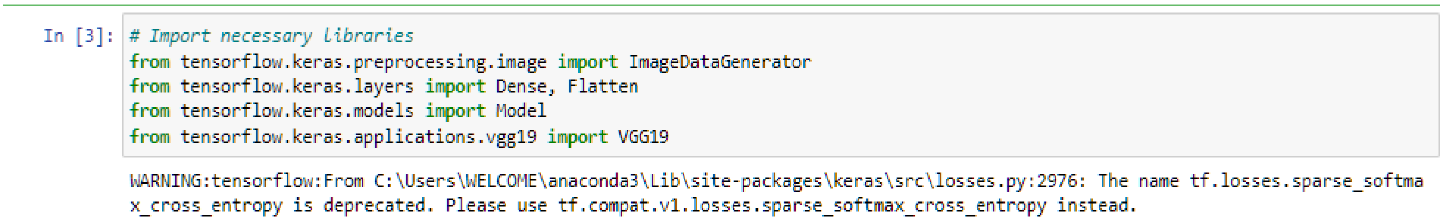
7. CODING AND SOLUTIONING

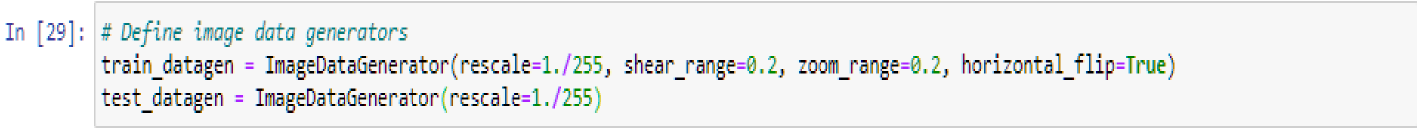
7.1 Feature 1: VGG19 MODEL

# 1.Import the necessary libraries

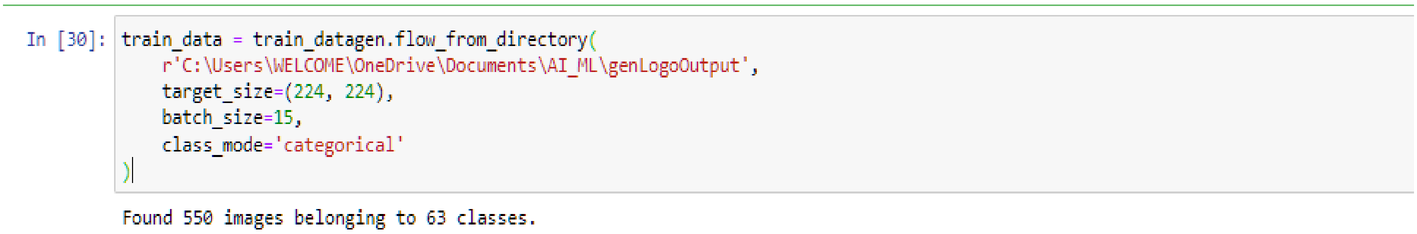


```
In [3]: # Import necessary libraries
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.layers import Dense, Flatten
        from tensorflow.keras.models import Model
        from tensorflow.keras.applications.vgg19 import VGG19

        WARNING:tensorflow:From C:\Users\WELCOME\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softma
        x_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

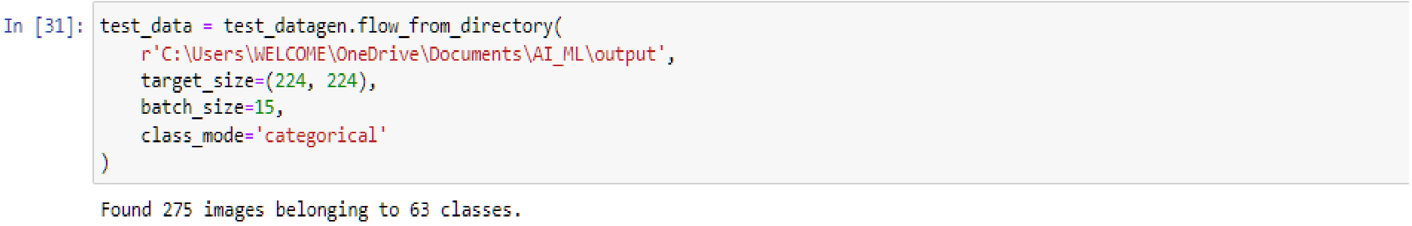
# 2.Define Image data generators



```
In [29]: # Define image data generators
         train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
         test_datagen = ImageDataGenerator(rescale=1./255)
```

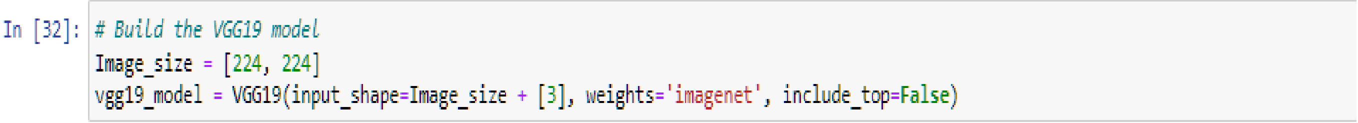


