Creating CNN Using Scratch And Transfer Learning

Name : Prathamesh Nale

prn: 2122000107

Roll No : DS13

```
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).
```

```
# import the libraries as shown below

from tensorflow.keras.layers import Input, Lambda, Dense, Flatten,Conv2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
from tensorflow.keras.applications.vgg19 import VGG19
```

```
# re-size all the images to this
IMAGE_SIZE = [224, 224]

train_path = '/content/drive/Mydrive/Maleria_Dataset/Maleria_Dataset/Train/*'
valid_path = '/content/drive/Mydrive/Maleria_Dataset/Maleria_Dataset/Test/*'
```

```
# Import the Vgg 16 library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights

mobilnet = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```
# don't train existing weights
for layer in mobilnet.layers:
    layer.trainable = False
```

```
  # useful for getting number of output classes
folders = glob('/content/drive/MyDrive/Malaria-Detection-master/Dataset/Dataset/Train')
```

```
!ls '/content/drive/MyDrive/Maleria_Dataset/Maleria_Dataset/Train'
```

```
ls: cannot access '/content/drive/MyDrive/Maleria_Dataset/Maleria_Dataset/Train': No such file or directory
```

```
folders
```

```
[]
```

```
# our layers - you can add more if you want
x = Flatten()(mobilnet.output)
```

```
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=mobilnet.input, outputs=prediction)
```

```
# view the structure of the model
model.summary()
```

```
Model: "functional"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
┃ Layer (type)                         ┃ Output Shape                ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
│ input_layer (InputLayer)             │ (None, 224, 224, 3)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block1_conv1 (Conv2D)                │ (None, 224, 224, 64)        │
├──────────────────────────────────────┼─────────────────────────────┼
│ block1_conv2 (Conv2D)                │ (None, 224, 224, 64)        │
├──────────────────────────────────────┼─────────────────────────────┼
│ block1_pool (MaxPooling2D)           │ (None, 112, 112, 64)        │
├──────────────────────────────────────┼─────────────────────────────┼
│ block2_conv1 (Conv2D)                │ (None, 112, 112, 128)       │
├──────────────────────────────────────┼─────────────────────────────┼
│ block2_conv2 (Conv2D)                │ (None, 112, 112, 128)       │
├──────────────────────────────────────┼─────────────────────────────┼
│ block2_pool (MaxPooling2D)           │ (None, 56, 56, 128)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block3_conv1 (Conv2D)                │ (None, 56, 56, 256)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block3_conv2 (Conv2D)                │ (None, 56, 56, 256)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block3_conv3 (Conv2D)                │ (None, 56, 56, 256)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block3_pool (MaxPooling2D)           │ (None, 28, 28, 256)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block4_conv1 (Conv2D)                │ (None, 28, 28, 512)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block4_conv2 (Conv2D)                │ (None, 28, 28, 512)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block4_conv3 (Conv2D)                │ (None, 28, 28, 512)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block4_pool (MaxPooling2D)           │ (None, 14, 14, 512)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block5_conv1 (Conv2D)                │ (None, 14, 14, 512)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block5_conv2 (Conv2D)                │ (None, 14, 14, 512)         │
├──────────────────────────────────────┼─────────────────────────────┼
│ block5_conv3 (Conv2D)                │ (None, 14, 14, 512)         │
```

```
├──────────────────────────────────┼─────────────────────────────┼───
from tensorflow.keras.layers import MaxPooling2D
│ flatten (Flatten)               │ (None, 25088)               │
```

```
### Create Model from scratch using CNN
model=Sequential()
model.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation ="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dense(500,activation="relu"))
model.add(Dense(2,activation="softmax"))
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequent
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                      ┃ Output Shape                  ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                   │ (None, 224, 224, 16)          │             208 │
├───────────────────────────────────┼───────────────────────────────┼─────────────────┤
│ max_pooling2d (MaxPooling2D)      │ (None, 112, 112, 16)          │               0 │
├───────────────────────────────────┼───────────────────────────────┼─────────────────┤
│ conv2d_1 (Conv2D)                 │ (None, 112, 112, 32)          │           2,080 │
├───────────────────────────────────┼───────────────────────────────┼─────────────────┤
│ max_pooling2d_1 (MaxPooling2D)    │ (None, 56, 56, 32)            │               0 │
├───────────────────────────────────┼───────────────────────────────┼─────────────────┤
│ conv2d_2 (Conv2D)                 │ (None, 56, 56, 64)            │           8,256 │
├───────────────────────────────────┼───────────────────────────────┼─────────────────┤
│ max_pooling2d_2 (MaxPooling2D)    │ (None, 28, 28, 64)            │               0 │
├───────────────────────────────────┼───────────────────────────────┼─────────────────┤
│ flatten_1 (Flatten)               │ (None, 50176)                 │               0 │
├───────────────────────────────────┼───────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                   │ (None, 500)                   │      25,088,500 │
├───────────────────────────────────┼───────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                   │ (None, 2)                     │           1,002 │
└───────────────────────────────────┴───────────────────────────────┴─────────────────┘
 Total params: 25,100,046 (95.75 MB)
 Trainable params: 25,100,046 (95.75 MB)
 Non-trainable params: 0 (0.00 B)
```

```
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

