Sana’a University
Faculty of Engineering
Mechatronics Engineering
Department
Mechatronics 5th Year

Lung Cancer Detection Using Transfer Learning

Artificial Intelligence Course Project

Using Python

Supervised By:
Eng.Yousef Al-Qeez
Done by:
Ayman Tawfiq Abdulbaki Mohammed 202170156
Eyad Khaled Hussen Al-Sayaghi 202170048

Introduction:

In our project, we build a classifier using the Transfer Learning technique which can classify normal lung tissues from cancerous.

Transfer Learning

In a convolutional neural network, the main task of the convolutional layers is to enhance the important features of an image. If a particular filter is used to identify the straight lines in an image then it will work for other images as well this is particularly what we do in transfer learning. There are models which are developed by researchers by regress hyperparameter tuning and training for weeks on millions of images belonging to 1000 different classes like imagenet dataset. A model that works well for one computer vision task proves to be good for others as well. Because of this reason, we leverage those trained convolutional layers parameters and tuned hyperparameter for our task to obtain higher accuracy.

Transfer learning is a technique in machine learning where a pre-trained model is used as a starting point for a new task or problem. Instead of training a model from scratch, transfer learning leverages the knowledge and learned features of a pre-trained model to improve performance on a new, related task.

By leveraging transfer learning, you can benefit from the knowledge and features learned by pre-trained models, even with limited data or computational resources. It is a powerful technique that has been widely used in various domains, including computer vision and natural language processing.

Computer Vision is one of the applications of deep neural networks that enables us to automate tasks that earlier required years of expertise and one such use in predicting the presence of cancerous cells.

This project has been developed using collab and the dataset has been taken from Kaggle.

Objectives:

1.The project aims to develop a system that can accurately detect lung cancer at an early stage, increasing the chances of successful treatment and improving patient outcomes.

2.The objective is to create an automated system that can accurately differentiate between lung cancer and non-cancerous conditions, reducing false positives and false negatives. This will streamline the diagnostic process, minimize human error, and save time and resources.

3. The project aims to provide a reliable and accessible tool for healthcare professionals to screen and diagnose lung cancer. Additionally, it aims to provide insights into the stage and severity of the disease, assisting in treatment planning and contributing to research and development in the field.

Working:

Here are the general steps involved in transfer learning:

1. Choose a pre-trained model that has been trained on a large dataset, typically on a similar task or domain as your target task. Our pre-trained model is Inception

2. Remove the last few layers of the pre-trained model, which are usually responsible for the final classification. These layers are task-specific and need to be replaced with new layers for your specific task.

3. Add new layers on top of the pre-trained model. These new layers will be responsible for adapting the pre-trained features to your specific task. The number and architecture of these layers depend on your specific problem.

4. Freeze the weights of the pre-trained layers to prevent them from being updated during training. This allows the pre-trained features to remain intact while only training the new layers.

5. Train the model using your own dataset. Since the pre-trained layers are frozen, only the new layers will be updated during training. This process is typically faster and requires less data compared to training a model from scratch.

