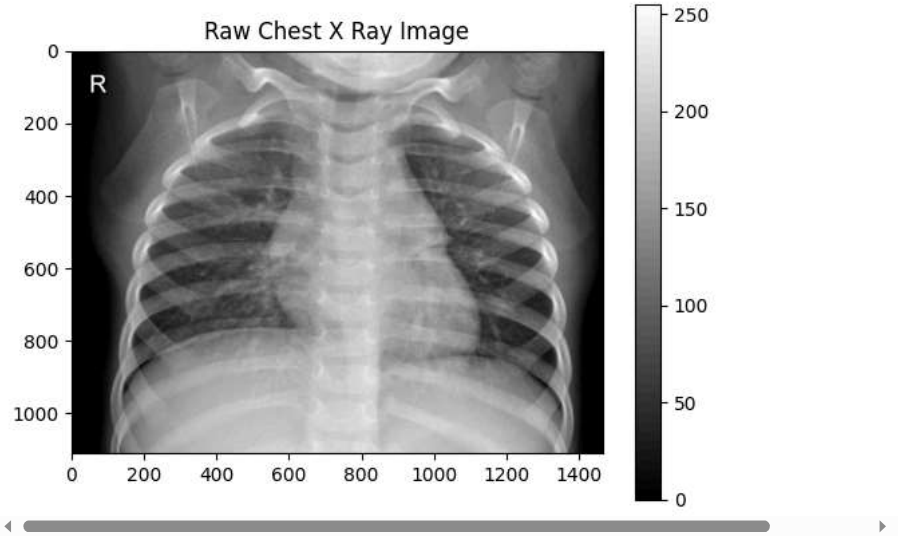


```
normal = os.listdir("../input/chest-xray-pneumonia/chest_xray/train/NORMAL")
normal_dir = "../input/chest-xray-pneumonia/chest_xray/train/NORMAL"

plt.figure(figsize=(20, 10))

for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = plt.imread(os.path.join(normal_dir, normal[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.tight_layout()
```



```
normal_img = os.listdir("../input/chest-xray-pneumonia/chest_xray/train/NORMAL")[0]
normal_dir = "../input/chest-xray-pneumonia/chest_xray/train/NORMAL"
sample_img = plt.imread(os.path.join(normal_dir, normal_img))
plt.imshow(sample_img, cmap='gray')
plt.colorbar()
plt.title('Raw Chest X Ray Image')

print(f"The dimensions of the image are {sample_img.shape[0]} pixels width and {sample_img.shape[1]} pixels height, one single color channel.
print(f"The maximum pixel value is {sample_img.max():.4f} and the minimum is {sample_img.min():.4f}")
print(f"The mean value of the pixels is {sample_img.mean():.4f} and the standard deviation is {sample_img.std():.4f}")
```

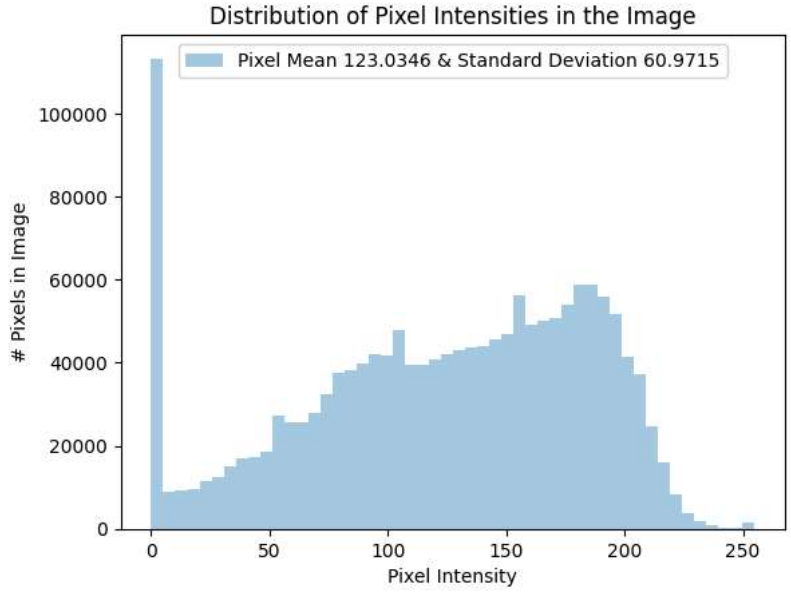
```
The dimensions of the image are 1109 pixels width and 1466 pixels height, one single col
The maximum pixel value is 255.0000 and the minimum is 0.0000
The mean value of the pixels is 123.0346 and the standard deviation is 60.9715
```