```
In [30]: train_data = train_datagen.flow_from_directory(
             r'C:\Users\WELCOME\OneDrive\Documents\AI_ML\genLogoOutput',
             target_size=(224, 224),
             batch_size=15,
             class_mode='categorical'
         )

         Found 550 images belonging to 63 classes.
```



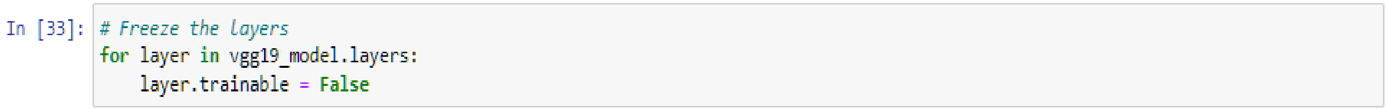
```
In [31]: test_data = test_datagen.flow_from_directory(
             r'C:\Users\WELCOME\OneDrive\Documents\AI_ML\output',
             target_size=(224, 224),
             batch_size=15,
             class_mode='categorical'
         )

         Found 275 images belonging to 63 classes.
```

# 3.Build the VGG19 Model



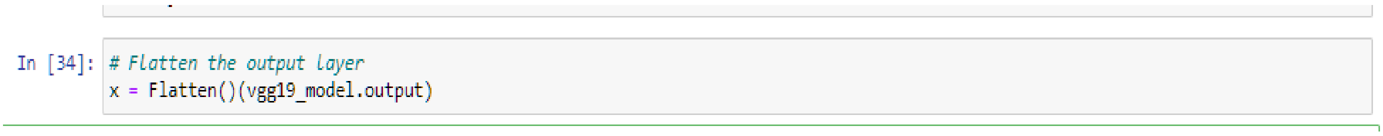
```
In [32]: # Build the VGG19 model
         Image_size = [224, 224]
         vgg19_model = VGG19(input_shape=Image_size + [3], weights='imagenet', include_top=False)