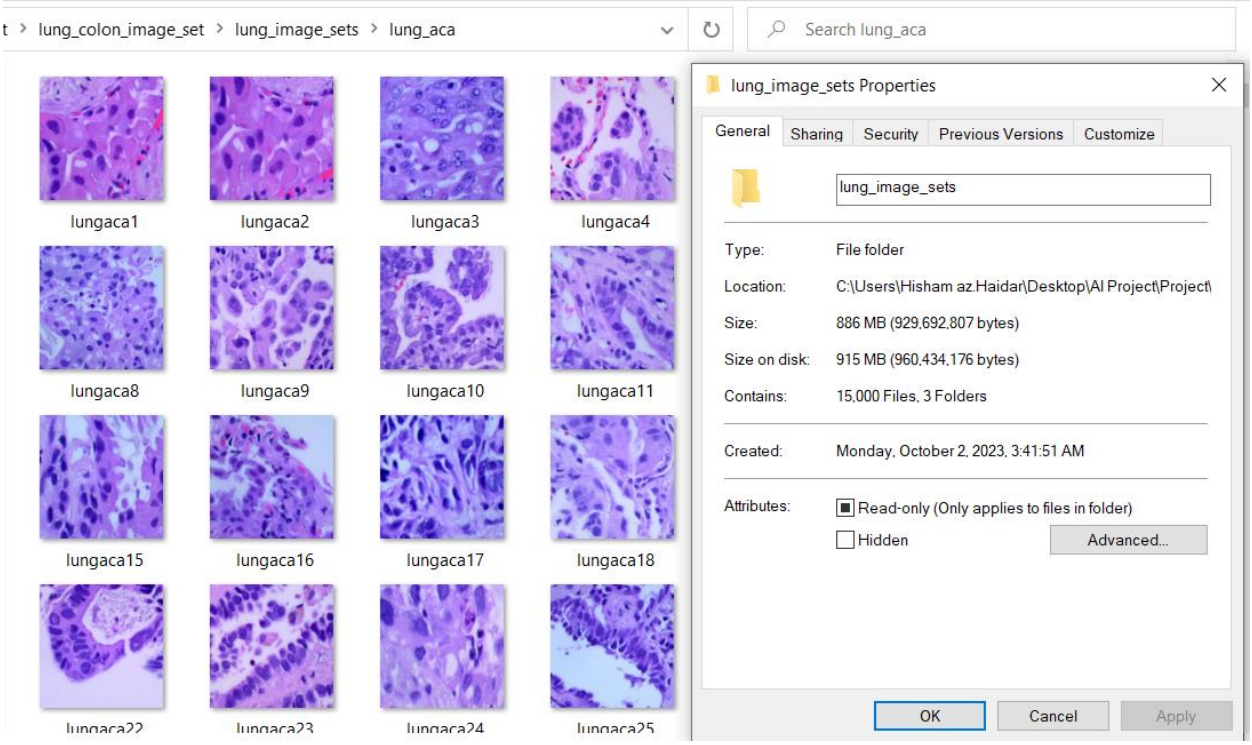
Libraries used in the implementation of the project

Python libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

- Pandas – This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
- Numpy – Numpy arrays are very fast and can perform large computations in a very short time.
- Matplotlib – This library is used to draw visualizations.
- Sklearn – This module contains multiple libraries having pre-implemented functions to perform tasks from data preprocessing to model development and evaluation.
- OpenCV – This is an open-source library mainly focused on image processing and handling.
- Tensorflow – This is an open-source library that is used for Machine Learning and Artificial intelligence and provides a range of functions to achieve complex functionalities with single lines of code.

Importing Dataset

The dataset which we will use here has been taken from- https://www.kaggle.com/datasets/andrewmvd/lung-and-colon-cancer-histopathological-images. .



This dataset includes 5000 images for three classes of lung conditions:

- Normal Class
- Lung Adenocarcinomas
- Lung Squamous Cell Carcinomas

Their size is about 1GB so it takes too long time during the processing of these images

These images for each class have been developed from 250 images by performing Data Augmentation on them. That is why we won't be using Data Augmentation further on these images.

Data Visualization

we will try to visualize some images which have been provided to us to build the classifier for each class.

Data Preparation for Training

This done through converting the given images into NumPy arrays of their pixels after resizing them because training a Deep Neural Network on large-size images is highly inefficient in terms of computational cost and time.

For this purpose, we will use the OpenCV library and Numpy library of python to serve the purpose. Also, after all the images are converted into the desired format we will

split them into training and validation data so, that we can evaluate the performance of our model.

Model Development

We will use pre-trained weight for an Inception network which is trained on imagenet dataset. This dataset contains millions of images for around 1000 classes of images.

Model Architecture

We will implement a model using the Functional API of Keras which will contain the following parts:

- The base model is the Inception model in this case.
- The Flatten layer flattens the output of the base model's output.
- Then we will have two fully connected layers followed by the output of the flattened layer.
- We have included some BatchNormalization layers to enable stable and fast training and a Dropout layer before the final layer to avoid any possibility of overfitting.
- The final layer is the output layer which outputs soft probabilities for the three classes.

The parameters of a model we import are already trained on millions of images and for weeks so, we do not need to train them again.

'Mixed7' is one of the layers in the inception network whose outputs we will use to build the classifier.

While compiling a model we provide these three essential parameters:

- optimizer – This is the method that helps to optimize the cost function by using gradient descent.
- loss – The loss function by which we monitor whether the model is improving with training or not.
- metrics – This helps to evaluate the model by predicting the training and the validation data.