Ivestigate pixel value distribution

```
sns.distplot(sample_img.ravel(),
             label=f"Pixel Mean {np.mean(sample_img):.4f} & Standard Deviation {np.std(sample_img):.4f}", kde=False)
plt.legend(loc='upper center')
plt.title('Distribution of Pixel Intensities in the Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('# Pixels in Image')
```

```
<ipython-input-8-d45ac08d2db9>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sample_img.ravel(),
Text(0, 0.5, '# Pixels in Image')
```



2. Image Preprocessing

Before training, we'll first modify your images to be better suited for training a convolutional neural network. For this task we'll use the Keras ImageDataGenerator function to perform data preprocessing and data augmentation.

This class also provides support for basic data augmentation such as random horizontal flipping of images. We also use the generator to transform the values in each batch so that their mean is 0 and their standard deviation is 1 (this will faciliate model training by standardizing the input distribution). The generator also converts our single channel X-ray images (gray-scale) to a three-channel format by repeating the values in the image across all channels (we will want this because the pre-trained model that we'll use requires three-channel inputs).

```
from keras.preprocessing.image import ImageDataGenerator

image_generator = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    samplewise_center=True,
    samplewise_std_normalization=True
)
```

Build a separate generator fo valid and test sets

Now we need to build a new generator for validation and t esting data.

Why can't use the same generator as for the training data?

Look back at the generator we wrote for the training data.

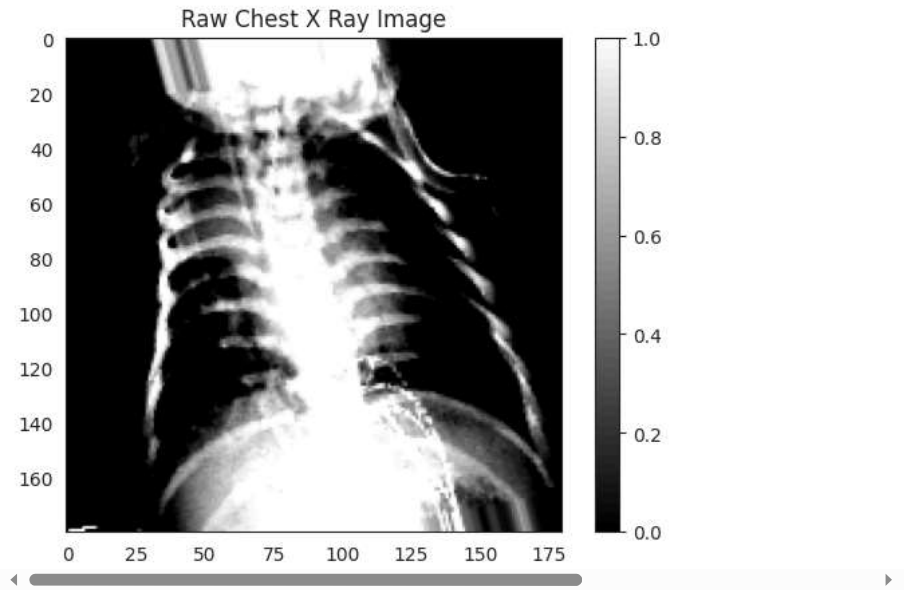
It normalizes each image per batch, meaning thatit uses batch statistics. We should not do this with the test and validation data, since in a real life scenario we don't process incoming images a batch at a time (we process one image at a time). Knowing the average per batch of test data would effectively give our model an advantage (The model should not have any information about the test data). What we need to do is to normalize incomming test data using the statistics computed from the training set.

```
train = image_generator.flow_from_directory(train_dir,
                                            batch_size=8,
                                            shuffle=True,
                                            class_mode='binary',
                                            target_size=(180, 180))

validation = image_generator.flow_from_directory(val_dir,
                                                 batch_size=1,
                                                 shuffle=False,
                                                 class_mode='binary',
                                                 target_size=(180, 180))

test = image_generator.flow_from_directory(test_dir,
                                           batch_size=1,
                                           shuffle=False,
                                           class_mode='binary',
                                           target_size=(180, 180))
```

