Research Draft – Cancer Detection With AI- processing Documentation

Redback Operations

Algorithm Team - Shashvat Joshi

Introduction

Artificial Intelligence, Machine Learning, Deep Learning are disrupting almost every industry. AI/Data science is also revolutionizing the healthcare, wellness, fitness industry. Over the decades we have been acknowledging the fact "Health is wealth". However, the Covid pandemic taught us the universal truth a hard way. We recognized the importance of every aspect of a healthy lifestyle, awareness, and increased focus of science and technology in healthcare.

The role of Data analysis, Data Science, AI is expanding fast in sports, health & fitness programs.

In recent years the image processing mechanisms are used widely in several medical areas. Cancer cure depends on early detection and treatment stages, in which the time factor is very important. If doctors can discover the disease in the patient as possible as fast. The use of AI in medical science has been attracting the attention of medical and healthcare industry in the latest years because of its high prevalence allied with the difficult treatment.

Detection of Cancer often involves radiological imaging. Radiological Imaging is used to check the spread of cancer and progress of treatment. It is also used to monitor cancer. Oncological imaging is continually becoming more varied and accurate. Different imaging techniques aim to find the most suitable treatment option for each patient. Imaging techniques are often used in combination to obtain sufficient information.

Objective

According to the ACS Guidelines on Nutrition and Physical Activity for Cancer Prevention, getting more physical activity is associated with a lower risk for several types of cancer, including breast, prostate, colon, endometrium, and possibly pancreatic cancer.

This project is aimed at developing a model capable of detecting cancer and in future to be integrated in games project, so the users who our part of our Open exercise community can utilize this system in our games to detect cancer or reduce risk for several types of cancer via means of exercise.

Document History

Task Name	Date added		
Initial Version	20-July-2022	Abstract, Introduction, Dataset details, Data Understanding	
V2	3-Aug-2022	Proposed Use Case Propose Solution Deep Learning Model Methodology CNN – Model selection CNN Architecture study Load Sample Cancer Image Data Visualize Image Data	
V3	17-Aug-2022	Researching other dataset – lung cancer images	
V4	7-Sep-2022	Lung Cancer images - Data Processing & Model development and build first model with CNN.	
V5	21-Sep-2022	Data Augmentation Techniques How to improve accuracy of the CNN model.	

Research

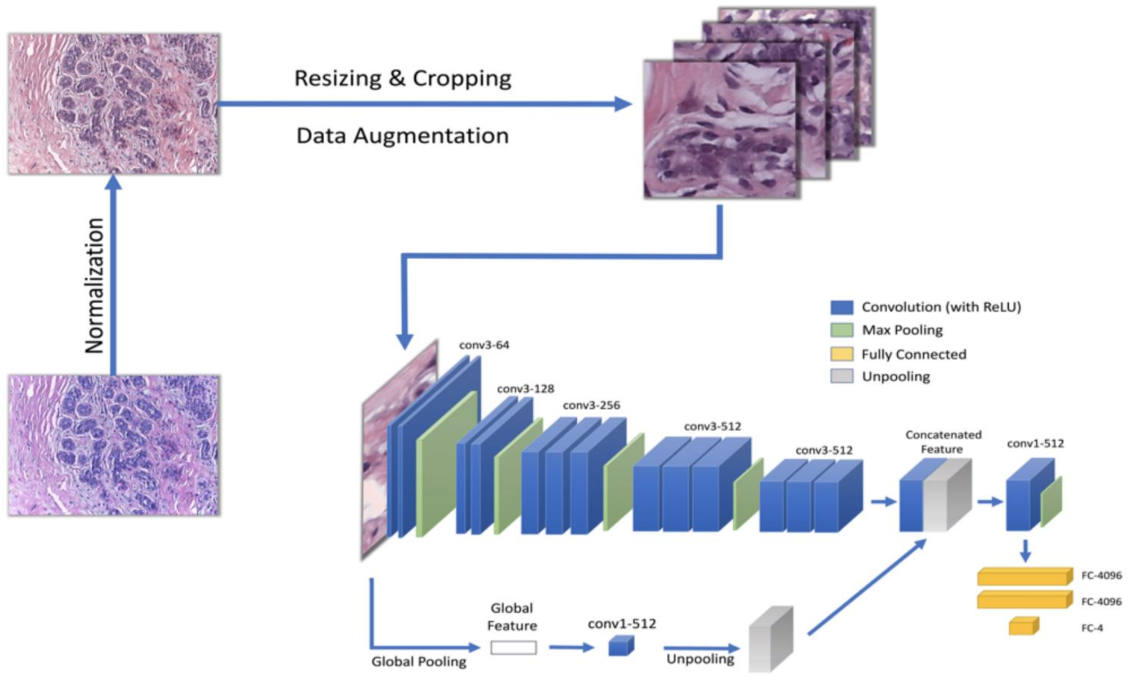


Fig. 1: *Classification of Histopathological images for early Detection of Cancer using Deep Learning*

In this submission we will study the various kind of images used in medical study. The image data quality, type of images, volume of images are important for the project success.

We will research on different algorithms used for detection and classification of cancer which include steps such as Image Pre-Processing, Image segmentation, Feature Extraction and Classification.

The Image Preprocessing techniques discussed are Median Filter, Gaussian Filter and Contrast Enhancement technique. The various techniques used in several study of AI driven cancer detection projects are Image Segmentation are K-means algorithm, Otsu thresholding, Edge Detection.

Feature Extraction is based on ABCD Rule of Dermatoscopy, Texture, etc.

The classifiers will be evaluated and discussed in the next few weeks are Support Vector Machine (SVM), Neural Network, K-NN Algorithm and Random Forest. We will also explore the Deep Learning models like CNN, RESNET etc. Several proposed algorithms and their evaluation criteria, performance will be compared.

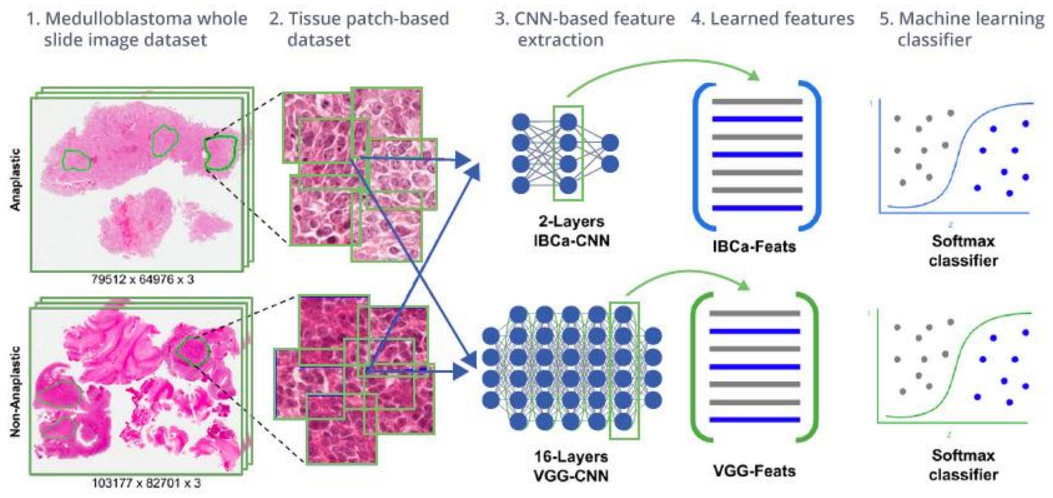


Fig. 2: *CNN -based transfer learning medulloblastoma tumor differentiation [2]*

Proposed Use Case

We propose to build a CNN based model which can accurately detect cancer.

For this project we selected publicly available cancer images which are available for multiple cancer types. The sample data is having nine different classes of cancers.

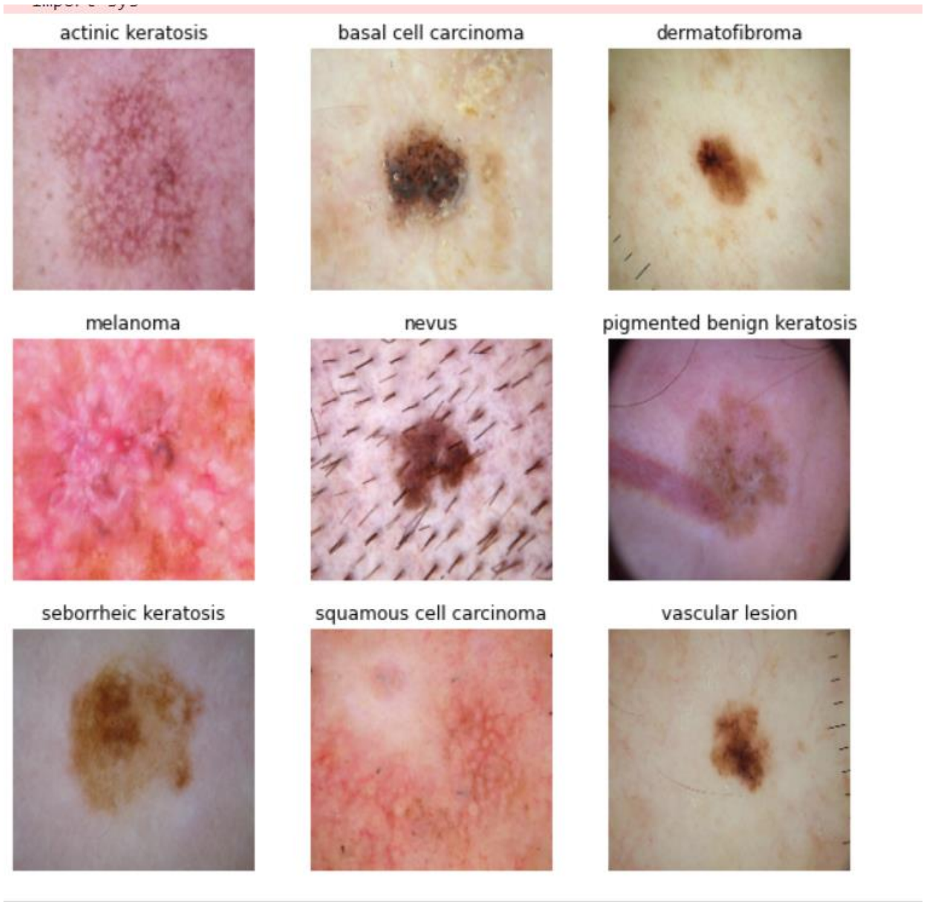


Fig. 3: *Types of cancer*

The **melanoma** is a type of cancer that can be deadly if not detected early. It accounts for 75% of cancer deaths.

Proposed solution

A Deep Learning AI solution which can evaluate images and alert the users about the presence of cancer cell (here testing on melanoma) has the potential to reduce a lot of manual effort needed in diagnosis.

We would like to highlight the similar model development approach can be used to detect different cancer types also. However, for different images different image processing, clean up techniques will be used.

Model architecture will also be different depending image quality and image volumes.

Methodology

- We will load the data by kera.preprocessing and build train_ds. The images will be labeled using the directry/folder structure.
- The number of images are less here.
- We will first run the base model.
- If the accuracy is less and loss factor is higher, we have to apply the augmentation technique to increase the number of images.
- Hence the next the model will go through couple of iterations to achieve the good accuracy.
- For model building we will use CNN – Deep Learning Architecture.