Callback

Callbacks are used to check whether the model is improving with each epoch or not. If not then what are the necessary steps to be taken like ReduceLROnPlateau decreases the learning rate further? Even then if model performance is not improving then training

will be stopped by EarlyStopping. We can also define some custom callbacks to stop training in between if the desired results have been obtained early.

Program:

Importing Libraries

In [3]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from glob import glob

from sklearn.model_selection import train_test_split
from sklearn import metrics

import cv2
import gc
import os

import tensorflow as tf
from tensorflow import keras
from keras import layers

import warnings
warnings.filterwarnings('ignore')
```

Extract Data

In [4]:

```
from zipfile import ZipFile
data_path = 'lung_colon_image_set\lung_image_sets.zip'

with ZipFile(data_path,'r') as zip:
  zip.extractall()
  print('The data set has been extracted.')
```

```
The data set has been extracted.
```

Data Visualization

We will try to visualize some images which have been provided to us to build the classifier for each class.

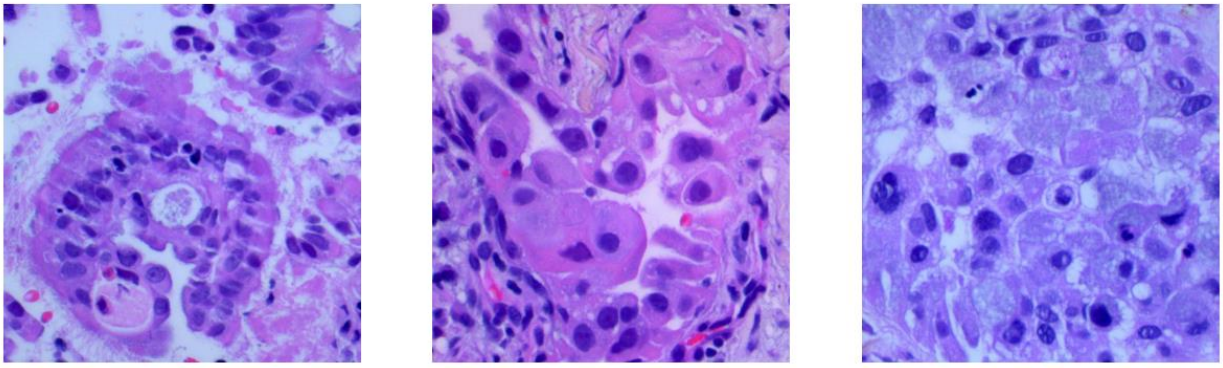
```
path = 'lung_colon_image_set\lung_image_sets'
classes = os.listdir(path)
classes
```

```
 ['lung_aca', 'lung_n', 'lung_scc']
```

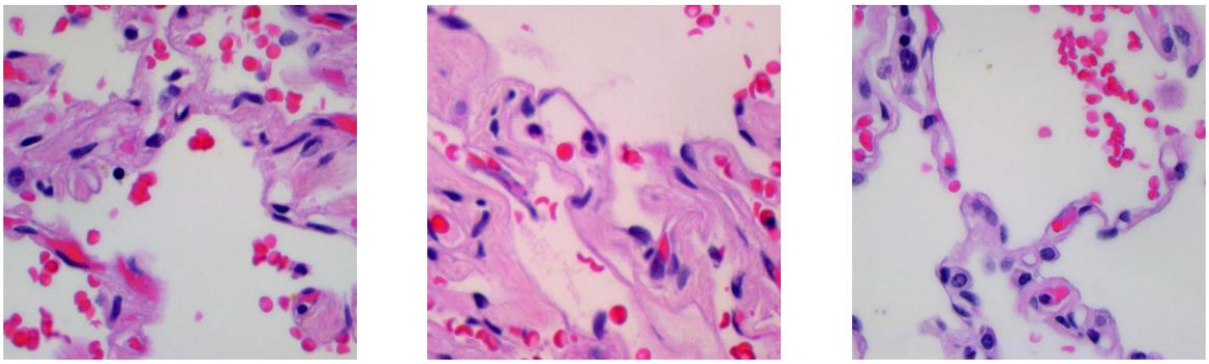
Three images from each class of the three classes