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

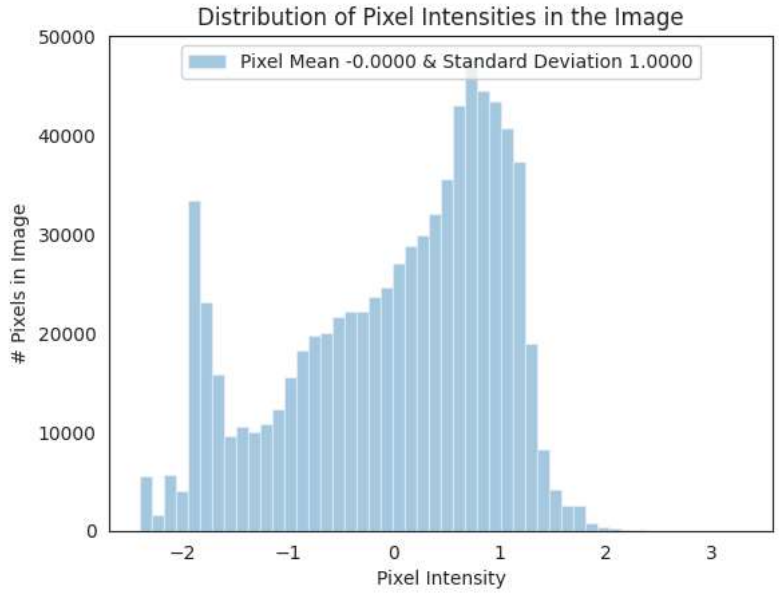
```
sns.set_style('white')
generated_image, label = train.__getitem__(0)
plt.imshow(generated_image[0], cmap='gray')
plt.colorbar()
plt.title('Raw Chest X Ray Image')

print(f"The dimensions of the image are {generated_image.shape[1]} pixels width and {generated_image.shape[2]} pixels height, one single col
print(f"The maximum pixel value is {generated_image.max():.4f} and the minimum is {generated_image.min():.4f}")
print(f"The mean value of the pixels is {generated_image.mean():.4f} and the standard deviation is {generated_image.std():.4f}")
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data
The dimensions of the image are 180 pixels width and 180 pixels height, one single color
The maximum pixel value is 3.2975 and the minimum is -2.4050
The mean value of the pixels is -0.0000 and the standard deviation is 1.0000
```



```
sns.distplot(generated_image.ravel(),
             label=f"Pixel Mean {np.mean(generated_image):.4f} & Standard Deviation {np.std(generated_image):.4f}", kde=False)
plt.legend(loc='upper center')
plt.title('Distribution of Pixel Intensities in the Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('# Pixels in Image')
```

```
<ipython-input-12-3361fab08a7d>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(generated_image.ravel(),
Text(0, 0.5, '# Pixels in Image')
```



Building a CNN model

Impact of imbalance data on loss function

Loss Function:

$$\mathcal{L}_{cross-entropy}(x_i) = -(y_i \log(f(x_i)) + (1 - y_i) \log(1 - f(x_i))),$$

We can rewrite the the overall average cross-entropy loss over the entire training set `D` of size `N` as follows:

$$\mathcal{L}_{cross-entropy}(\mathcal{D}) = -\frac{1}{N}\Big(\sum_{\text{positive examples}} \log(f(x_i)) + \sum_{\text{negative examples}} \log(1 - f(x_i))\Big).$$

When we have an imbalance data, using a normal loss function will result a model that bias toward the dominating class. One solution is to use a weighted loss function. Using weighted loss function will balance the contribution in the loss function.