- The CNN algorithm needs GPU infrastructure to tarin the model. Hence model is running in google colab with GPU instance.

Convolutional Neural Network (CNN)

Convolutional Neural Networks, or CNNs, are specialized architectures which work particularly well with visual data, i.e. images and videos. They have been largely responsible for revolutionizing 'deep learning' by setting new benchmarks for many images processing tasks that were very recently considered extremely hard. Although the vanilla neural networks (MLPs) can learn extremely complex functions, their architecture does not exploit what we know about how the brain reads and processes images. For this reason, although MLPs are successful in solving many complex problems, they haven't been able to achieve any major breakthroughs in the image processing domain.

CNN Architecture

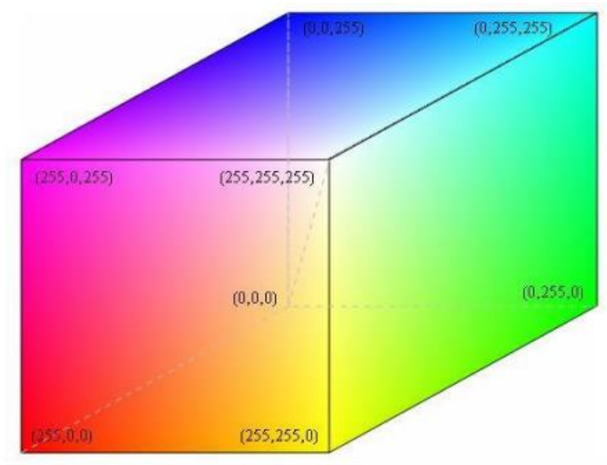


Figure 4: Pixel values in colour image

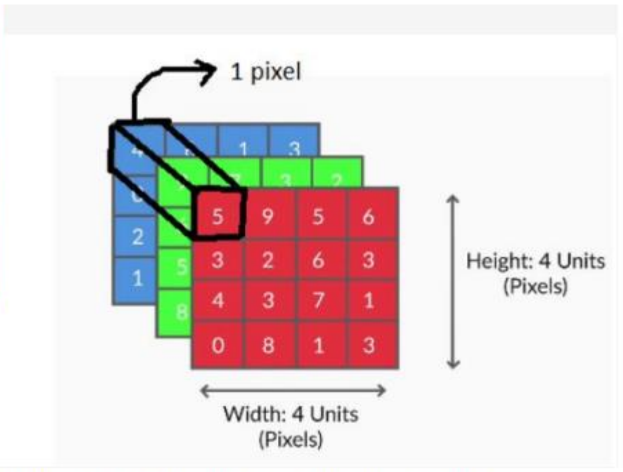


Figure 5: Color image represented as matrix

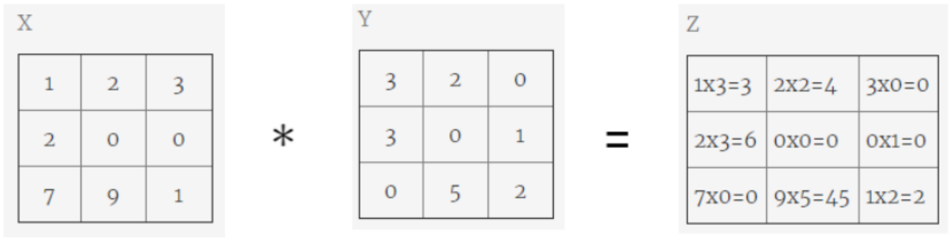
***Fig: 4 & 5** - Represents the color image of size 4x4x3, where height is 4, width is 4, the number of channels is 3(RGB). The number of pixels is 16 (height x width).*

Main concepts in CNNs:

- Convolution, and why it 'shrinks' the size of the input image
- Stride and Padding
- Pooling layers
- Feature maps

Convolution

Mathematically, the convolution operation is the summation of the element-wise product of two matrices. Let's take two matrices, X and Y. If you 'convolve the image X using the filter Y', this operation will produce the matrix Z. Let's say when we have X and Y of the same dimension.



X

1	2	3
2	0	0
7	9	1

*

Y

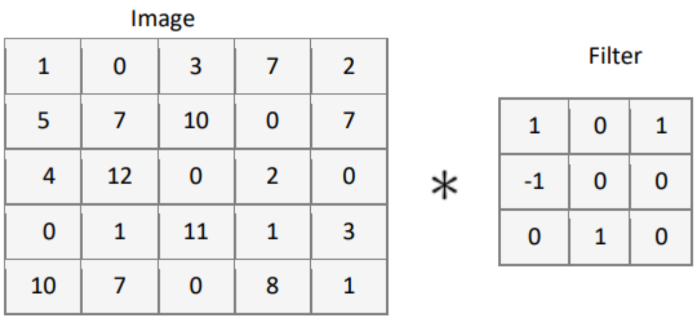
3	2	0
3	0	1
0	5	2

=

Z

1X3=3	2X2=4	3X0=0
2X3=6	0X0=0	0X1=0
7X0=0	9X5=45	1X2=2

Let's see another case when Image size is 5x5 and filter size is 3x3.



Image

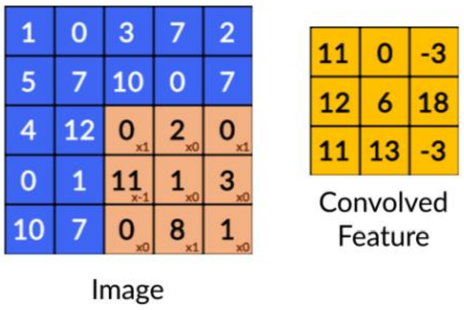
1	0	3	7	2
5	7	10	0	7
4	12	0	2	0
0	1	11	1	3
10	7	0	8	1

*

Filter

1	0	1
-1	0	0
0	1	0

the result of convolution in above figure



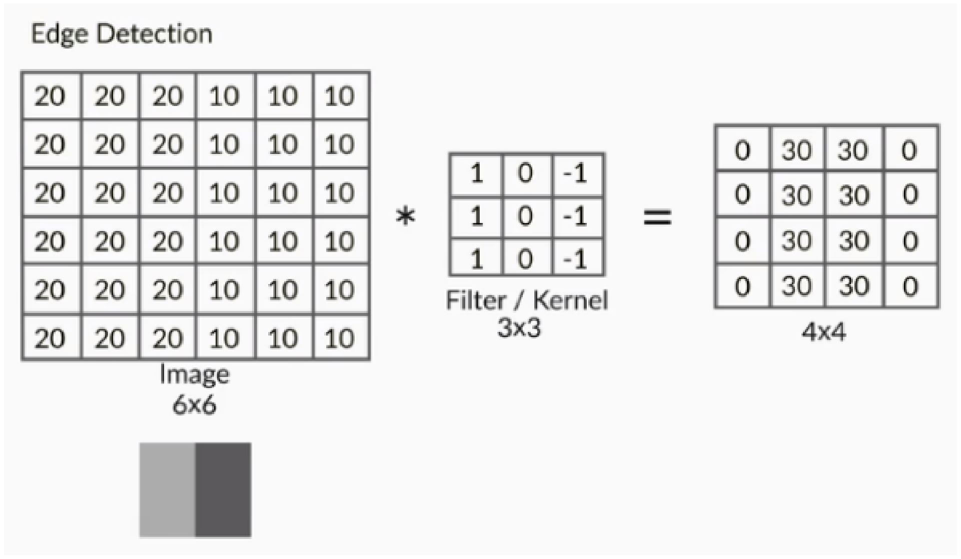
1	0	3	7	2
5	7	10	0	7
4	12	0 x1	2 x0	0 x1
0	1	11 x-1	1 x0	3 x0
10	7	0 x0	8 x1	1 x0

Image

11	0	-3
12	6	18
11	13	-3

Convolved Feature

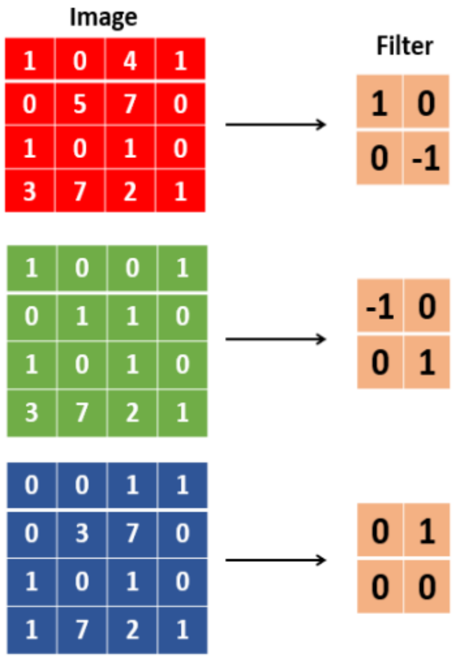
The basic idea of filters is to detect desired features (such as vertical or horizontal edges) through convolution. In the Figure 6, filter extracts edge present in the image.



$$\begin{bmatrix} 20 & 20 & 20 & 10 & 10 & 10 \\ 20 & 20 & 20 & 10 & 10 & 10 \\ 20 & 20 & 20 & 10 & 10 & 10 \\ 20 & 20 & 20 & 10 & 10 & 10 \\ 20 & 20 & 20 & 10 & 10 & 10 \\ 20 & 20 & 20 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

Fig: 6 – Filter Extracting edge presents in Images

Convolution with color image



$$\begin{bmatrix} 1 & 0 & 4 & 1 \\ 0 & 5 & 7 & 0 \\ 1 & 0 & 1 & 0 \\ 3 & 7 & 2 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

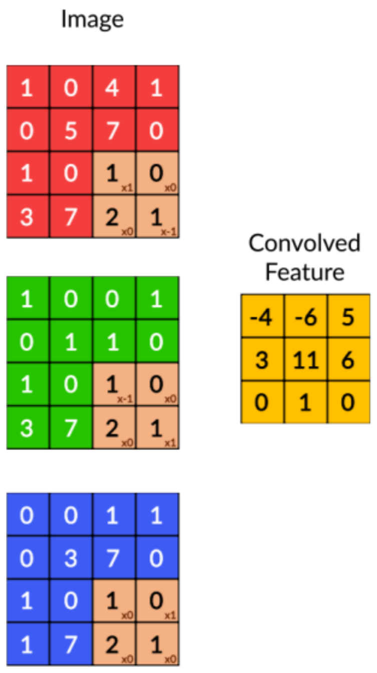
$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 3 & 7 & 2 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 3 & 7 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 7 & 2 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

***Fig: 7 –** RGB channel Images with filters*

RGB channels of Image with corresponding filters. Image with RGB channel as in above figure and convolve with their corresponding filters.

The result is shown in below figure:



Input, Filter and Output Dimension

In convolution, the number of filters should match the depth of the image. Say if we have a greyscale image with depth of 1 of size 10x10, the filter also needs to have the depth of 1 of size say 5x5. In case of a colour image with RGB channel, say an image of size 10x10x3 where 3 is the depth, we need a filter of dimension 5x5x3 where the depth of 3 is present in both the image and the filter. It is also to be noted that the result of convolution is a 2D array/matrix, irrespective of the depth of the input. So, if we use multiple filters, there will be multiple 2D array/matrix and we can stack them up.