```

# 4.Freeze the layers



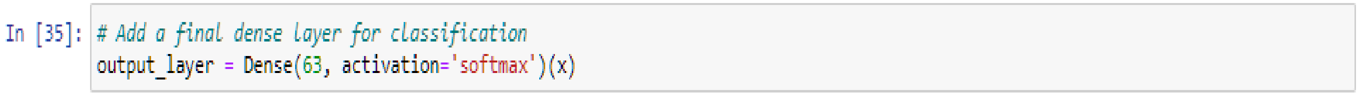
```
In [33]: # Freeze the layers
         for layer in vgg19_model.layers:
             layer.trainable = False
```

# 5.Flatten the output layer



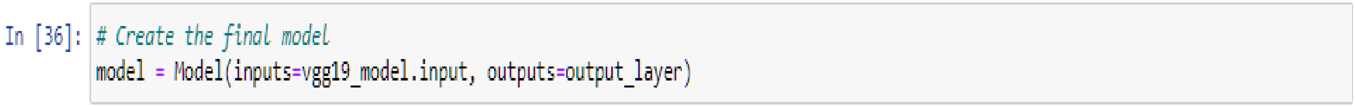
```
In [34]: # Flatten the output layer
         x = Flatten()(vgg19_model.output)
```

# 6.Add a final dense layer for classification



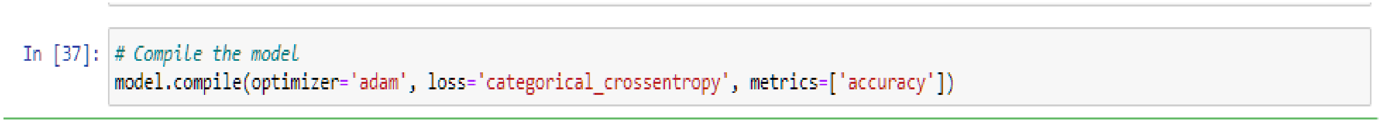
```
In [35]: # Add a final dense layer for classification
         output_layer = Dense(63, activation='softmax')(x)
```

# 7.Create the final model



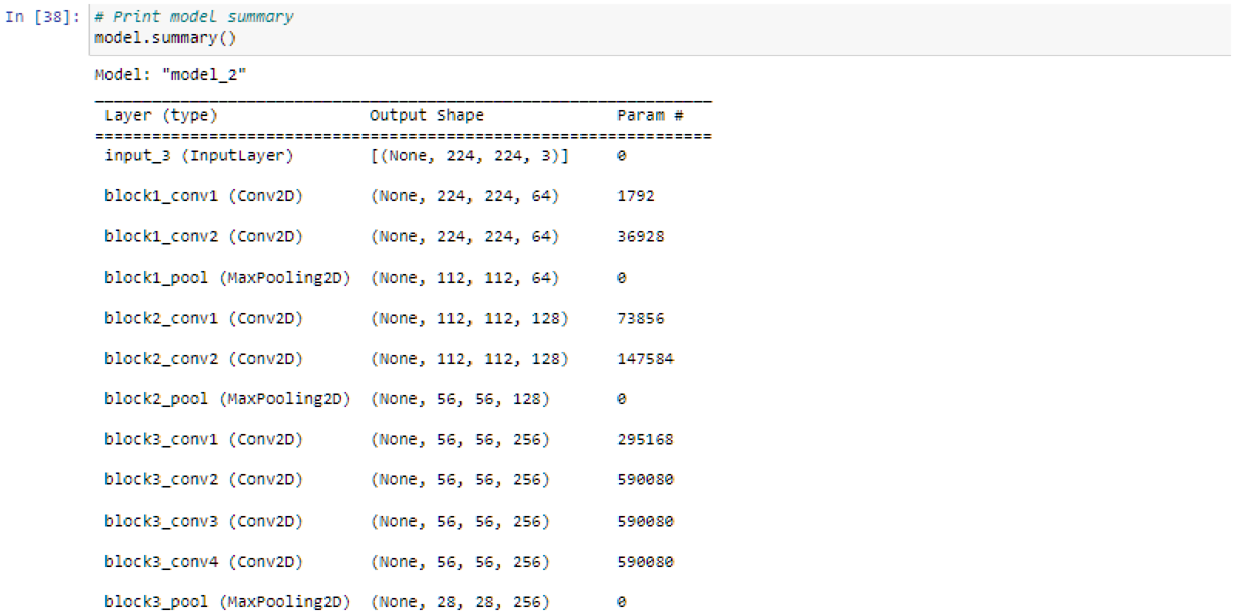
```
In [36]: # Create the final model
         model = Model(inputs=vgg19_model.input, outputs=output_layer)
```

# 8.Compile the model

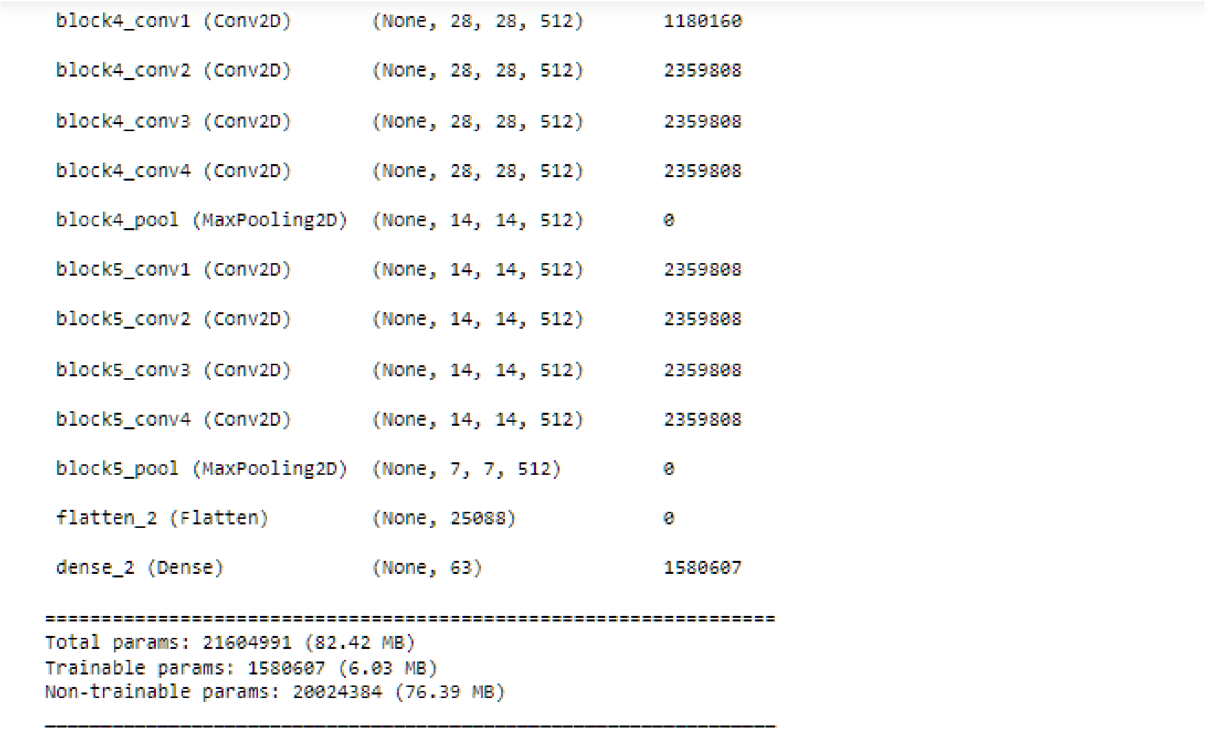