$$\mathcal{L}^{w}_{cross-entropy}(x) = -(w_p y \log(f(x)) + w_n (1 - y) \log(1 - f(x))).$$

```
# Class weights

weight_for_0 = num_pneumonia / (num_normal + num_pneumonia)
weight_for_1 = num_normal / (num_normal + num_pneumonia)

class_weight = {0: weight_for_0, 1: weight_for_1}

print(f"Weight for class 0: {weight_for_0:.2f}")
print(f"Weight for class 1: {weight_for_1:.2f}")
```

```
Weight for class 0: 0.74
Weight for class 1: 0.26
```

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Dropout, Flatten, BatchNormalization


model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(180, 180, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(180, 180, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

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


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 178, 178, 32)      896

 batch_normalization (Batch  (None, 178, 178, 32)      128
 Normalization)

 conv2d_1 (Conv2D)           (None, 176, 176, 32)      9248

 batch_normalization_1 (Bat  (None, 176, 176, 32)      128
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 88, 88, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 86, 86, 64)        18496

 batch_normalization_2 (Bat  (None, 86, 86, 64)        256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 84, 84, 64)        36928

 batch_normalization_3 (Bat  (None, 84, 84, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 42, 42, 64)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 40, 40, 128)       73856

 batch_normalization_4 (Bat  (None, 40, 40, 128)       512
 chNormalization)

 conv2d_5 (Conv2D)           (None, 38, 38, 128)       147584

 batch_normalization_5 (Bat  (None, 38, 38, 128)       512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 19, 19, 128)       0
 g2D)

 flatten (Flatten)           (None, 46208)             0
```

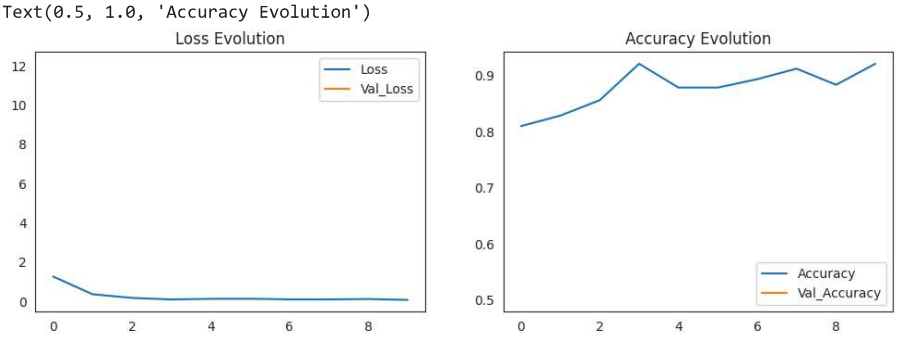
```
 dense (Dense)               (None, 128)               5914752

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 6203681 (23.67 MB)
Trainable params: 6202785 (23.66 MB)
Non-trainable params: 896 (3.50 KB)
_________________________________________________________________
```

```
r = model.fit(
    train,
    epochs=10,
    validation_data=validation,
    class_weight=class_weight,
    steps_per_epoch=100,
    validation_steps=25,
)
```

```
Epoch 1/10
100/100 [==============================] - ETA: 0s - loss: 1.2590 - accuracy: 0.8100WARNING:tensorflow:Your input ran out of data; inter
100/100 [==============================] - 200s 2s/step - loss: 1.2590 - accuracy: 0.8100 - val_loss: 12.1107 - val_accuracy: 0.5000
Epoch 2/10
100/100 [==============================] - 191s 2s/step - loss: 0.3574 - accuracy: 0.8288
Epoch 3/10
100/100 [==============================] - 186s 2s/step - loss: 0.1711 - accuracy: 0.8562
Epoch 4/10
100/100 [==============================] - 185s 2s/step - loss: 0.0922 - accuracy: 0.9212
Epoch 5/10
100/100 [==============================] - 184s 2s/step - loss: 0.1239 - accuracy: 0.8788
Epoch 6/10
100/100 [==============================] - 189s 2s/step - loss: 0.1285 - accuracy: 0.8788
Epoch 7/10
100/100 [==============================] - 183s 2s/step - loss: 0.0963 - accuracy: 0.8938
Epoch 8/10
100/100 [==============================] - 182s 2s/step - loss: 0.0944 - accuracy: 0.9125
Epoch 9/10
100/100 [==============================] - 186s 2s/step - loss: 0.1138 - accuracy: 0.8838
Epoch 10/10
100/100 [==============================] - 182s 2s/step - loss: 0.0690 - accuracy: 0.9212
```