Problem with reducing spatial dimension (height and width) in convolution

If we want to make a very deep convolutional neural network with lots of convolutional layers, the spatial dimension will reduce with each convolutional operation, and we may not be able to make the layers deep enough. But we also want a compact representation of the image for classification. To overcome this problem, we designed a schema in which the spatial dimension of the input should be preserved with each convolutional operation through the choice of proper stride and padding while reducing the representation periodically through pooling. So, the pooling layer alone is responsible for down-sampling the volume spatially

Stride & Padding

Stride: Stride is the number of pixels we move the filter (both horizontally and vertically) to perform convolution operation. Padding: Padding is the number of pixels we add all around the image. As you can in Figure 10 that the padding of 1 is used.

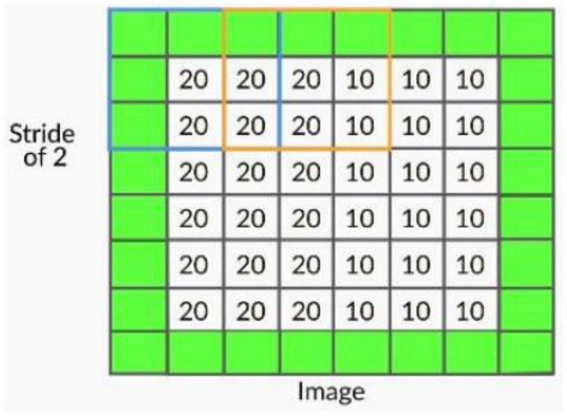
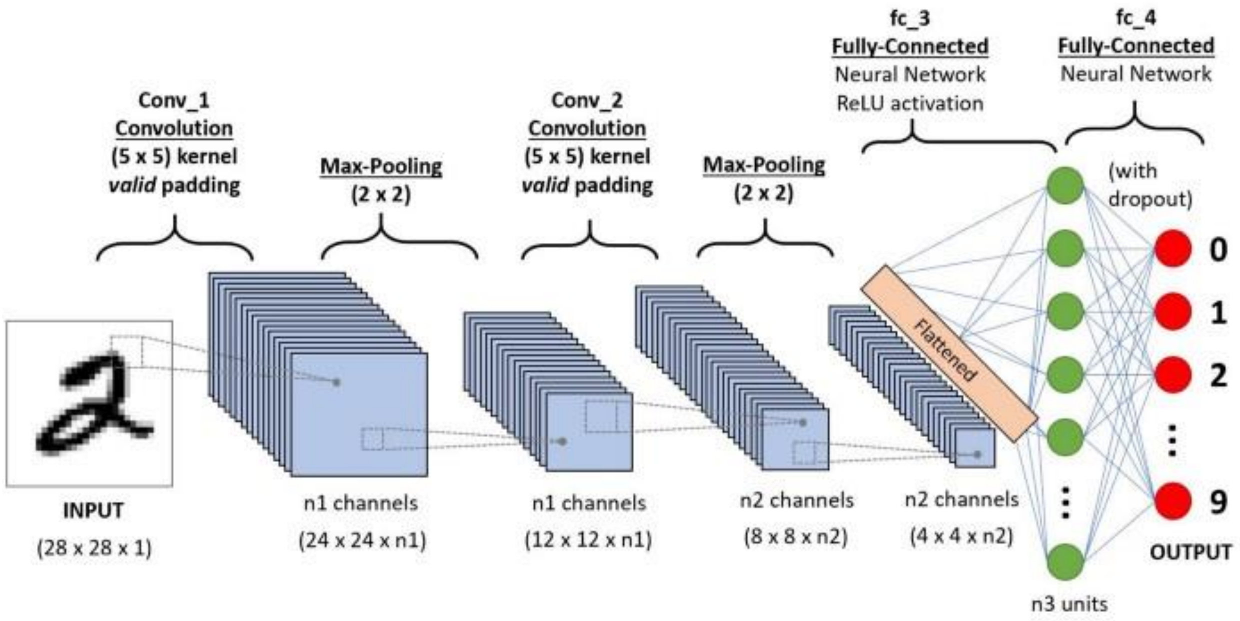


__Fig: 8 -__ Stride of 2 and padding of 1

Moving filter with smaller stride helps to capture more fine-grained information, Also, moving with larger stride helps to reduce the total number of operations in convolution and reduce the spatial dimension. But, when we are just interested in capturing more fine-grained features, and pooling layer takes care of reducing the spatial dimension, so we use a stride of 1.

Padding operation helps to maintain the height and width of the output (same as input) that we get after the convolution. Also, it helps to preserve the information at the borders. There are also alternate ways to do convolution by reducing the filter size as we approach the edge, but they are not common. Zero-padding: Padding image with '0' all around the image.



Weights in CNN

In the case of the neural network, we don’t define a specific filter. We just initialize the weights and they are learned during backpropagation. In Figure 8, the filter has three channels, and each channel of the filter convolves to the corresponding channel of the image. Thus, each step in the convolution involves the element-wise multiplication of 12 (4 operations in each RGB) pairs of numbers and adding the resultant products to get a single scalar output. Note that in each step, a single scalar number is generated, and at the end of the convolution, a 2D array is generated. You can express the convolution operation as a dot product between the weights and the input image. If you treat the (2, 2, 3) filter as a vector w of length 12, and the 12 corresponding elements of the input image as the vector p (i.e. both unrolled to a 1D vector), each step of the convolution is simply the dot product of wT and p. The dot product is computed at every patch to get a (3, 3) output array, as shown above. Apart from the weights, each filter has only 1 also bias.

$$w^T \cdot x + b = \begin{bmatrix} \text{sum}\,(w^T \cdot p_{11}) & \text{sum}\,(w^T \cdot p_{12}) & \text{sum}(w^T \cdot p_{13}) \\ \text{sum}\,(w^T \cdot p_{21}) & \text{sum}\,(w^T \cdot p_{22}) & \text{sum}(w^T \cdot p_{23}) \\ \text{sum}\,(w^T \cdot p_{31}) & \text{sum}\,(w^T \cdot p_{32}) & \text{sum}(w^T \cdot p_{33}) \end{bmatrix} + \begin{bmatrix} b & b & b \\ b & b & b \\ b & b & b \end{bmatrix}$$

$$= \begin{bmatrix} -4 & -6 & 5 \\ 3 & 11 & 6 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} b & b & b \\ b & b & b \\ b & b & b \end{bmatrix}$$

Feature Map

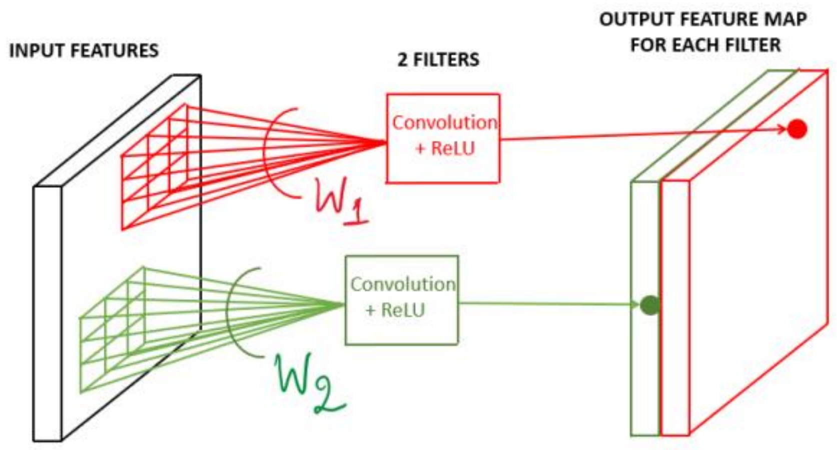


Fig: 9 – *Output Feature Map*

The term 'feature map' refers to the (non-linear) output of the activation function, not what goes into the activation function (i.e. the output of the convolution). Generally, ReLU is used as an activation function, except in the last layer where we use SoftMax activation for classification. Let’s continue the above example, use bias (b) of value ‘2’ and apply activation ReLU to get feature map

Let’s continue the above example, use bias (b) of value ‘2’ and apply activation ReLU to get feature map

$$\begin{bmatrix}-4 & -6 & 5\\ 3 & 11 & 6\\ 0 & 1 & 0\end{bmatrix} + \begin{bmatrix}2 & 2 & 2\\ 2 & 2 & 2\\ 2 & 2 & 2\end{bmatrix} = \begin{bmatrix}-2 & -4 & 7\\ 5 & 13 & 8\\ 2 & 3 & 2\end{bmatrix} \xrightarrow[\text{ReLU}]{\text{Activation}} \begin{bmatrix}0 & 0 & 7\\ 5 & 13 & 8\\ 2 & 3 & 2\end{bmatrix}$$

So, now we get one feature map as [0 0 7; 5 13 8; 2 3 2]. As you can see in the figure 11, when you have multiple filters, you will get multiple feature map.

Pooling

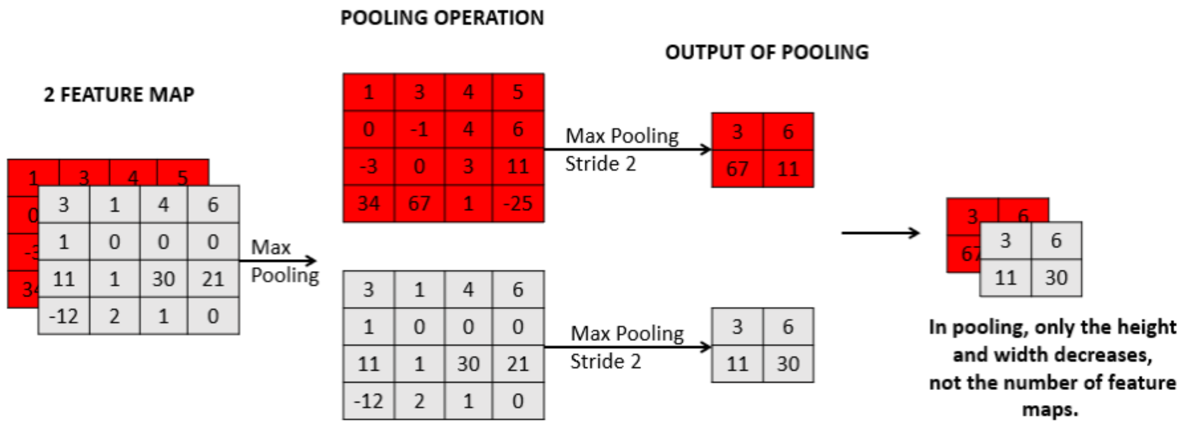


Fig: 10 – Pooling layer on Feature Map by CNN

After extracting features (as feature maps), CNNs typically aggregate these features using the pooling layer to make the representation more compact. We already used padding to make the width and height of feature map same as that of input. But we also need compact representation of the feature map. We take aggregate over a patch of feature map to get the output of pooling. Most popular are ‘Max Pooling’ and ‘Average Pooling’. Max pooling is more popular than average pooling as it has shown to work better average pooling.