```
In [37]: # Compile the model
         model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

# 9.Print model summary

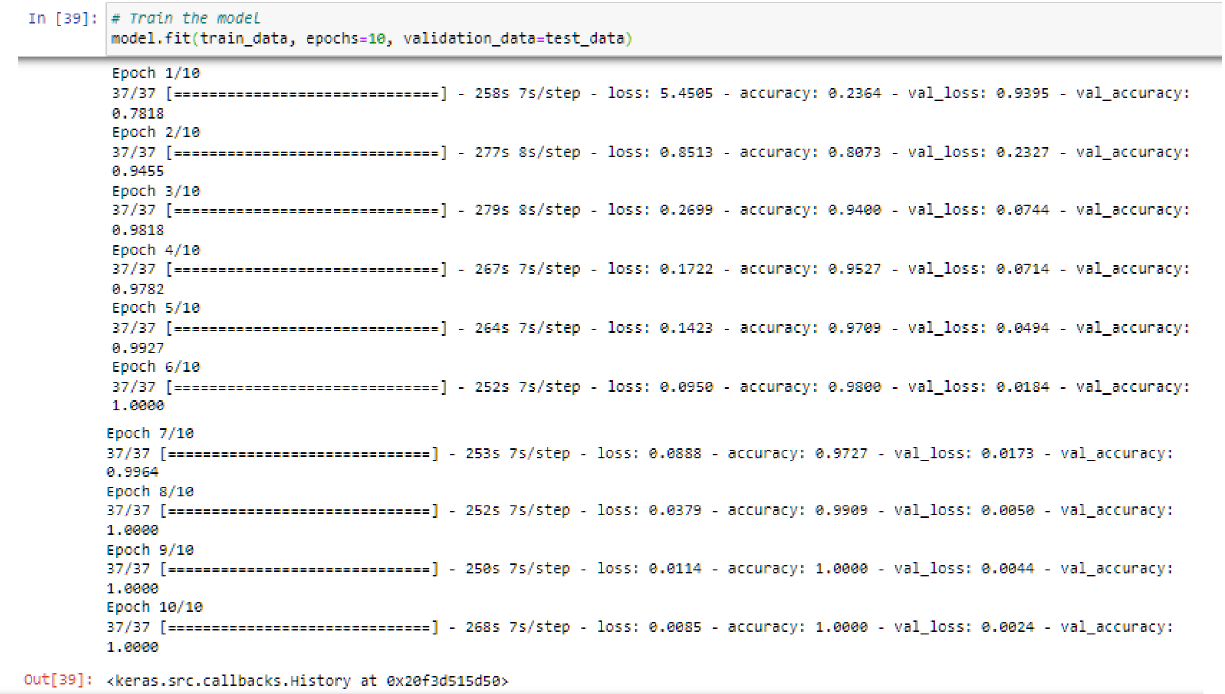


```
In [38]: # Print model summary
         model.summary()

Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
```



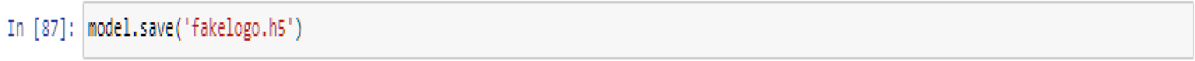
```
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv4 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv4 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten_2 (Flatten)         (None, 25088)             0

 dense_2 (Dense)             (None, 63)                1580607

=================================================================
Total params: 21604991 (82.42 MB)
Trainable params: 1580607 (6.03 MB)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
```