```
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='Val_Loss')
plt.legend()
plt.title('Loss Evolution')

plt.subplot(2, 2, 2)
plt.plot(r.history['accuracy'], label='Accuracy')
plt.plot(r.history['val_accuracy'], label='Val_Accuracy')
plt.legend()
plt.title('Accuracy Evolution')
```

```
Text(0.5, 1.0, 'Accuracy Evolution')
```



```
evaluation = model.evaluate(test)
print(f"Test Accuracy: {evaluation[1] * 100:.2f}%")

evaluation = model.evaluate(train)
print(f"Train Accuracy: {evaluation[1] * 100:.2f}%")
```

```
624/624 [==============================] - 80s 121ms/step - loss: 0.9403 - accuracy: 0.4423
Test Accuracy: 44.23%
652/652 [==============================] - 539s 826ms/step - loss: 1.1077 - accuracy: 0.3296
Train Accuracy: 32.96%
```

```
from sklearn.metrics import confusion_matrix, classification_report

pred = model.predict(test)

print(confusion_matrix(test.classes, pred > 0.5))
pd.DataFrame(classification_report(test.classes, pred > 0.5, output_dict=True))
```

```
624/624 [==============================] - 79s 123ms/step
[[229   5]
 [345  45]]
```

	0	1	accuracy	macro avg	weighted avg
precision	0.398955	0.900000	0.439103	0.649477	0.712108
recall	0.978632	0.115385	0.439103	0.547009	0.439103
f1-score	0.566832	0.204545	0.439103	0.385689	0.340403
support	234.000000	390.000000	0.439103	624.000000	624.000000

```
print(confusion_matrix(test.classes, pred > 0.7))
pd.DataFrame(classification_report(test.classes, pred > 0.7, output_dict=True))
```

Transfer Learning

DenseNet

Densenet is a convolutional network where each layer is connected to all other layers that are deeper in the network:

- The first layer is connected to the 2nd, 3rd, 4th etc.
- The second layer is conected to the 3rd, 4th, 5th etc.

for more information about the DenseNet Architecture visit this website : https://keras.io/api/applications/densenet/

```
from keras.applications.densenet import DenseNet121
from keras.layers import Dense, GlobalAveragePooling2D
from keras.models import Model
from keras import backend as K

base_model = DenseNet121(input_shape=(180, 180, 3), include_top=False, weights='imagenet', pooling='avg')

base_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_ker
29084464/29084464 [==============================] - 0s 0us/step
Model: "densenet121"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0         []

 zero_padding2d (ZeroPaddin  (None, 186, 186, 3)       0         ['input_1[0][0]']
 g2D)

 conv1/conv (Conv2D)         (None, 90, 90, 64)        9408      ['zero_padding2d[0][0]']

 conv1/bn (BatchNormalizati  (None, 90, 90, 64)        256       ['conv1/conv[0][0]']
 on)

 conv1/relu (Activation)     (None, 90, 90, 64)        0         ['conv1/bn[0][0]']
```

```
zero_padding2d_1 (ZeroPadd  (None, 92, 92, 64)          0         ['conv1/relu[0][0]']
ing2D)

pool1 (MaxPooling2D)        (None, 45, 45, 64)          0         ['zero_padding2d_1[0][0]']

conv2_block1_0_bn (BatchNo  (None, 45, 45, 64)          256       ['pool1[0][0]']
rmalization)

conv2_block1_0_relu (Activ  (None, 45, 45, 64)          0         ['conv2_block1_0_bn[0][0]']
ation)

conv2_block1_1_conv (Conv2  (None, 45, 45, 128)         8192      ['conv2_block1_0_relu[0][0]']
D)