Max pooling: If any one of the patches says something strongly about the presence of a certain feature, then the pooling layer counts that feature as 'detected'. Average pooling: If one patch says something very firmly but the other ones disagree, the pooling layer takes the average to find out.

Important formula

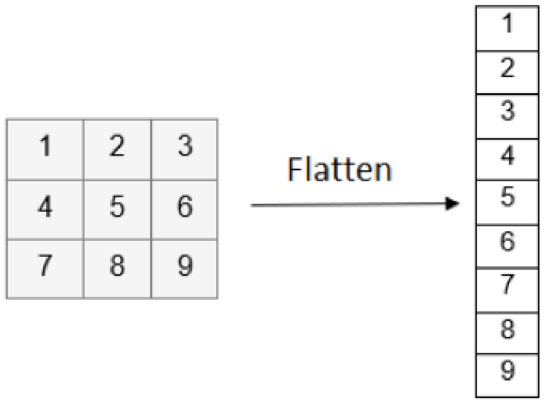
Given the following size : • Image - n x n • Filter - k x k • Padding - P • Stride - S After padding, we get an image of size (n + 2P) x (n+2P). After we convolve this padded image with the filter, we get:

$$\text{Size of convolved image } = \left(\frac{n+2P-k}{S}+1\right), \left(\frac{n+2P-k}{S}+1\right)$$

Similarly, to find the size of the Output after Pooling, you can use the above formula, and make the value of P=0, since there is no padding in Pooling operation.

Flatten, Fully Connected & Softmax

The output of the convolutional layer is 3-dimension matrix say 32x32x256. But for classification, we need a dense layer. Say, we if we want to classify an image into 5 classes, finally we need 5 neurons what will output probability. So, we need to convert the 3-dimensional matrix into fully connected. Generally, we use pooled feature map to convert into the dense layer.



After flattening the layer, we need to look at all the features extracted by the convolutional layer before we classify them. So, we need the dense layer in which all neurons in the next layer is connected to all the neurons in the previous layer. On the top of fully connected, we use 'softmax' activation for classification.

Data Loading

Importing all the important libraries

```
%tensorflow_version 2.x
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

```
import pathlib
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import os
import PIL
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Side Note: - If you are using the data by mounting the google drive, use the following:
## from google.colab import drive
## drive.mount('/content/gdrive')

This assignment uses a dataset of about 2357 images of cancer types. The dataset contains 9 sub-directories in each train and test subdirectories. The 9 sub-directories contains the images of 9 cancer types respectively.

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```python
# Defining the path for train and test images
## Todo: Update the paths of the train and test dataset

data_dir_train = pathlib.Path("/content/drive/MyDrive/Skin cancer ISIC The International Skin Imaging Collaboration/Train")
data_dir_test = pathlib.Path('/content/drive/MyDrive/Skin cancer ISIC The International Skin Imaging Collaboration/Test')
```

```python
image_count_train = len(list(data_dir_train.glob('*/*.jpg')))
print(image_count_train)
image_count_test = len(list(data_dir_test.glob('*/*.jpg')))
print(image_count_test)
```

```
2239
118
```

Load using keras.preprocessing

Let's load these images off disk using the helpful image_dataset_from_directory utility.

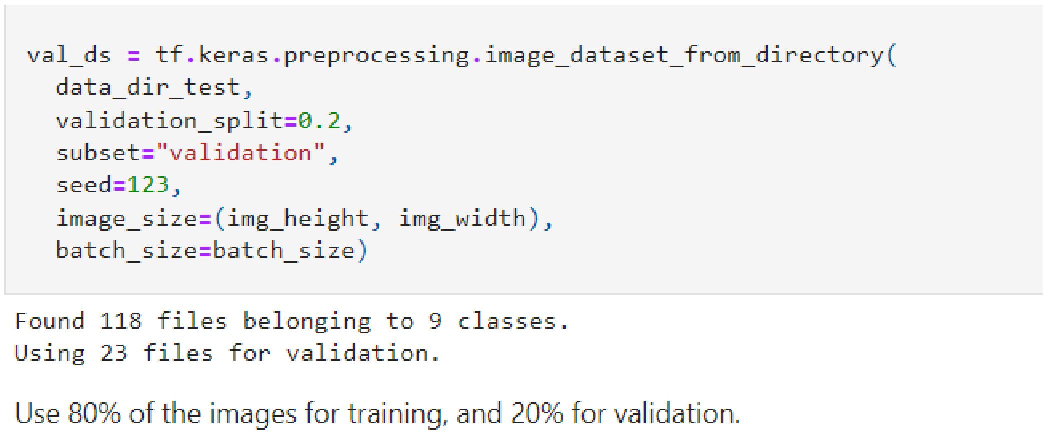
Create a dataset

Define some parameters for the loader:

```python
batch_size = 32
img_height = 180
img_width = 180
```

```python
## Write your train dataset here
## Note use seed=123 while creating your dataset using tf.keras.preprocessing.image_dataset_from_directory
## Note, make sure your resize your images to the size img_height*img_width, while writting the dataset
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_train,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 2239 files belonging to 9 classes.
Using 1792 files for training.
```



```python
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_test,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 118 files belonging to 9 classes.
Using 23 files for validation.
```

Use 80% of the images for training, and 20% for validation.

```python
# List out all the classes of skin cancer and store them in a list.
# You can find the class names in the class_names attribute on these datasets.
# These correspond to the directory names in alphabetical order.
class_names = train_ds.class_names
print(class_names)
```

```
['actinic keratosis', 'basal cell carcinoma', 'dermatofibroma', 'melanoma', 'nevus', 'pigmented benign keratosis', 'seborrheic keratosis', 'squamous ce
ll carcinoma', 'vascular lesion']
```

Importing all the important libraries

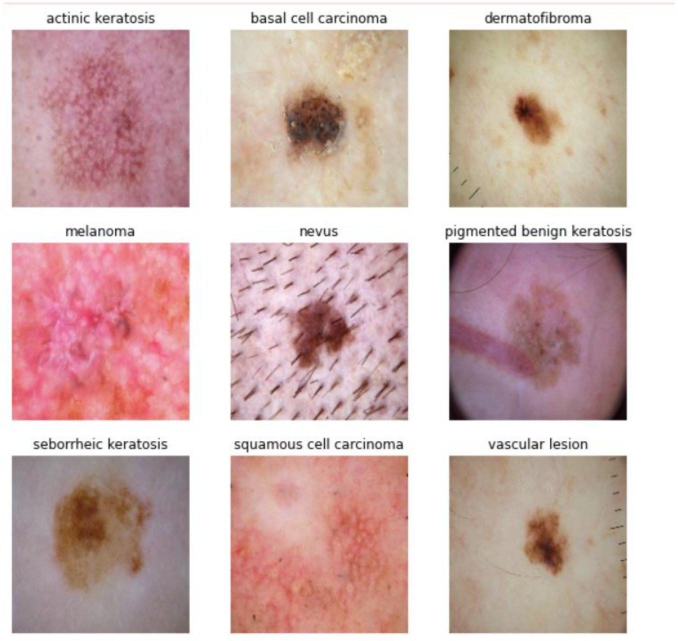
```
%tensorflow_version 2.x
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

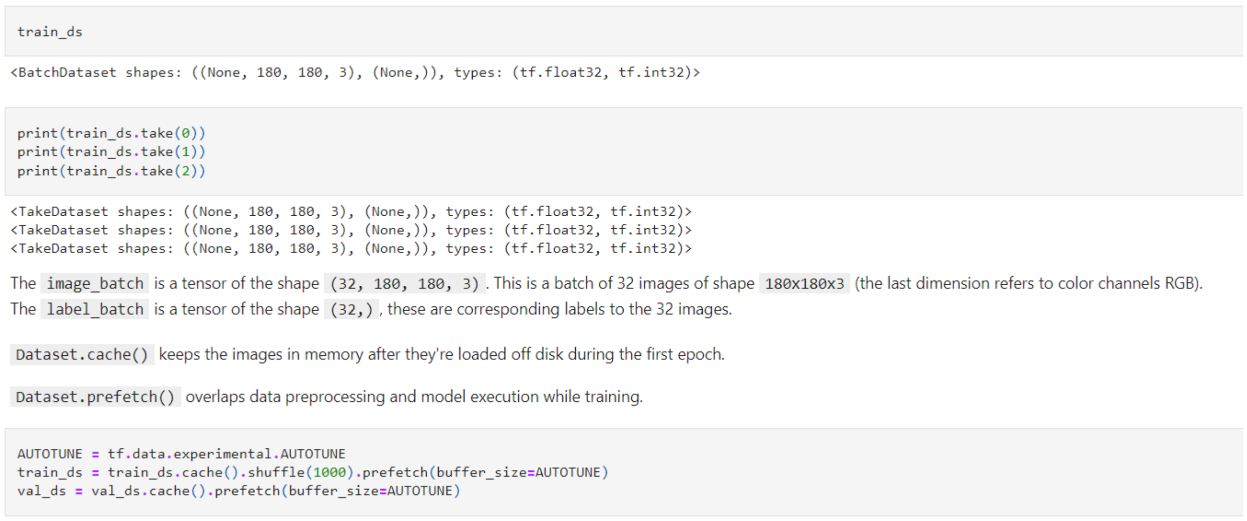
```
Found GPU at: /device:GPU:0
```

```
import pathlib
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import os
import PIL
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Visualize the data

```
import matplotlib.pyplot as plt

### your code goes here, you can use training or validation data to visualize
plt.figure(figsize = (10,10))
for images, labels in train_ds.take(2):
  for i in range(9):
    ax = plt.subplot(3,3, i+1)
    plt.imshow(images[i].numpy().astype("int64"))
    plt.title(class_names[i])
    plt.axis("off")
```





```
train_ds
```

```
<BatchDataset shapes: ((None, 180, 180, 3), (None,)), types: (tf.float32, tf.int32)>
```