# 10.Train the Model

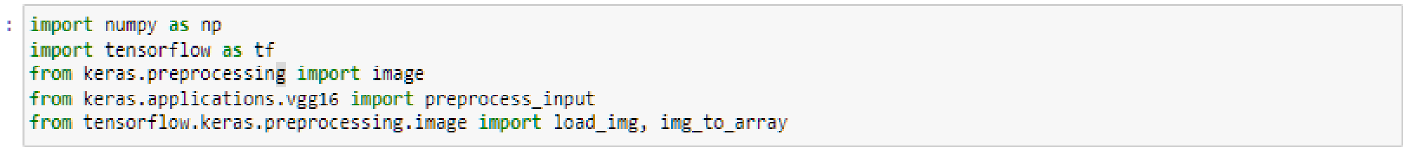


```
In [39]: # Train the model
         model.fit(train_data, epochs=10, validation_data=test_data)

Epoch 1/10
37/37 [==============================] - 258s 7s/step - loss: 5.4505 - accuracy: 0.2364 - val_loss: 0.9395 - val_accuracy:
0.7818
Epoch 2/10
37/37 [==============================] - 277s 8s/step - loss: 0.8513 - accuracy: 0.8073 - val_loss: 0.2327 - val_accuracy:
0.9455
Epoch 3/10
37/37 [==============================] - 279s 8s/step - loss: 0.2699 - accuracy: 0.9400 - val_loss: 0.0744 - val_accuracy:
0.9818
Epoch 4/10
37/37 [==============================] - 267s 7s/step - loss: 0.1722 - accuracy: 0.9527 - val_loss: 0.0714 - val_accuracy:
0.9782
Epoch 5/10
37/37 [==============================] - 264s 7s/step - loss: 0.1423 - accuracy: 0.9709 - val_loss: 0.0494 - val_accuracy:
0.9927
Epoch 6/10
37/37 [==============================] - 252s 7s/step - loss: 0.0950 - accuracy: 0.9800 - val_loss: 0.0184 - val_accuracy:
1.0000

Epoch 7/10
37/37 [==============================] - 253s 7s/step - loss: 0.0888 - accuracy: 0.9727 - val_loss: 0.0173 - val_accuracy:
0.9964
Epoch 8/10
37/37 [==============================] - 252s 7s/step - loss: 0.0379 - accuracy: 0.9909 - val_loss: 0.0050 - val_accuracy:
1.0000
Epoch 9/10
37/37 [==============================] - 250s 7s/step - loss: 0.0114 - accuracy: 1.0000 - val_loss: 0.0044 - val_accuracy:
1.0000
Epoch 10/10
37/37 [==============================] - 268s 7s/step - loss: 0.0085 - accuracy: 1.0000 - val_loss: 0.0024 - val_accuracy:
1.0000

Out[39]: <keras.src.callbacks.History at 0x20f3d515d50>
```

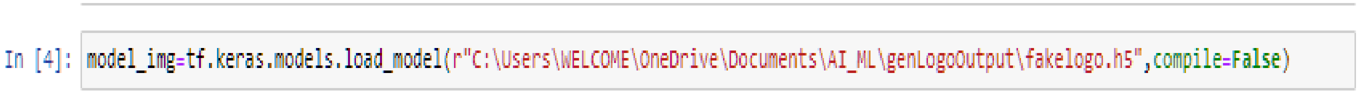
# 11.Save the Model



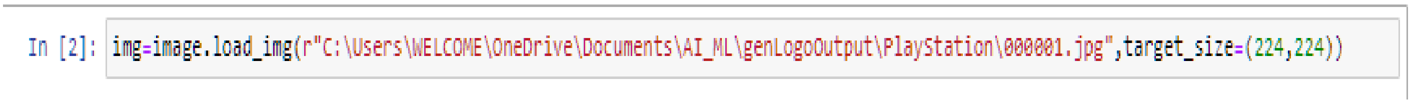
```
In [87]: model.save('fakelogo.h5')
```



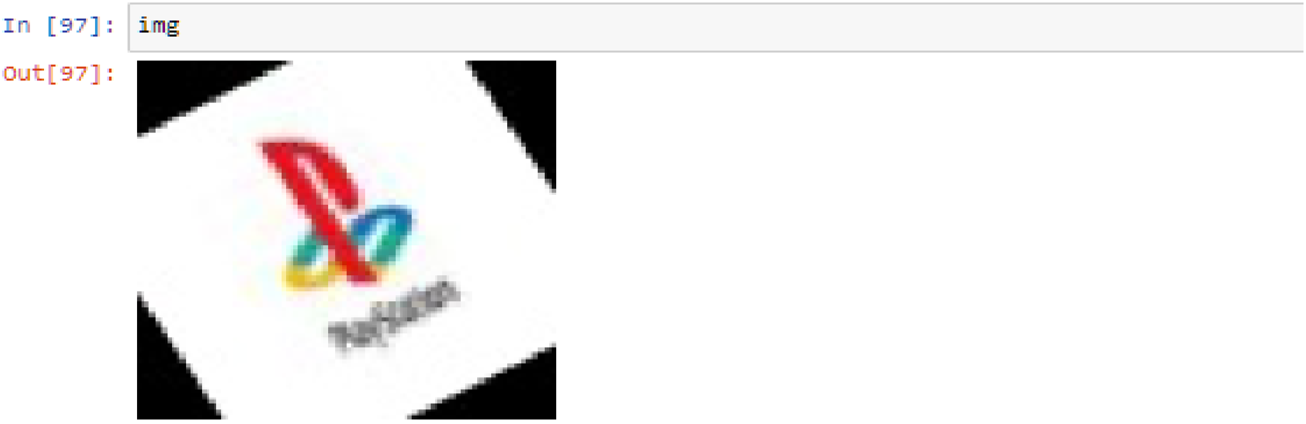
```
: import numpy as np
  import tensorflow as tf
  from keras.preprocessing import image
  from keras.applications.vgg16 import preprocess_input
  from tensorflow.keras.preprocessing.image import load_img, img_to_array
```