conv2_block1_1_bn (BatchNo  (None, 45, 45, 128)         512       ['conv2_block1_1_conv[0][0]']
rmalization)

conv2_block1_1_relu (Activ  (None, 45, 45, 128)         0         ['conv2_block1_1_bn[0][0]']
ation)

conv2_block1_2_conv (Conv2  (None, 45, 45, 32)          36864     ['conv2_block1_1_relu[0][0]']
D)

conv2_block1_concat (Conca  (None, 45, 45, 96)          0         ['pool1[0][0]',
tenate)                                                          'conv2_block1_2_conv[0][0]']

conv2_block2_0_bn (BatchNo  (None, 45, 45, 96)          384       ['conv2_block1_concat[0][0]']
rmalization)

conv2_block2_0_relu (Activ  (None, 45, 45, 96)          0         ['conv2_block2_0_bn[0][0]']
ation)

conv2_block2_1_conv (Conv2  (None, 45, 45, 128)         12288     ['conv2_block2_0_relu[0][0]']
D)

conv2_block2_1_bn (BatchNo  (None, 45, 45, 128)         512       ['conv2_block2_1_conv[0][0]']
rmalization)

conv2_block2_1_relu (Activ  (None, 45, 45, 128)         0         ['conv2_block2_1_bn[0][0]']
```

```
layers = base_model.layers
print(f"The model has {len(layers)} layers")
```

```
The model has 427 layers
```

```
print(f"The input shape {base_model.input}")
print(f"The output shape {base_model.output}")
```

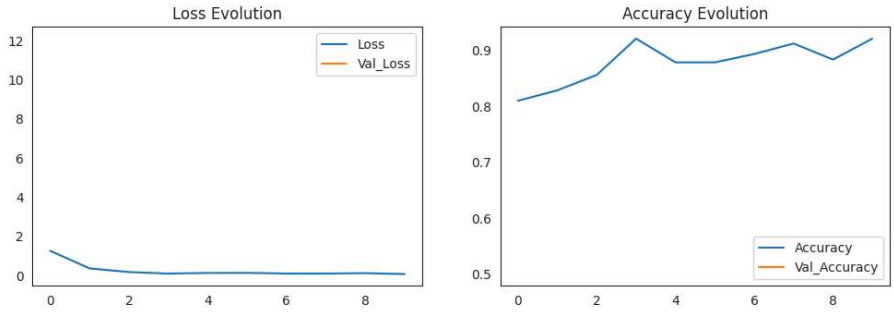
```
The input shape KerasTensor(type_spec=TensorSpec(shape=(None, None, None, 3), dtype=tf.float32, name='input_3'), name='input_3', descrip
The output shape KerasTensor(type_spec=TensorSpec(shape=(None, None, None, 1024), dtype=tf.float32, name=None), name='relu/Relu:0', desc
```

```
#model = Sequential()
base_model = DenseNet121(include_top=False, weights='imagenet')
x = base_model.output

x = GlobalAveragePooling2D()(x)

predictions = Dense(1, activation="sigmoid")(x)

model = Model(inputs=base_model.input, outputs=predictions)
#model.add(base_model)
#model.add(GlobalAveragePooling2D())
#model.add(Dense(1, activation='sigmoid'))

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


r = model.fit(
    train,
    epochs=10,
    validation_data=validation,
    class_weight=class_weight,
    steps_per_epoch=100,
    validation_steps=25,
)
```