```
path = 'lung_colon_image_set\lung_image_sets'

for cat in classes:
      image_dir = f'{path}/{cat}'
      images = os.listdir(image_dir)

      fig, ax = plt.subplots(1, 3, figsize = (15, 5))
      fig.suptitle(f'Images for {cat} category . . . .',
                        fontsize = 20)

      for i in range(3):
            k = np.random.randint(0, len(images))
            img =
np.array(Image.open(f'{path}/{cat}/{images[k]}'))
            ax[i].imshow(img)
            ax[i].axis('off')
      plt.show()
```

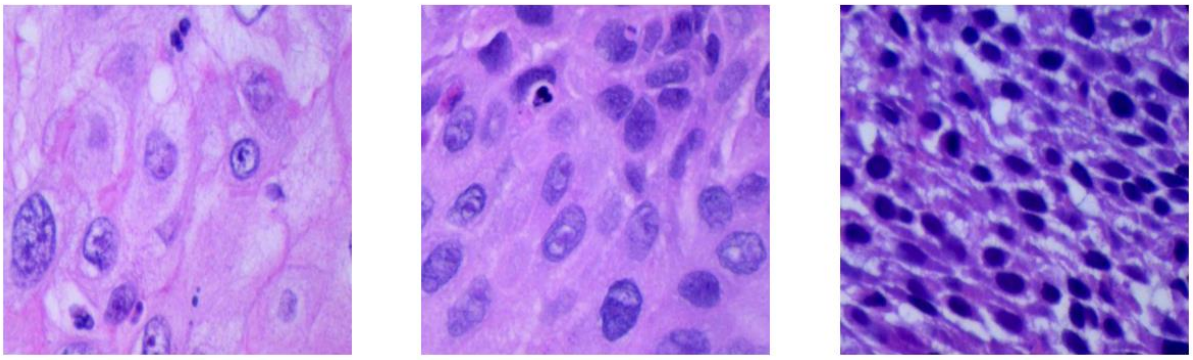
Images for lung_aca category . . . .



Images for lung_n category . . . .



Images for lung_scc category . . . .



Data Preparation for Training

```
IMG_SIZE = 256
SPLIT = 0.2
EPOCHS = 10
BATCH_SIZE = 64
```

convert the given images into NumPy arrays of their pixels after resizing them because training

```
X = []
Y = []

for i, cat in enumerate(classes):
    images = glob(f'{path}/{cat}/*.jpeg')

    for image in images:
          img = cv2.imread(image)

          X.append(cv2.resize(img, (IMG_SIZE, IMG_SIZE)))
          Y.append(i)

X = np.asarray(X)
one_hot_encoded_Y = pd.get_dummies(Y).values
```

One hot encoding

```
X_train, X_val, Y_train, Y_val = train_test_split(
    X, one_hot_encoded_Y, test_size = SPLIT, random_state =
2022)
print(X_train.shape, X_val.shape)
```

```
 (12000, 256, 256, 3) (3000, 256, 256, 3)
```

Model Development

```
from tensorflow.keras.applications.inception_v3 import
InceptionV3

pre_trained_model = InceptionV3(
      input_shape = (IMG_SIZE, IMG_SIZE, 3),
      weights = 'imagenet',
      include_top = False
```

```
)

Downloading data from
https://storage.googleapis.com/tensorflow/keras-
applications/inception_v3/inception_v3_weights_tf_dim_ordering_t
f_kernels_notop.h5
87910968/87910968 [==============================] - 527s
6us/step
```

how deep this model Number of layers in the pre-train

```
len(pre_trained_model.layers)
```

```
311
```

The parameters of a model we import are already trained, we do not need to train them again.

In [15]:

```
for layer in pre_trained_model.layers:
    layer.trainable = False
```

In [17]:

```
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output
last layer output shape:  (None, 14, 14, 768)
```

In [19]:

```
x = layers.Flatten()(last_output)

x = layers.Dense(256,activation='relu')(x)
x = layers.BatchNormalization()(x)

x = layers.Dense(128,activation='relu')(x)
x = layers.Dropout(0.3)(x)
x = layers.BatchNormalization()(x)

output = layers.Dense(3, activation='softmax')(x)

model = keras.Model(pre_trained_model.input, output)
```

compiling a model

In [22]:

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

Callback

In [24]:

```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs = {}):
        if logs.get('val_accuracy') > 0.90:
            print('\n Validation accuracy has reached upto 90%\
            so, stopping further training.')
            self.model.stop_training = True

es = EarlyStopping(patience = 3,
                           monitor = 'val_accuracy',
                           restore_best_weights = True)

lr = ReduceLROnPlateau(monitor = 'val_loss',
                                  patience = 2,
                                  factor = 0.5,
                                  verbose = 1)
```

Now we will train our model:

In [ ]:

```
history = model.fit(X_train, Y_train,
                                  validation_data = (X_val, Y_val),
                                  batch_size = BATCH_SIZE,
                                  epochs = EPOCHS,
                                  verbose = 1,
```

```
                                        callbacks = [es, lr,
myCallback()])
```

```
In [*]: history = model.fit(X_train, Y_train,
            validation_data = (X_val, Y_val),
            batch_size = BATCH_SIZE,
            epochs = EPOCHS,
            verbose = 1,
            callbacks = [es, lr, myCallback()])

Epoch 1/10
188/188 [==============================] - 1276s 7s/step - loss: 0.2236 - accuracy: 0.9109 - val_loss: 1.2079 - val_accuracy:
0.6883 - lr: 0.0010
Epoch 2/10
188/188 [==============================] - 808s 4s/step - loss: 0.1705 - accuracy: 0.9326 - val_loss: 0.7515 - val_accuracy: 0.
7723 - lr: 0.0010
Epoch 3/10
188/188 [==============================] - 806s 4s/step - loss: 0.1517 - accuracy: 0.9421 - val_loss: 1.7885 - val_accuracy: 0.
6003 - lr: 0.0010
Epoch 4/10
188/188 [==============================] - ETA: 0s - loss: 0.1200 - accuracy: 0.9548
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
188/188 [==============================] - 841s 4s/step - loss: 0.1200 - accuracy: 0.9548 - val_loss: 1.0540 - val_accuracy: 0.
7617 - lr: 0.0010
Epoch 5/10
188/188 [==============================] - 814s 4s/step - loss: 0.0798 - accuracy: 0.9718 - val_loss: 0.4553 - val_accuracy: 0.
8807 - lr: 5.0000e-04
Epoch 6/10
188/188 [==============================] - ETA: 0s - loss: 0.0619 - accuracy: 0.9768
```

Let's visualize the training and validation accuracy with each epoch.

In [ ]:

```
history_df = pd.DataFrame(history.history)
history_df.loc[:,['loss','val_loss']].plot()
history_df.loc[:,['accuracy','val_accuracy']].plot()
plt.show()
```

Model Evaluation

In [ ]:

```
Y_pred = model.predict(X_val)