```
print(train_ds.take(0))
print(train_ds.take(1))
print(train_ds.take(2))
```

```
<TakeDataset shapes: ((None, 180, 180, 3), (None,)), types: (tf.float32, tf.int32)>
<TakeDataset shapes: ((None, 180, 180, 3), (None,)), types: (tf.float32, tf.int32)>
<TakeDataset shapes: ((None, 180, 180, 3), (None,)), types: (tf.float32, tf.int32)>
```

The `image_batch` is a tensor of the shape `(32, 180, 180, 3)`. This is a batch of 32 images of shape `180x180x3` (the last dimension refers to color channels RGB). The `label_batch` is a tensor of the shape `(32,)`, these are corresponding labels to the 32 images.

`Dataset.cache()` keeps the images in memory after they're loaded off disk during the first epoch.

`Dataset.prefetch()` overlaps data preprocessing and model execution while training.

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

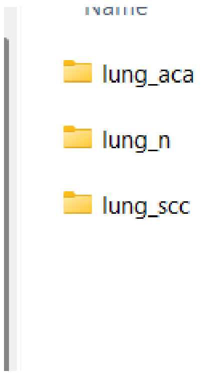
Methodology: Detection of Lung Cancer

Here, tried another deep learning solution with Lung Cancer images. We can use the lung cancer detection solution in our previous gaming project, so the users who are part of our open exercise community can utilize this system to detect cancer or reduce risk for several types of cancer via means of exercise.

Data is collected from: https://www.kaggle.com/code/sandiptabhadra/stackermodeld/data

Data Loading & Data Processing

There are three categories of lung cancers images are collected.



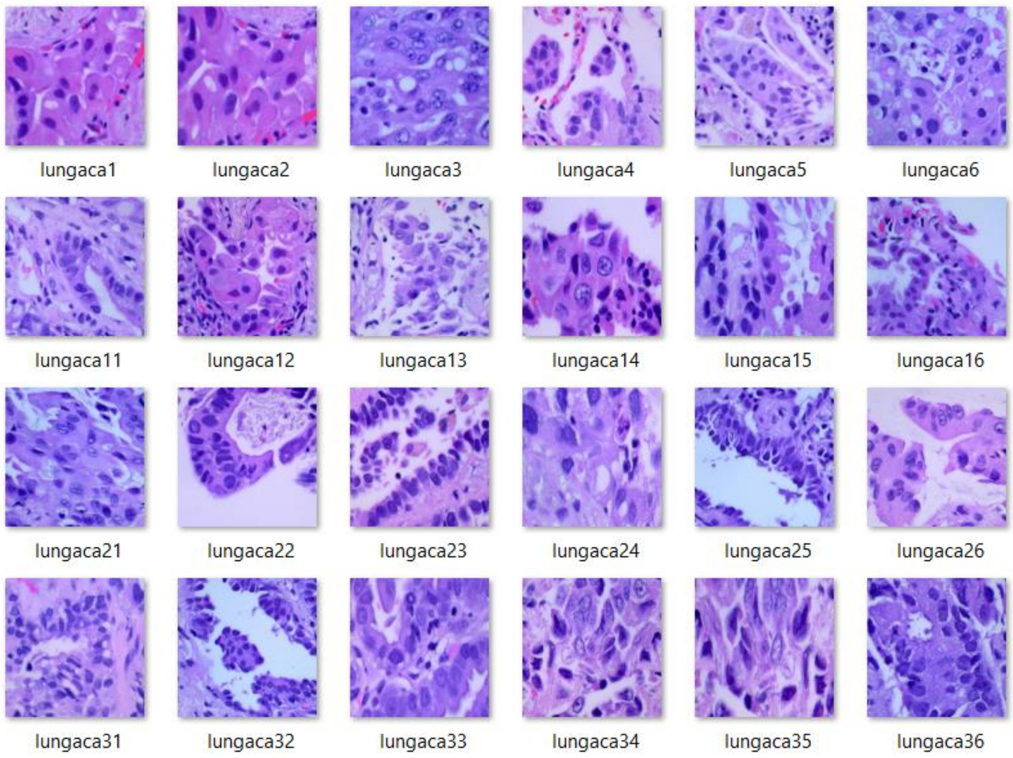
Name

lung_aca

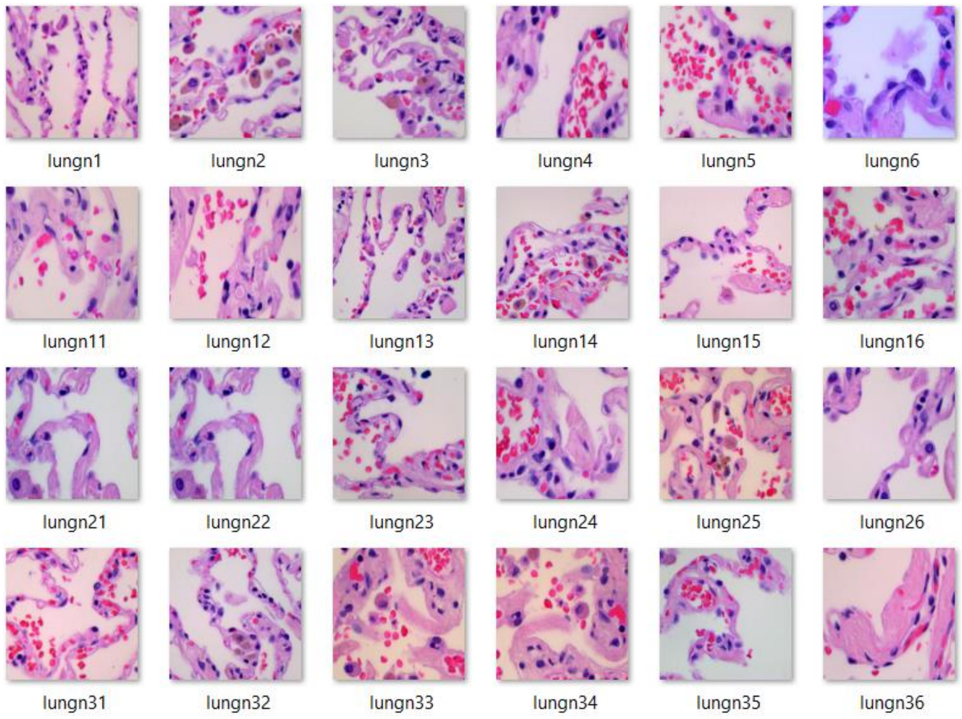
lung_n

lung_scc

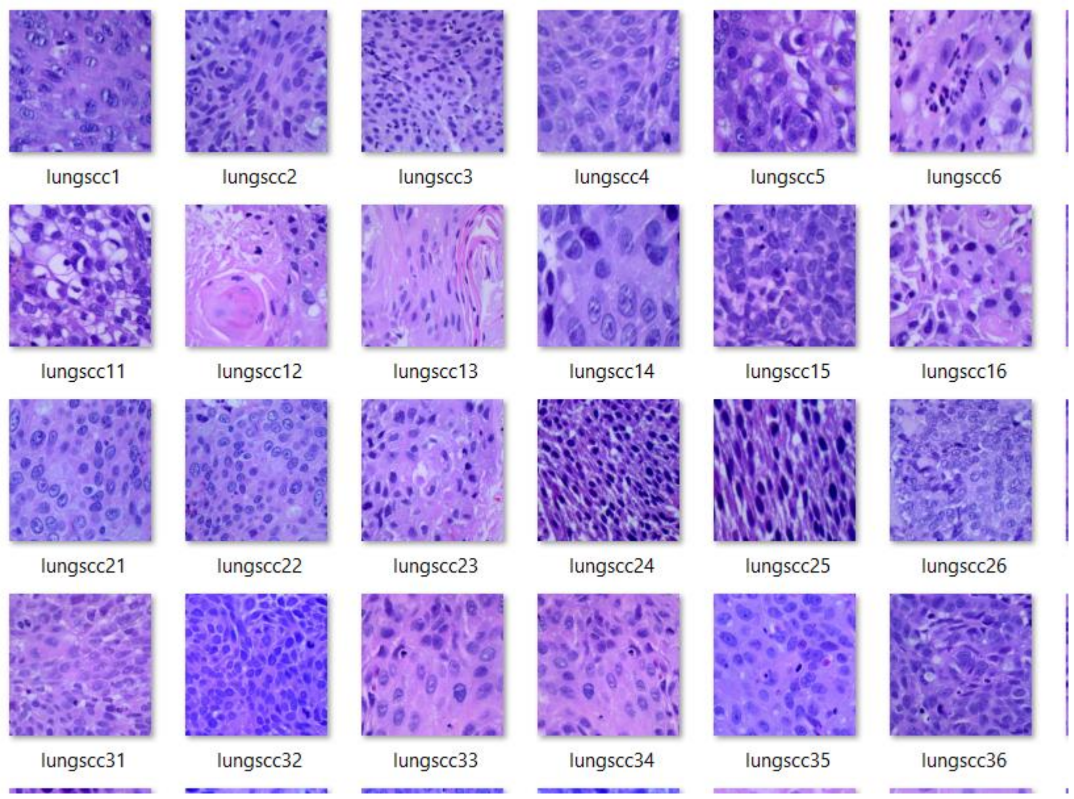
Sample Lung ACA

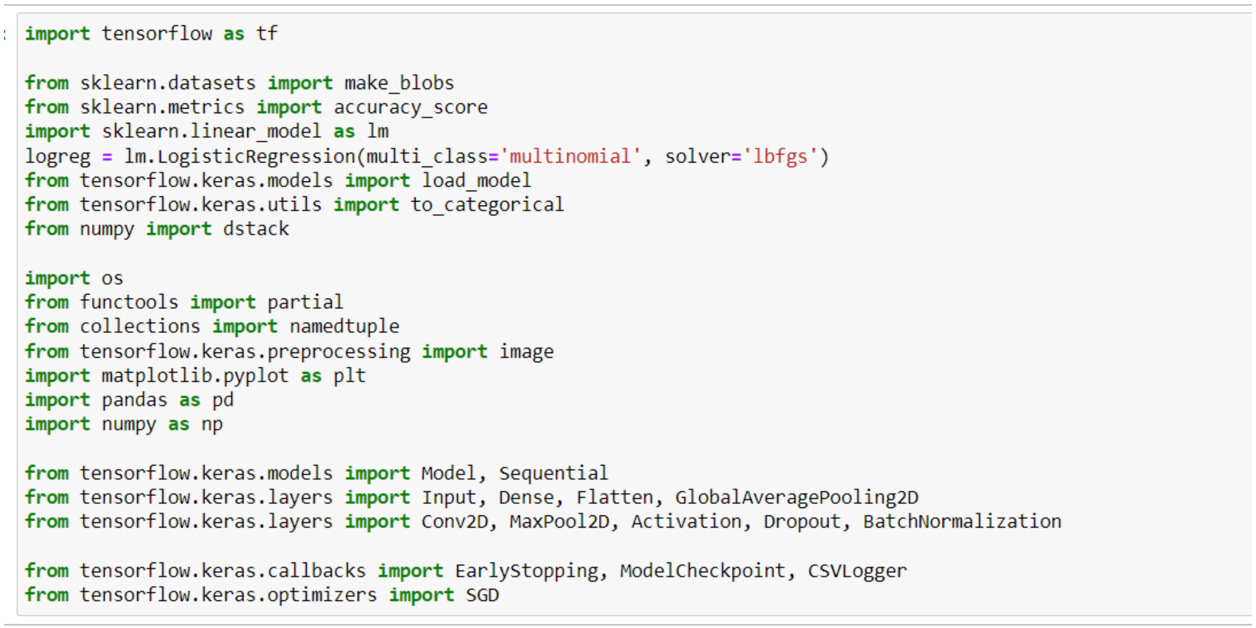


Lung N



Lung SCC





```
import tensorflow as tf

from sklearn.datasets import make_blobs
from sklearn.metrics import accuracy_score
import sklearn.linear_model as lm
logreg = lm.LogisticRegression(multi_class='multinomial', solver='lbfgs')
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
from numpy import dstack

import os
from functools import partial
from collections import namedtuple
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense, Flatten, GlobalAveragePooling2D
from tensorflow.keras.layers import Conv2D, MaxPool2D, Activation, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, CSVLogger
from tensorflow.keras.optimizers import SGD
```