```
Epoch 1/10
100/100 [==============================] - ETA: 0s - loss: 0.1856 - accuracy: 0.7912WARNING:tensorflow:Your input ran out of data; inter
100/100 [==============================] - 391s 3s/step - loss: 0.1856 - accuracy: 0.7912 - val_loss: 2.0945 - val_accuracy: 0.5625
Epoch 2/10
100/100 [==============================] - 333s 3s/step - loss: 0.1110 - accuracy: 0.8838
Epoch 3/10
100/100 [==============================] - 333s 3s/step - loss: 0.1080 - accuracy: 0.8687
Epoch 4/10
100/100 [==============================] - 334s 3s/step - loss: 0.0914 - accuracy: 0.8963
Epoch 5/10
100/100 [==============================] - 334s 3s/step - loss: 0.0951 - accuracy: 0.8963
Epoch 6/10
100/100 [==============================] - 334s 3s/step - loss: 0.0835 - accuracy: 0.9075
Epoch 7/10
100/100 [==============================] - 335s 3s/step - loss: 0.1086 - accuracy: 0.8775
Epoch 8/10
100/100 [==============================] - 334s 3s/step - loss: 0.0806 - accuracy: 0.9162
Epoch 9/10
100/100 [==============================] - 335s 3s/step - loss: 0.0839 - accuracy: 0.9100
Epoch 10/10
100/100 [==============================] - 332s 3s/step - loss: 0.0818 - accuracy: 0.8950
```

```
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='Val_Loss')
plt.legend()
plt.title('Loss Evolution')

plt.subplot(2, 2, 2)
plt.plot(r.history['accuracy'], label='Accuracy')
plt.plot(r.history['val_accuracy'], label='Val_Accuracy')
plt.legend()
plt.title('Accuracy Evolution')
```

```
Text(0.5, 1.0, 'Accuracy Evolution')
```




```
evaluation = model.evaluate(test)
print(f"Test Accuracy: {evaluation[1] * 100:.2f}%")

evaluation = model.evaluate(train)
print(f"Train Accuracy: {evaluation[1] * 100:.2f}%")
```

```
624/624 [==============================] - 90s 136ms/step - loss: 0.7473 - accuracy: 0.5689
Test Accuracy: 56.89%
652/652 [==============================] - 545s 836ms/step - loss: 0.6638 - accuracy: 0.6447
Train Accuracy: 64.47%
```

Evaluation

```
predicted_vals = model.predict(test, steps=len(test))
```

```
624/624 [==============================] - 79s 124ms/step
```

```
print(confusion_matrix(test.classes, predicted_vals > 0.5))
pd.DataFrame(classification_report(test.classes, predicted_vals > 0.5, output_dict=True))
```

```
[[ 40 194]
 [ 77 313]]
```

	0	1	accuracy	macro avg	weighted avg
precision	0.34188	0.617357	0.565705	0.479619	0.514053
recall	0.17094	0.802564	0.565705	0.486752	0.565705
f1-score	0.22792	0.697882	0.565705	0.462901	0.521646
support	234.00000	390.000000	0.565705	624.000000	624.000000

VGG16

Presented in 2014, VGG16 has a very simple and classical architecture, with blocks of 2 or 3 convolutional layers followed by a pooling layer, plus a final dense network composed of 2 hidden layers (of 4096 nodes each) and one output layer (of 1000 nodes). Only 3x3 filters are used.


```
from keras.models import Sequential
from keras.layers import GlobalAveragePooling2D
from keras.applications import VGG16


vgg16_base_model = VGG16(input_shape=(180,180,3),include_top=False,weights='imagenet')
```

```
vgg16_base_model.summary()
```

```
    vgg16_model = tf.keras.Sequential([
        vgg16_base_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])


    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    METRICS = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
    vgg16_model.compile(optimizer=opt,loss='binary_crossentropy',metrics=METRICS)


r = vgg16_model.fit(train,
          epochs=10,
          validation_data=validation,
          class_weight=class_weight,
          steps_per_epoch=100,
          validation_steps=25)
```

```
Epoch 1/10
100/100 [==============================] - ETA: 0s - loss: 0.2622 - accuracy: 0.6275 - precision: 0.8750 - recall: 0.5853 WARNING:tensor
100/100 [==============================] - 1029s 10s/step - loss: 0.2622 - accuracy: 0.6275 - precision: 0.8750 - recall: 0.5853 - val_l
```