Y_val = np.argmax(Y_val, axis=1)
Y_pred = np.argmax(Y_pred, axis=1)
```

Let's draw the confusion metrics and classification report using the predicted labels and the true labels.

In [ ]:

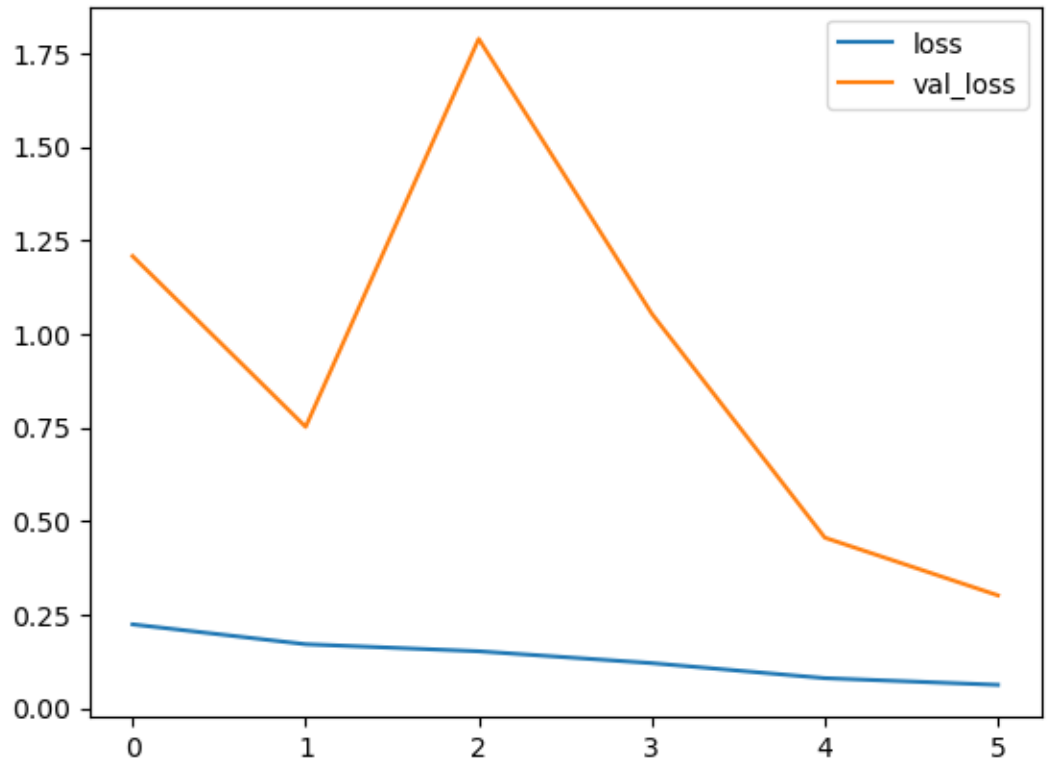
```
metrics.confusion_matrix(Y_val, Y_pred)
```

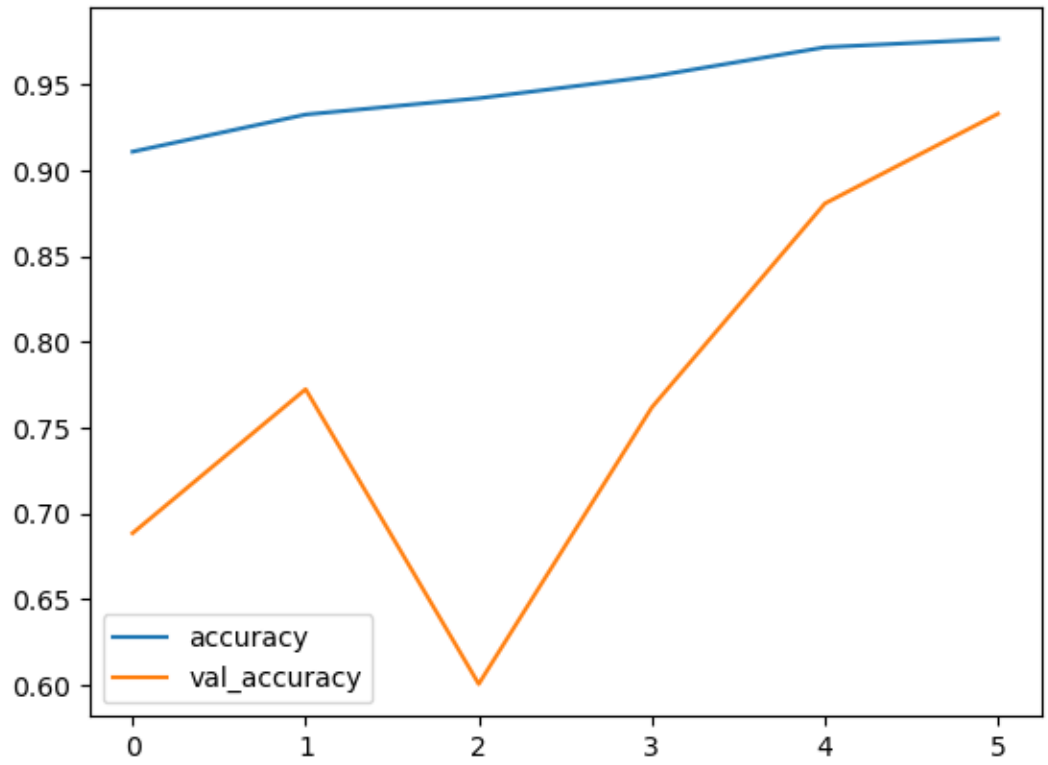
In [ ]:

```
print(metrics.classification_report(Y_val, Y_pred,
      target_names=classes))
```

Results

visualize the training and validation accuracy with each epoch





From the above graphs, we can certainly say that the model has not overfitted the training data as the difference between the training and validation accuracy is very low.

Model Evaluation

Now as we have our model ready let's evaluate its performance on the validation data using different metrics. For this purpose, we will first predict the class for the validation data using this model and then compare the output with the true labels.

Conclusion:

Indeed the performance of our model using the Transfer Learning Technique has achieved higher accuracy without overfitting which is very good as the f1-score for each class is also above 0.90 which means our model's prediction is correct 90% of the time.

Reference:

https://www.kaggle.com/datasets/andrewmvd/lung-and-colon-cancer-histopathological-images