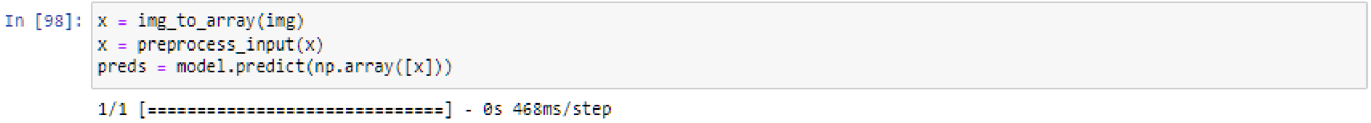
```
In [4]: model_img=tf.keras.models.load_model(r"C:\Users\WELCOME\OneDrive\Documents\AI_ML\genLogoOutput\fakelogo.h5",compile=False)
```



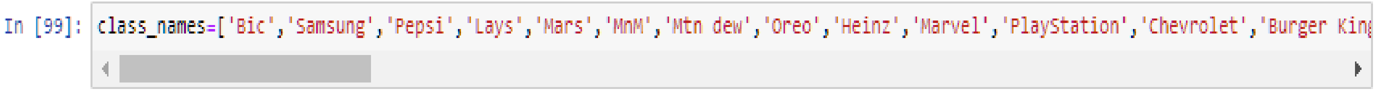
```
In [2]: img=image.load_img(r"C:\Users\WELCOME\OneDrive\Documents\AI_ML\genLogoOutput\PlayStation\000001.jpg",target_size=(224,224))
```



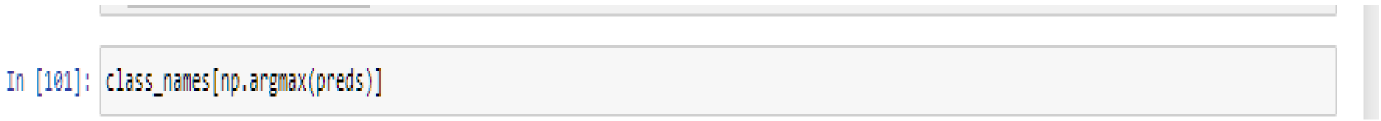
```
In [97]: img
Out[97]:
```



```
In [98]: x = img_to_array(img)
         x = preprocess_input(x)
         preds = model.predict(np.array([x]))

         1/1 [==============================] - 0s 468ms/step
```



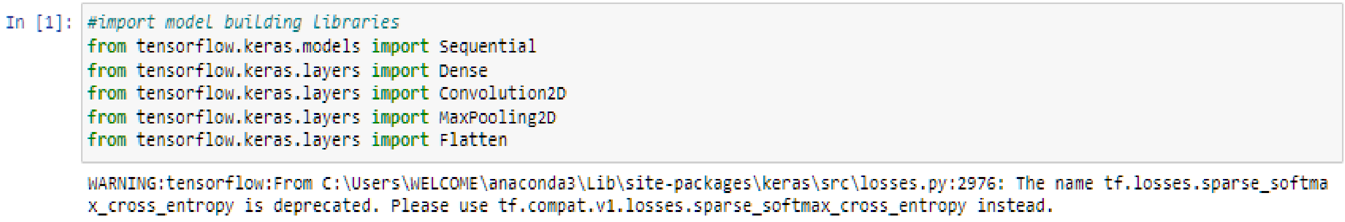
```
In [99]: class_names=['Bic','Samsung','Pepsi','Lays','Mars','MnM','Mtn dew','Oreo','Heinz','Marvel','PlayStation','Chevrolet','Burger Kin
```



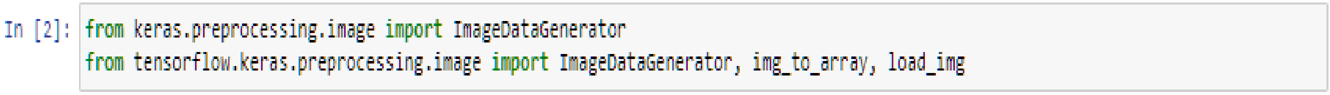
```
In [101]: class_names[np.argmax(preds)]
```