```
# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

#test_datagen = ImageDataGenerator(rescale = 1./255)

test_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
```

```
train_datagen
```

```
<keras.src.legacy.preprocessing.image.ImageDataGenerator at 0x7a3591787f70>
```

```
test_datagen
```

```
<keras.src.legacy.preprocessing.image.ImageDataGenerator at 0x7a3591784310>
```

```
# Make sure you provide the same target size as initialied for the image size
training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/Colab Notebooks/DL/Maleria_Dataset/Train',target_size = (224, 224),batch_size = 32,class_mode = 'categorical')
```

```
Found 436 images belonging to 2 classes.
```

```
training_set
```

```
<keras.src.legacy.preprocessing.image.DirectoryIterator at 0x7a359195eb60>
```

```
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/Colab Notebooks/DL/Maleria_Dataset/Test',target_size = (224, 224),batch_size = 32,class_mode = 'categorical')
```

```
Found 134 images belonging to 2 classes.
```

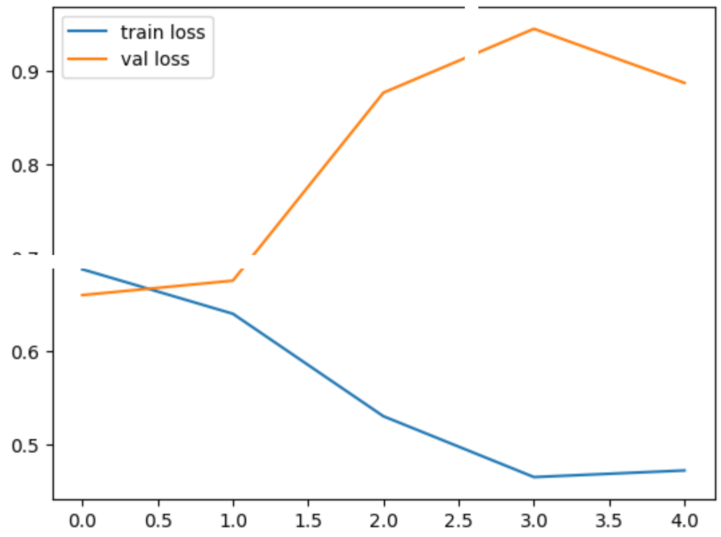
```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