```
Epoch 2/10
100/100 [==============================] - 1012s 10s/step - loss: 0.2163 - accuracy: 0.6988 - precision: 0.9007 - recall: 0.6695
Epoch 3/10
100/100 [==============================] - 1003s 10s/step - loss: 0.1924 - accuracy: 0.7475 - precision: 0.9329 - recall: 0.7159
Epoch 4/10
100/100 [==============================] - 1064s 11s/step - loss: 0.1921 - accuracy: 0.7725 - precision: 0.9087 - recall: 0.7710
Epoch 5/10
100/100 [==============================] - 1085s 11s/step - loss: 0.1962 - accuracy: 0.7563 - precision: 0.9102 - recall: 0.7471
Epoch 6/10
100/100 [==============================] - 1081s 11s/step - loss: 0.1736 - accuracy: 0.7650 - precision: 0.9468 - recall: 0.7125
Epoch 7/10
100/100 [==============================] - 1078s 11s/step - loss: 0.1733 - accuracy: 0.7825 - precision: 0.9415 - recall: 0.7631
Epoch 8/10
100/100 [==============================] - 1080s 11s/step - loss: 0.1590 - accuracy: 0.8012 - precision: 0.9528 - recall: 0.7733
Epoch 9/10
 54/100 [===============>.............] - ETA: 8:15 - loss: 0.1763 - accuracy: 0.7847 - precision: 0.9219 - recall: 0.7638
```

```
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='Val_Loss')
plt.legend()
plt.title('Loss Evolution')

plt.subplot(2, 2, 2)
plt.plot(r.history['accuracy'], label='Accuracy')
plt.plot(r.history['val_accuracy'], label='Val_Accuracy')
plt.legend()
plt.title('Accuracy Evolution')
```

```
evaluation =vgg16_model.evaluate(test)
print(f"Test Accuracy: {evaluation[1] * 100:.2f}%")

evaluation = vgg16_model.evaluate(train)
print(f"Train Accuracy: {evaluation[1] * 100:.2f}%")
```

ResNet

See the full explanation and schemes in the Research Paper on Deep Residual Learning (https://arxiv.org/pdf/1512.03385.pdf)

```
from keras.applications import ResNet50

resnet_base_model = ResNet50(input_shape=(180,180,3), include_top=False, weights='imagenet')
```

```
resnet_base_model.summary()
```

```
    resnet_model = tf.keras.Sequential([
        resnet_base_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])

    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    METRICS = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
    resnet_model.compile(optimizer=opt,loss='binary_crossentropy',metrics=METRICS)
```

```
r = resnet_model.fit(train,
          epochs=10,
          validation_data=validation,
          class_weight=class_weight,
          steps_per_epoch=100,
          validation_steps=25)



plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='Val_Loss')
plt.legend()
plt.title('Loss Evolution')

plt.subplot(2, 2, 2)
plt.plot(r.history['accuracy'], label='Accuracy')
plt.plot(r.history['val_accuracy'], label='Val_Accuracy')
plt.legend()
plt.title('Accuracy Evolution')


evaluation =resnet_model.evaluate(test)
print(f"Test Accuracy: {evaluation[1] * 100:.2f}%")

evaluation = resnet_model.evaluate(train)
print(f"Train Accuracy: {evaluation[1] * 100:.2f}%")
```

InceptionNet

Also known as GoogleNet, this architecture presents sub-networks called inception modules, which allows fast training computing, complex patterns detection, and optimal use of parameters

for more information visit https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43022.pdf

```
from keras.applications import InceptionV3

inception_base_model = InceptionV3(input_shape=(180,180,3),include_top=False,weights='imagenet')
```

```
    inception_model = tf.keras.Sequential([
        inception_base_model,
        GlobalAveragePooling2D(),
        Dense(512, activation="relu"),
        BatchNormalization(),
        Dropout(0.6),
        Dense(128, activation="relu"),
        BatchNormalization(),
        Dropout(0.4),
        Dense(64,activation="relu"),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1,activation="sigmoid")
    ])

    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    METRICS = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
    inception_model.compile(optimizer=opt,loss='binary_crossentropy',metrics=METRICS)
```

```
r = inception_model.fit(train,
          epochs=10,
          validation_data=validation,
          class_weight=class_weight,
          steps_per_epoch=100,
          validation_steps=25)



plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='Val_Loss')
plt.legend()
plt.title('Loss Evolution')
```