7.2 Feature 2: CNN

# Import the necessary libraries

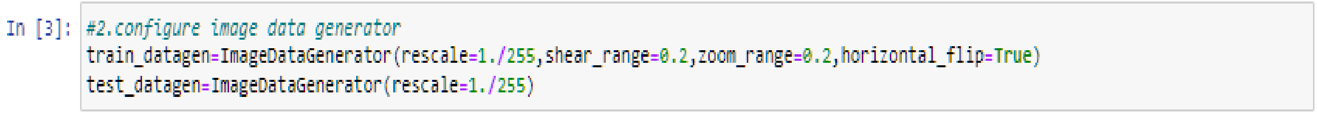


```
In [1]: #import model building libraries
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense
        from tensorflow.keras.layers import Convolution2D
        from tensorflow.keras.layers import MaxPooling2D
        from tensorflow.keras.layers import Flatten

        WARNING:tensorflow:From C:\Users\WELCOME\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softma
        x_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```



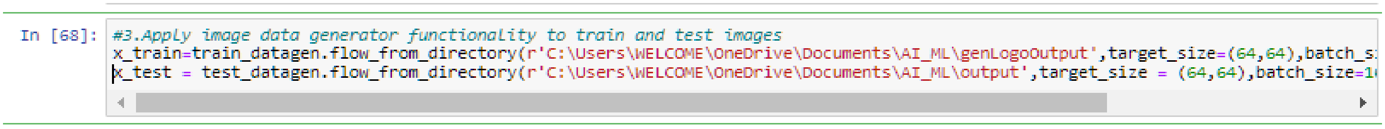
```
In [2]: from keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
```

# 2.Configure image data generator

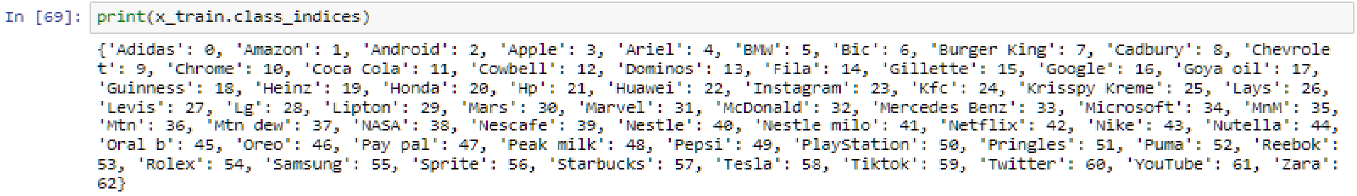


```
In [3]: #2.configure image data generator
        train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
        test_datagen=ImageDataGenerator(rescale=1./255)
```

3.Apply image data generator functionality to train and test images

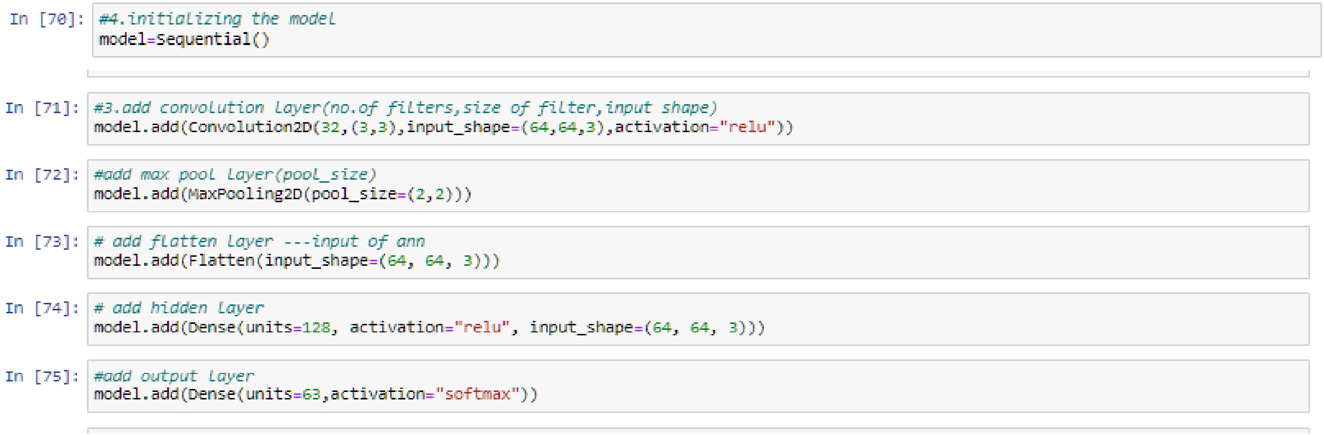


```
In [68]: #3.Apply image data generator functionality to train and test images
         x_train=train_datagen.flow_from_directory(r'C:\Users\WELCOME\OneDrive\Documents\AI_ML\genLogoOutput',target_size=(64,64),batch_s
         x_test = test_datagen.flow_from_directory(r'C:\Users\WELCOME\OneDrive\Documents\AI_ML\output',target_size = (64,64),batch_size=1
```



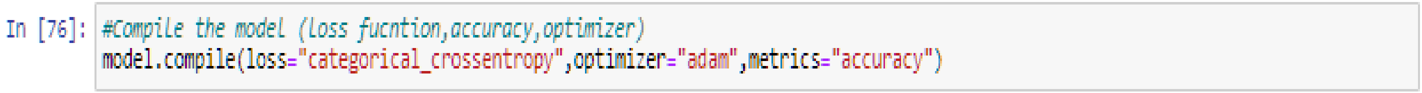
```
In [69]: print(x_train.class_indices)