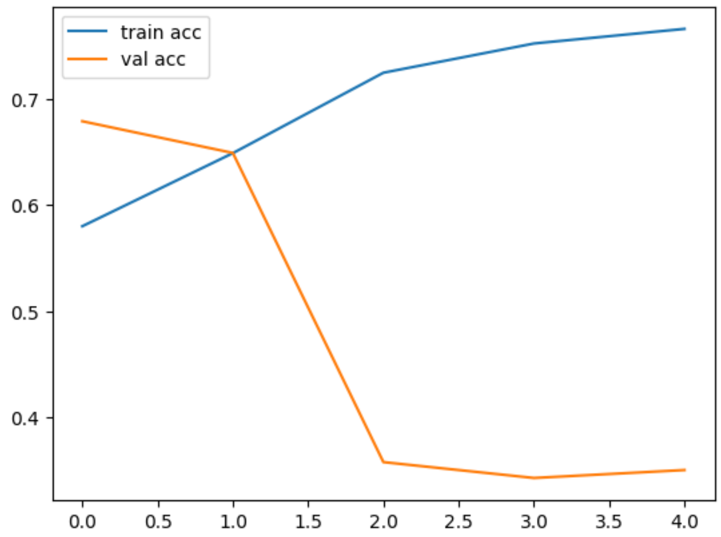
```
r = model.fit(training_set,validation_data=test_set,epochs=5)
```

```
Epoch 1/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 90s 6s/step - accuracy: 0.5274 - loss: 0.7039 - val_accuracy: 0.6791 - val_loss: 0.6597
Epoch 2/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 43s 3s/step - accuracy: 0.6207 - loss: 0.6578 - val_accuracy: 0.6493 - val_loss: 0.6752
Epoch 3/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 43s 3s/step - accuracy: 0.6990 - loss: 0.5603 - val_accuracy: 0.3582 - val_loss: 0.8765
Epoch 4/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 79s 2s/step - accuracy: 0.7313 - loss: 0.4998 - val_accuracy: 0.3433 - val_loss: 0.9449
Epoch 5/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 38s 2s/step - accuracy: 0.7697 - loss: 0.4660 - val_accuracy: 0.3507 - val_loss: 0.8869
```

```
# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc'
plt.legend()
plt.show()
```





```
<Figure size 640x480 with 0 Axes>
```

```
# save it as a h5 file


from tensorflow.keras.models import load_model

model.save('model_vgg16.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the na
```

```
y_pred = model.predict(test_set)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 725ms/step
```

```
y_pred
```

```
       [0.41696718, 0.5830328 ],
       [0.24578376, 0.7542162 ],
       [0.5290245 , 0.4709755 ],
       [0.33541575, 0.6645844 ],
       [0.5689765 , 0.43102345],
       [0.23995136, 0.7600486 ],
       [0.33186072, 0.6681392 ],
       [0.27582282, 0.72417706],
       [0.3304841 , 0.6695159 ],
       [0.74073106, 0.2592689 ],
       [0.53563803, 0.46436197],
       [0.35182658, 0.6481735 ],
       [0.27926123, 0.72073877],
       [0.16532348, 0.8346765 ],
       [0.6355352 , 0.36446482],
       [0.4428313 , 0.55716866],
       [0.5145273 , 0.48547265],
       [0.63661313, 0.36338675],
       [0.6203091 , 0.37969092],
       [0.44427907, 0.5557209 ],
       [0.6925833 , 0.30741677],
       [0.48939517, 0.5106049 ],
       [0.28295138, 0.7170485 ],
       [0.2770329 , 0.722967  ],
       [0.6990246 , 0.30097538],
       [0.5825912 , 0.41740885],
       [0.5628534 , 0.43714663],
       [0.6544954 , 0.34550473],
       [0.49879366, 0.5012063 ],
       [0.26671433, 0.73328567],
       [0.25842398, 0.741576  ],
       [0.25314912, 0.7468508 ],
       [0.16767254, 0.8323274 ],
       [0.42634514, 0.5736548 ],
       [0.4131232 , 0.58687687],
       [0.3384899 , 0.6615102 ],
       [0.3152123 , 0.6847877 ],
       [0.38767555, 0.61232454],
       [0.53670365, 0.4632964 ],
       [0.2639973 , 0.7360026 ],
       [0.47945556, 0.5205444 ],
       [0.5887256 , 0.41127437],
       [0.20196655, 0.79803336],
       [0.68852615, 0.31147394],
       [0.56993717, 0.43006277],
       [0.547982  , 0.45201808],
       [0.41249707, 0.5875029 ],
       [0.61323535, 0.3867647 ],
       [0.21236026, 0.78763974],
       [0.627871  , 0.37212902],
       [0.10749181, 0.8925082 ],
       [0.6101282 , 0.38987175],
       [0.16617587, 0.8338241 ],
       [0.22547197, 0.77452797],
       [0.46784997, 0.53215003],
       [0.6384137 , 0.36158633],
       [0.33974844, 0.66025156],
       [0.37919483, 0.62080514],
       [0.58618   , 0.41381997],
import numpy as np
y_pred = np.argmax(y_pred, axis=1)
```

```
y_pred
```

```
array([0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 0])
```

Start coding or generate with AI.

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

```
model=load_model('model_vgg16.h5')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

Start coding or generate with AI.