```
INPUT_ROOT = ''
DATA_DIR = os.path.join(INPUT_ROOT,'lung_image_sets')
```

```
DATA_DIR
```

```
'lung_image_sets'
```

```
data = image.ImageDataGenerator(validation_split = 0.2)
BATCH_SIZE = 15
# Change batch size, if you run out of memory!
# Or increase it, if you have a lot of RAM (>> 16 Gb) for faster trainig
IMG_SHAPE = (256, 256, 3)
```

```
generate_data = partial(
    data.flow_from_directory,
    DATA_DIR,
    class_mode = 'categorical',
    color_mode = 'rgb',
    target_size = IMG_SHAPE[:2],
    batch_size = BATCH_SIZE,
    shuffle = False,
    seed = 666
)

print('Trainig data:')
train_iter = generate_data(subset='training')
print('Validating data:')
validate_iter = generate_data(subset='validation')
```

```
Trainig data:
Found 12000 images belonging to 3 classes.
Validating data:
Found 3000 images belonging to 3 classes.
```

Importing all the important libraries

```
pip install tensorflow
```

```
import tensorflow as tf

from sklearn.datasets import make_blobs
from sklearn.metrics import accuracy_score
import sklearn.linear_model as lm
logreg = lm.LogisticRegression(multi_class='multinomial', solver='lbfgs')
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
from numpy import dstack

import os
from functools import partial
from collections import namedtuple
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense, Flatten, GlobalAveragePooling2D
from tensorflow.keras.layers import Conv2D, MaxPool2D, Activation, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, CSVLogger
from tensorflow.keras.optimizers import SGD
```

```
import pathlib
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import os
import PIL
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.image import ImageDataGenerator

print(tf.__version__)
```

```
2.9.1
```

Data Loading

```
# Defining the path for train and test images


data_dir_train = pathlib.Path("C:/Users/Documents/lung_image_sets/Train")
data_dir_test = pathlib.Path('C:/Users/Documents/lung image_sets/Test')
```

```
image_count_train = len(list(data_dir_train.glob('*/*')))
print(image_count_train)
image_count_test = len(list(data_dir_test.glob('*/*')))
print(image_count_test)
```

```
14700
300
```

```
batch_size = 32
img_height = 180
img_width = 180
```

Building train dataset here with seed=123 while creating dataset using tf.keras.preprocessing.image_dataset_from_directory.

## **Note**, make sure you resize your images to the size img_height*img_width, while writing the dataset

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_train,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 14700 files belonging to 3 classes.
Using 11760 files for training.
```

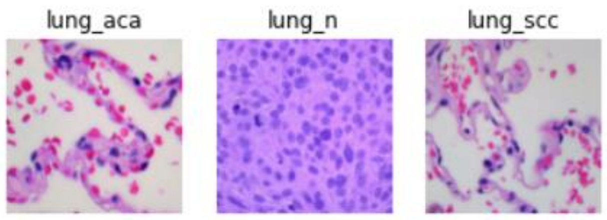
Validation dataset are built here. We used seed=123 while creating your dataset using tf.keras.preprocessing.image_dataset_from_directory.

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_test,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 300 files belonging to 3 classes.
Using 60 files for validation.
```

List out all the classes of cancer and store them in a list. We can find the class names in the class_names attribute on these datasets. These correspond to the directory names in alphabetical order.

```
import matplotlib.pyplot as plt

### visualize training or validation data to
plt.figure(figsize = (10,10))
for images, labels in train_ds.take(10):
  for i in range(3):
    ax = plt.subplot(3,5, i+1)
    plt.imshow(images[i].numpy().astype("int64"))
    plt.title(class_names[i])
    plt.axis("off")
```

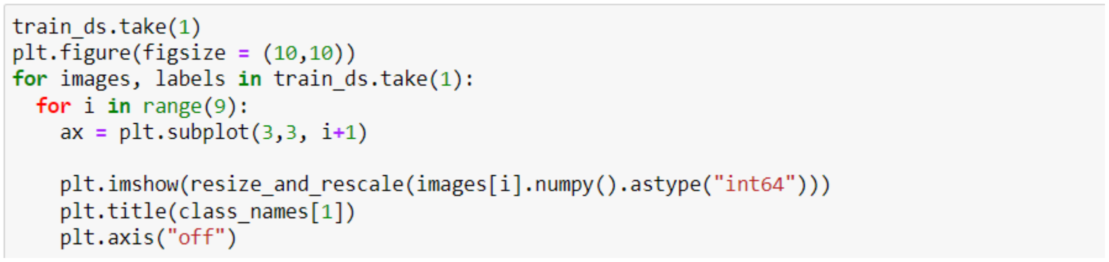


```
train_ds
```

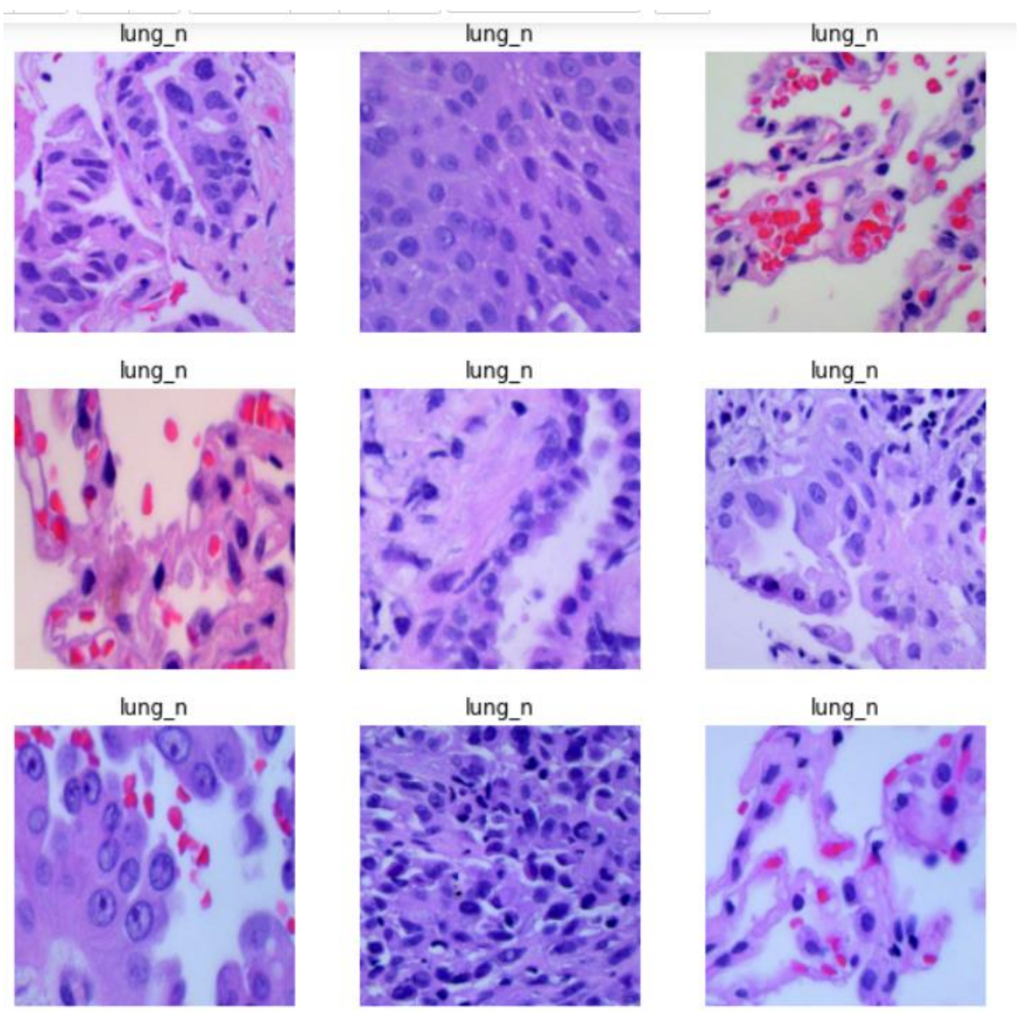
```
<BatchDataset element_spec=(TensorSpec(shape=(None, 180, 180, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype
=tf.int32, name=None))>
```

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
image_size = 180

resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(image_size, image_size),
  layers.experimental.preprocessing.Rescaling(1./255)
  #layers.experimental.preprocessing.Rescaling(1./127.5, offset=-1)
])
```



```
train_ds.take(1)
plt.figure(figsize = (10,10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3,3, i+1)

    plt.imshow(resize_and_rescale(images[i].numpy().astype("int64")))
    plt.title(class_names[1])
    plt.axis("off")
```

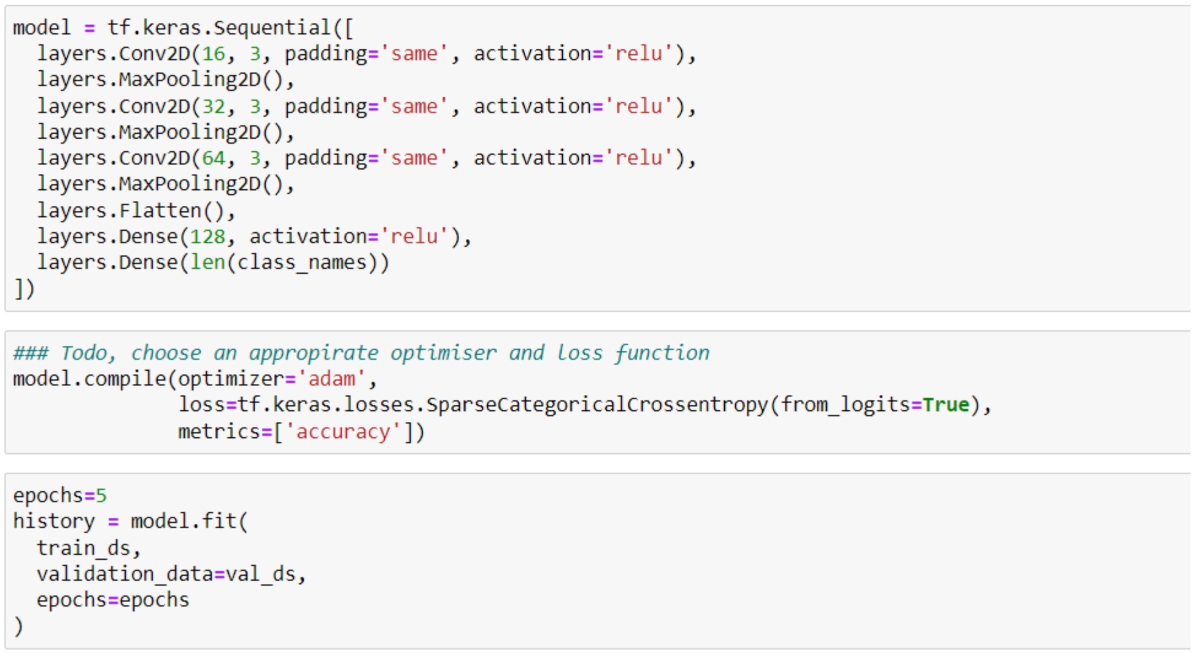


Model Building

Let’s build our first model using CNN architecture. Create the model

Using layers.experimental.preprocessing. Rescaling to normalize pixel values between (0,1). The RGB channel values are in the [0, 255] range. This is not ideal for a neural network. Here, it is good to standardize values to be in the [0, 1]