```
img=image.load_img('/content/drive/MyDrive/Colab Notebooks/DL/Maleria_Dataset/Test/Uninfected/2.png',target_size=(224,224))
```

```
x=image.img_to_array(img)
x
```

```
array([[[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        ...,
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        ...,
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        ...,
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       ...,

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        ...,
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        ...,
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        ...,
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]], dtype=float32)
```

```
x.shape
```

```
(224, 224, 3)
```

```
x=x/255
```

```
x=np.expand_dims(x,axis=0)
img_data=preprocess_input(x)
img_data.shape
```

```
(1, 224, 224, 3)
```

```
model.predict(img_data)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
array([[1., 0.]], dtype=float32)
```

```
a=np.argmax(model.predict(img_data), axis=1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
```

```
if(a==0):
    print("Uninfected")
else:
    print("Infected")
```

```
Uninfected
```

```
mobilnet2 = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

```
for layer in mobilnet2.layers:
    layer.trainable = False
```

```
folders2 = glob('/content/drive/MyDrive/Malaria-Detection-master/Dataset/Dataset/Train')
```

```
folders2
```

```
[]
```

```
x = Flatten()(mobilnet.output)
```

```
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model2 = Model(inputs=mobilnet.input, outputs=prediction)
```

```
model2.summary()
```

```
Model: "functional_20"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                       ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)           │ (None, 224, 224, 3)         │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block1_conv1 (Conv2D)              │ (None, 224, 224, 64)        │           1,792 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block1_conv2 (Conv2D)              │ (None, 224, 224, 64)        │          36,928 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block1_pool (MaxPooling2D)         │ (None, 112, 112, 64)        │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block2_conv1 (Conv2D)              │ (None, 112, 112, 128)       │          73,856 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block2_conv2 (Conv2D)              │ (None, 112, 112, 128)       │         147,584 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block2_pool (MaxPooling2D)         │ (None, 56, 56, 128)         │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block3_conv1 (Conv2D)              │ (None, 56, 56, 256)         │         295,168 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block3_conv2 (Conv2D)              │ (None, 56, 56, 256)         │         590,080 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block3_conv3 (Conv2D)              │ (None, 56, 56, 256)         │         590,080 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block3_pool (MaxPooling2D)         │ (None, 28, 28, 256)         │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block4_conv1 (Conv2D)              │ (None, 28, 28, 512)         │       1,180,160 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block4_conv2 (Conv2D)              │ (None, 28, 28, 512)         │       2,359,808 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block4_conv3 (Conv2D)              │ (None, 28, 28, 512)         │       2,359,808 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block4_pool (MaxPooling2D)         │ (None, 14, 14, 512)         │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block5_conv1 (Conv2D)              │ (None, 14, 14, 512)         │       2,359,808 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block5_conv2 (Conv2D)              │ (None, 14, 14, 512)         │       2,359,808 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block5_conv3 (Conv2D)              │ (None, 14, 14, 512)         │       2,359,808 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block5_pool (MaxPooling2D)         │ (None, 7, 7, 512)           │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_2 (Flatten)                │ (None, 25088)               │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_4 (Dense)                    │ (None, 0)                   │               0 │
└────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 14,714,688 (56.13 MB)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 14,714,688 (56.13 MB)
```

```
model2=Sequential()
model2.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",input_shape=(224,224,3)))
model2.add(MaxPooling2D(pool_size=2))
model2.add(Conv2D(filters=32,kernel_size=2,padding="same",activation ="relu"))
model2.add(MaxPooling2D(pool_size=2))
model2.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model2.add(MaxPooling2D(pool_size=2))
model2.add(Flatten())
model2.add(Dense(500,activation="relu"))
model2.add(Dense(2,activation="softmax"))
model2.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequent
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                       ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d_3 (Conv2D)                  │ (None, 224, 224, 16)        │             208 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_3 (MaxPooling2D)     │ (None, 112, 112, 16)        │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_4 (Conv2D)                  │ (None, 112, 112, 32)        │           2,080 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_4 (MaxPooling2D)     │ (None, 56, 56, 32)          │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_5 (Conv2D)                  │ (None, 56, 56, 64)          │           8,256 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_5 (MaxPooling2D)     │ (None, 28, 28, 64)          │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_3 (Flatten)                │ (None, 50176)               │               0 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_5 (Dense)                    │ (None, 500)                 │      25,088,500 │
├────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_6 (Dense)                    │ (None, 2)                   │           1,002 │
└────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 25,100,046 (95.75 MB)
 Trainable params: 25,100,046 (95.75 MB)
 Non-trainable params: 0 (0.00 B)
```

```
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
f = model2.fit(training_set,validation_data=test_set,epochs=5)
```