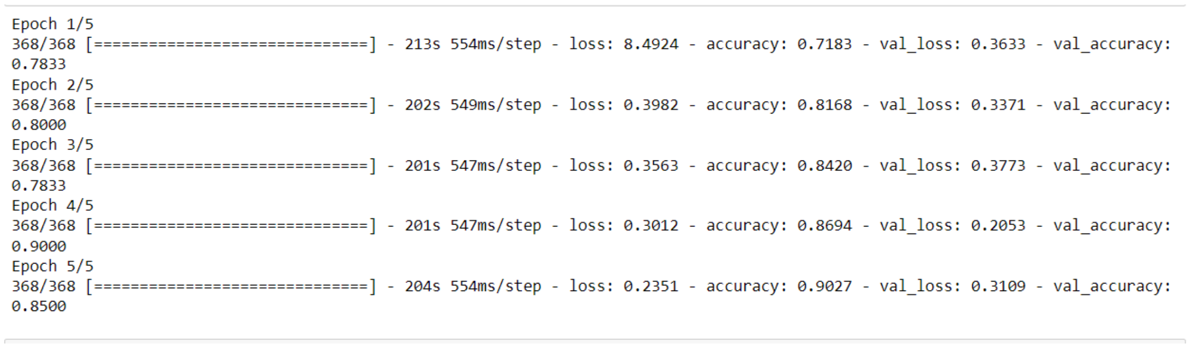
Note: the rescaling layer below standardizes pixel values to [0,1]. If instead we want [-1,1], we have to write Rescaling (1. /127.5, offset=-1).



```
model = tf.keras.Sequential([
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(len(class_names))
])
```

```
### Todo, choose an appropriate optimiser and loss function
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

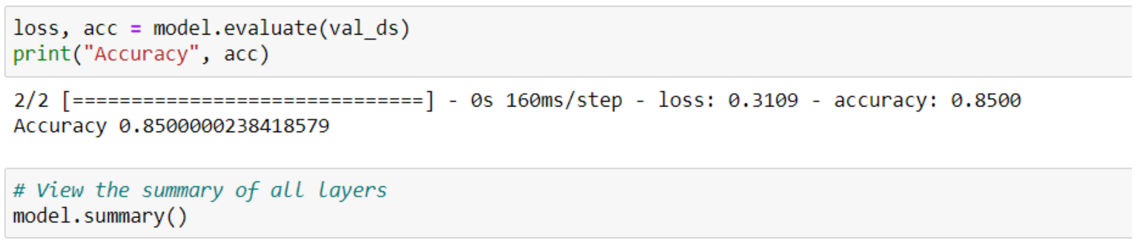
```
epochs=5
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```



```
Epoch 1/5
368/368 [==============================] - 213s 554ms/step - loss: 8.4924 - accuracy: 0.7183 - val_loss: 0.3633 - val_accuracy:
0.7833
Epoch 2/5
368/368 [==============================] - 202s 549ms/step - loss: 0.3982 - accuracy: 0.8168 - val_loss: 0.3371 - val_accuracy:
0.8000
Epoch 3/5
368/368 [==============================] - 201s 547ms/step - loss: 0.3563 - accuracy: 0.8420 - val_loss: 0.3773 - val_accuracy:
0.7833
Epoch 4/5
368/368 [==============================] - 201s 547ms/step - loss: 0.3012 - accuracy: 0.8694 - val_loss: 0.2053 - val_accuracy:
0.9000
Epoch 5/5
368/368 [==============================] - 204s 554ms/step - loss: 0.2351 - accuracy: 0.9027 - val_loss: 0.3109 - val_accuracy:
0.8500
```

Evaluation of Model

Accuracy achieved: 85%



```
loss, acc = model.evaluate(val_ds)
print("Accuracy", acc)
```

```
2/2 [==============================] - 0s 160ms/step - loss: 0.3109 - accuracy: 0.8500
Accuracy 0.8500000238418579
```

```
# View the summary of all layers
model.summary()
```

```python
# View the summary of all layers
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 180, 180, 16)      448       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 90, 90, 16)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)       0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 30976)             0         
                                                                 
 dense (Dense)               (None, 128)               3965056   
                                                                 
 dense_1 (Dense)             (None, 3)                 387       
                                                                 
=================================================================
Total params: 3,989,027
Trainable params: 3,989,027
Non-trainable params: 0
_________________________________________________________________
```

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

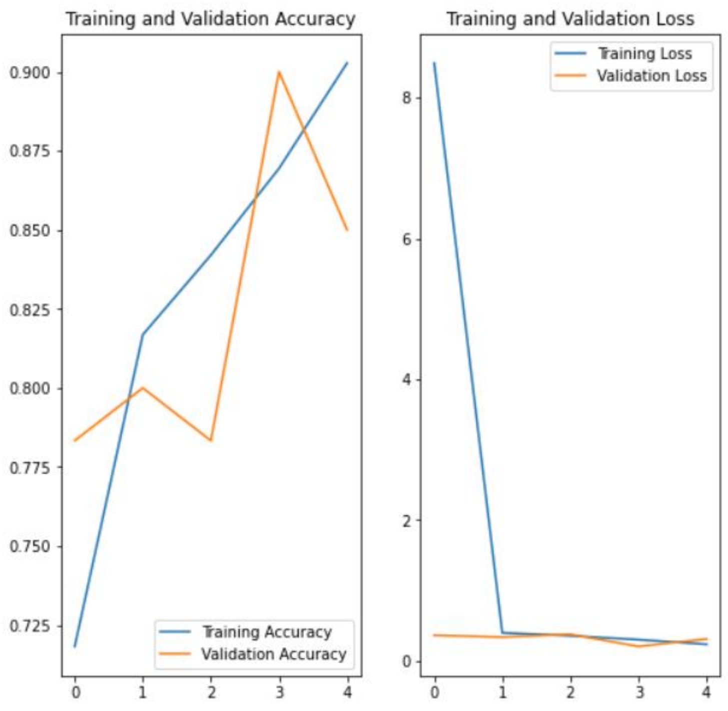


Image Data Augmentation

The model result is showing accuracy of validation dataset is highest with epoch = 3. After that accuracy has gone down drastically with increasing epochs. The accuracy is lower than training accuracy. This is the evidence of overfitting.

In this section I will **describe data augmentation**: A technique to increase the diversity of our training set by applying random (but realistic) transformations, such as image rotation.

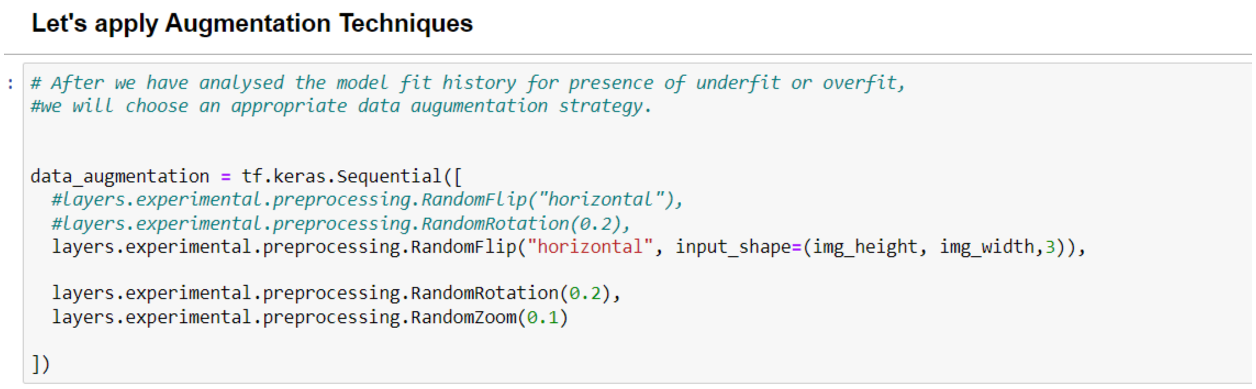
This will help to increase the validation accuracy as we will now increase the number of images for better learning.

We will apply data augmentation in two ways:

Use the Keras preprocessing layers, such as tf.keras.layers.RandomFlip-horizontal, and tf.keras.layers.RandomRotation & tf.keras.layers.RandomZoom

Model Building with Augmentation Techniques

s



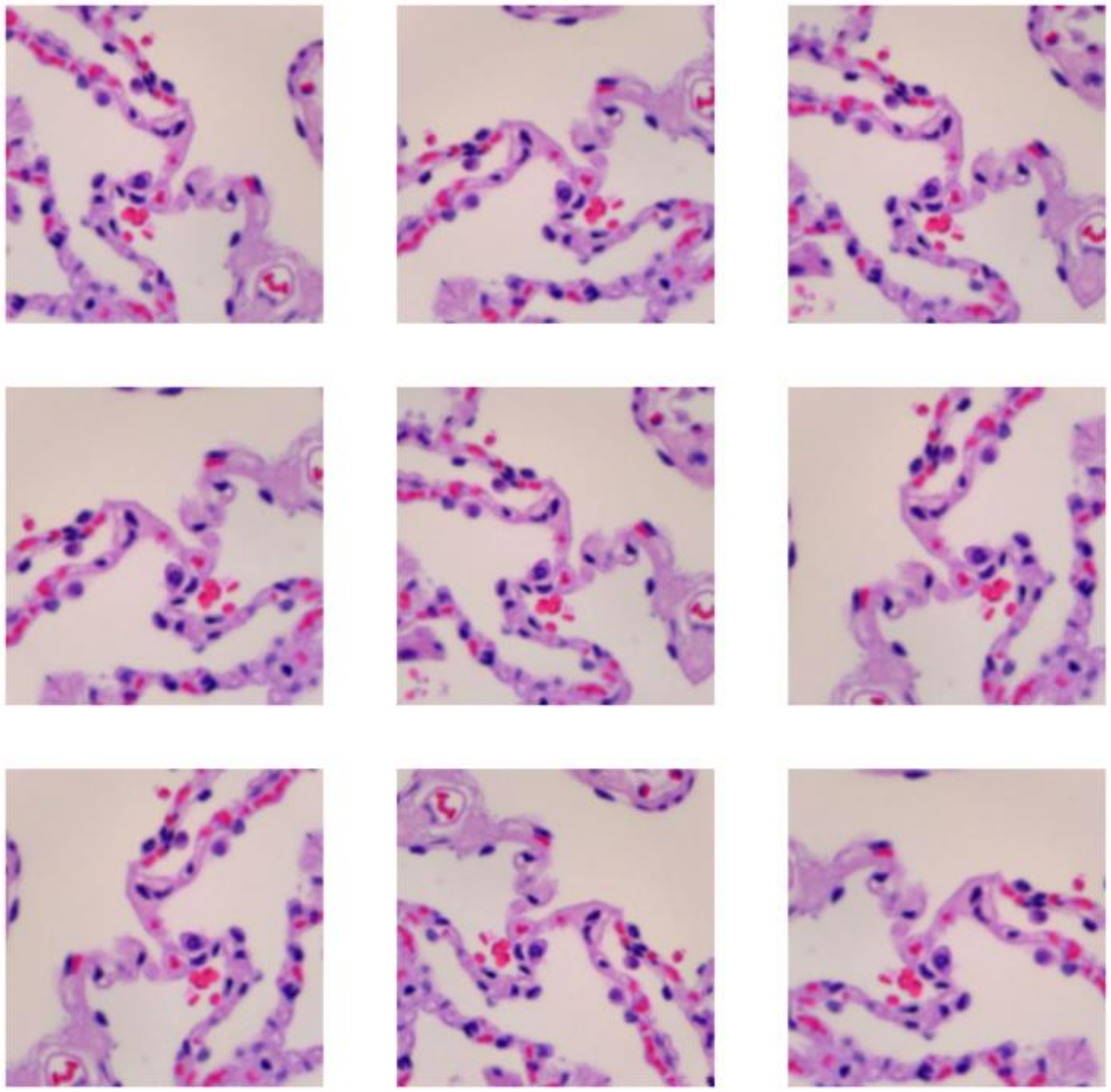
Let's apply Augmentation Techniques

```
# After we have analysed the model fit history for presence of underfit or overfit,
#we will choose an appropriate data augumentation strategy.


data_augmentation = tf.keras.Sequential([
  #layers.experimental.preprocessing.RandomFlip("horizontal"),
  #layers.experimental.preprocessing.RandomRotation(0.2),
  layers.experimental.preprocessing.RandomFlip("horizontal", input_shape=(img_height, img_width,3)),

  layers.experimental.preprocessing.RandomRotation(0.2),
  layers.experimental.preprocessing.RandomZoom(0.1)

])
```

Visualize few augmented examples how do they look like by applying data augmentation to the same image several times:

Here, use data augmentation to train the model.