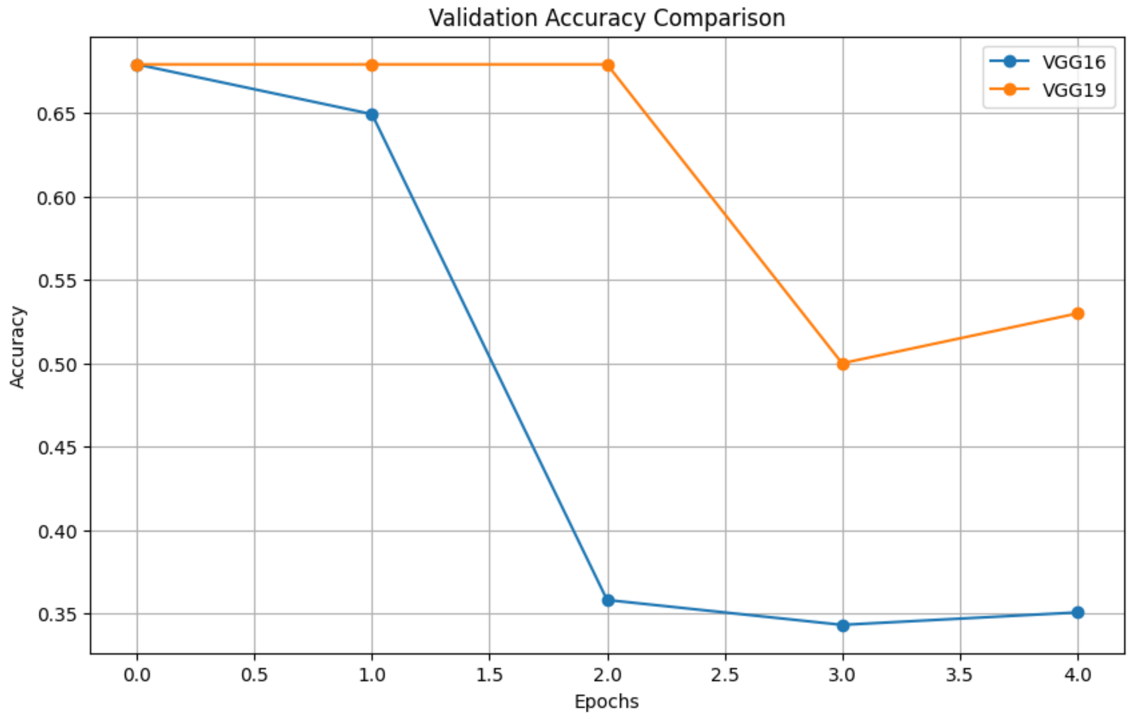
```
Epoch 1/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 50s 3s/step - accuracy: 0.5110 - loss: 2.6884 - val_accuracy: 0.6791 - val_loss: 0.6478
Epoch 2/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 46s 3s/step - accuracy: 0.5631 - loss: 0.6885 - val_accuracy: 0.6791 - val_loss: 0.6818
Epoch 3/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 73s 2s/step - accuracy: 0.5801 - loss: 0.6756 - val_accuracy: 0.6791 - val_loss: 0.6500
Epoch 4/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.6183 - loss: 0.6258 - val_accuracy: 0.5000 - val_loss: 0.7094
Epoch 5/5
14/14 ━━━━━━━━━━━━━━━━━━━━ 40s 3s/step - accuracy: 0.7114 - loss: 0.5754 - val_accuracy: 0.5299 - val_loss: 0.7061
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
plt.figure(figsize=(10, 6))
plt.plot(r.history['val_accuracy'], label='VGG16', marker='o')
plt.plot(f.history['val_accuracy'], label='VGG19', marker='o')
plt.title('Validation Accuracy Comparison')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
plt.show()
```



Start coding or generate with AI.