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(4):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```



```
image, label = next(iter(train_ds))
```

```
# Add the image to a batch
image = tf.expand_dims(image, 0)
```

Create the model, compile and train the model

```
model = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(len(class_names))
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Train the model

```
epochs=20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/20
368/368 [==============================] - 166s 442ms/step - loss: 0.4108 - accuracy: 0.8094 - val_loss: 0.2766 - val_accuracy:
0.8500
Epoch 2/20
368/368 [==============================] - 168s 457ms/step - loss: 0.2207 - accuracy: 0.9117 - val_loss: 0.1998 - val_accuracy:
0.9167
Epoch 3/20
368/368 [==============================] - 173s 469ms/step - loss: 0.1733 - accuracy: 0.9310 - val_loss: 0.2117 - val_accuracy:
0.9000
Epoch 4/20
368/368 [==============================] - 175s 476ms/step - loss: 0.1466 - accuracy: 0.9439 - val_loss: 0.1589 - val_accuracy:
0.9167
Epoch 5/20
368/368 [==============================] - 3585s 10s/step - loss: 0.1364 - accuracy: 0.9465 - val_loss: 0.1323 - val_accuracy:
0.9333
Epoch 6/20
368/368 [==============================] - 146s 396ms/step - loss: 0.1310 - accuracy: 0.9484 - val_loss: 0.1183 - val_accuracy:
0.9500
Epoch 7/20
368/368 [==============================] - 167s 455ms/step - loss: 0.1182 - accuracy: 0.9545 - val_loss: 0.0829 - val_accuracy:
0.9500
Epoch 8/20
368/368 [==============================] - 171s 466ms/step - loss: 0.1023 - accuracy: 0.9573 - val_loss: 0.1181 - val_accuracy:
0.9667
Epoch 9/20
368/368 [==============================] - 184s 501ms/step - loss: 0.1028 - accuracy: 0.9577 - val_loss: 0.2151 - val_accuracy:
0.9000
Epoch 10/20
368/368 [==============================] - 193s 526ms/step - loss: 0.0901 - accuracy: 0.9662 - val_loss: 0.1859 - val_accuracy:
0.9000
```

```
Epoch 11/20
368/368 [==============================] - 185s 504ms/step - loss: 0.0993 - accuracy: 0.9600 - val_loss: 0.1018 - val_accuracy:
0.9667
Epoch 12/20
368/368 [==============================] - 176s 477ms/step - loss: 0.0888 - accuracy: 0.9649 - val_loss: 0.0747 - val_accuracy:
0.9500
Epoch 13/20
368/368 [==============================] - 166s 450ms/step - loss: 0.0735 - accuracy: 0.9714 - val_loss: 0.0676 - val_accuracy:
0.9500
Epoch 14/20
368/368 [==============================] - 172s 466ms/step - loss: 0.0754 - accuracy: 0.9714 - val_loss: 0.0246 - val_accuracy:
1.0000
Epoch 15/20
368/368 [==============================] - 167s 455ms/step - loss: 0.0682 - accuracy: 0.9736 - val_loss: 0.0302 - val_accuracy:
1.0000
Epoch 16/20
368/368 [==============================] - 238s 648ms/step - loss: 0.0717 - accuracy: 0.9717 - val_loss: 0.0725 - val_accuracy:
0.9833
Epoch 17/20
368/368 [==============================] - 163s 442ms/step - loss: 0.0506 - accuracy: 0.9803 - val_loss: 0.0795 - val_accuracy:
0.9667
Epoch 18/20
368/368 [==============================] - 166s 452ms/step - loss: 0.0538 - accuracy: 0.9798 - val_loss: 0.0935 - val_accuracy:
0.9667
Epoch 19/20
368/368 [==============================] - 168s 456ms/step - loss: 0.0519 - accuracy: 0.9802 - val_loss: 0.0473 - val_accuracy:
0.9667
Epoch 20/20
368/368 [==============================] - 185s 503ms/step - loss: 0.0553 - accuracy: 0.9794 - val_loss: 0.1742 - val_accuracy:
0.9333
```

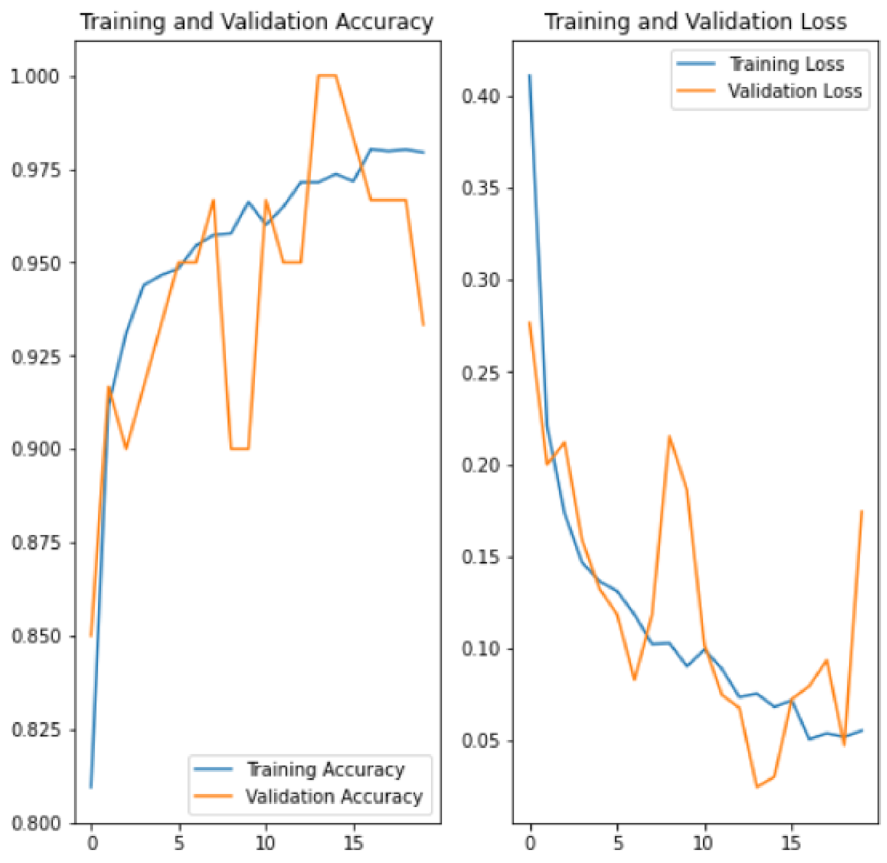
```
: model.summary()
```

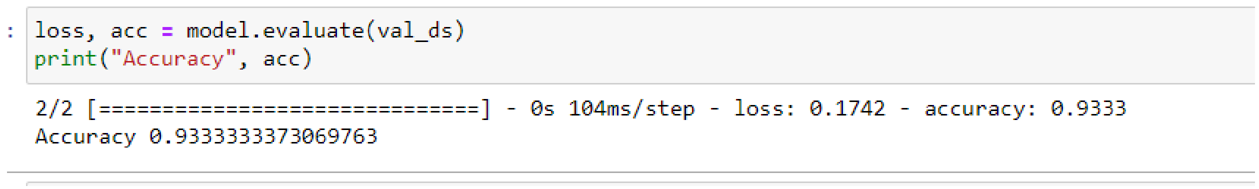
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_2 (Sequential)   (None, 180, 180, 3)       0

 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_3 (Conv2D)           (None, 180, 180, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 90, 90, 16)       0
 2D)

 conv2d_4 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 22, 22, 128)       73856

 max_pooling2d_6 (MaxPooling  (None, 11, 11, 128)      0
 2D)

 dropout (Dropout)           (None, 11, 11, 128)       0

 flatten_1 (Flatten)         (None, 15488)             0

 dense_2 (Dense)             (None, 128)               1982592

 dense_3 (Dense)             (None, 3)                 387
```

Evaluation of Model with Augmented Data

Visualizing Results

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```





```
: loss, acc = model.evaluate(val_ds)
  print("Accuracy", acc)

  2/2 [==============================] - 0s 104ms/step - loss: 0.1742 - accuracy: 0.9333
  Accuracy 0.9333333373069763
```

Hence the model validation accuracy has increased. Here we have shown how the image data can be augmented and we can increase the model accuracy.

Conclusion

In this research, we talked about an image detection technique to detect early-stage lung cancer. In cancer treatment, the time factor is very crucial. Image quality and accuracy are the core factors of this research, image quality assessment as well as enhancement stage are adopted through basic pre-processing like augmentation. There is a lot of research scope in this field that can bring life-saving innovations and contribute to the healthcare system.

Exercise helps to prevent severe cancer-type diseases and the gaming project can use this technique to develop an early detection process. The open exercise community can use this deep learning-based cancer detection system for early detection and reduce the risk of cancer severity.

This research is using CNN based deep learning technique which is offering an advanced architecture with various optimization processes. Image processing is an important part done before the model development.

Medical imaging 2D & 3D imaging are a great field of research for medical and AI professionals.

The proposed architecture can be used to build the base model and then gradually increase the CNN layers and experiment with various filters and layers.

This project will help the gaming project team to build an AI Deep Learning model and implement the cancer detection module in their gaming system as an advanced option which will attract more people from the open exercise community.

Reference

[1]	https://cdn-images-1.medium.com/max/1600/1*PhvTchwxs0HwZzKKq2KykA.png
[2]	https://sennalabs.com/rails/active_storage/blobs/eyJfcmFpbHMiOnsibWVzc2FnZSI6IkJBaHBBZ2NCIiwiZXhwIjpudWxsLCJwdXIiOiJibG9iX2lkIn19--f0628df2eec880693aaeab6a3d3a4f8320b98d90/Medical-care-deep-learning%20copy.jpg