         {'Adidas': 0, 'Amazon': 1, 'Android': 2, 'Apple': 3, 'Ariel': 4, 'BMW': 5, 'Bic': 6, 'Burger King': 7, 'Cadbury': 8, 'Chevrole
         t': 9, 'Chrome': 10, 'Coca Cola': 11, 'Cowbell': 12, 'Dominos': 13, 'Fila': 14, 'Gillette': 15, 'Google': 16, 'Goya oil': 17,
         'Guinness': 18, 'Heinz': 19, 'Honda': 20, 'Hp': 21, 'Huawei': 22, 'Instagram': 23, 'Kfc': 24, 'Krisspy Kreme': 25, 'Lays': 26,
         'Levis': 27, 'Lg': 28, 'Lipton': 29, 'Mars': 30, 'Marvel': 31, 'McDonald': 32, 'Mercedes Benz': 33, 'Microsoft': 34, 'MnM': 35,
         'Mtn': 36, 'Mtn dew': 37, 'NASA': 38, 'Nescafe': 39, 'Nestle': 40, 'Nestle milo': 41, 'Netflix': 42, 'Nike': 43, 'Nutella': 44,
         'Oral b': 45, 'Oreo': 46, 'Pay pal': 47, 'Peak milk': 48, 'Pepsi': 49, 'PlayStation': 50, 'Pringles': 51, 'Puma': 52, 'Reebok':
         53, 'Rolex': 54, 'Samsung': 55, 'Sprite': 56, 'Starbucks': 57, 'Tesla': 58, 'Tiktok': 59, 'Twitter': 60, 'YouTube': 61, 'Zara':
         62}
```

4.initializing the model



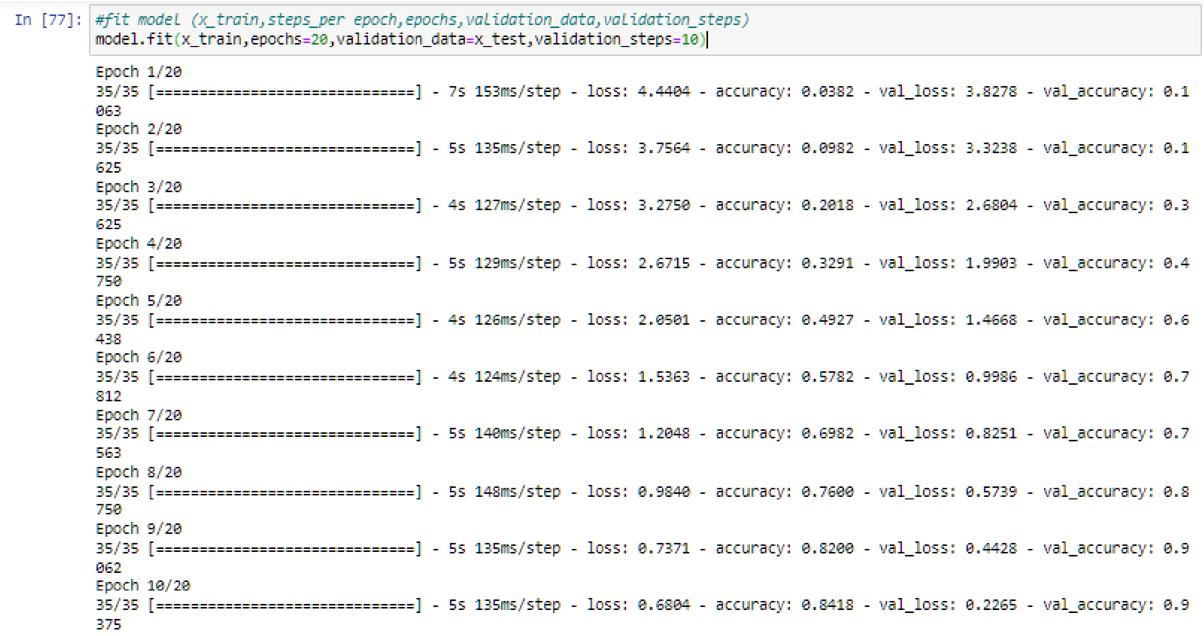
```
In [70]: #4.initializing the model
         model=Sequential()

In [71]: #3.add convolution layer(no.of filters,size of filter,input shape)
         model.add(Convolution2D(32,(3,3),input_shape=(64,64,3),activation="relu"))

In [72]: #add max pool layer(pool_size)
         model.add(MaxPooling2D(pool_size=(2,2)))

In [73]: # add flatten layer ---input of ann
         model.add(Flatten(input_shape=(64, 64, 3)))

In [74]: # add hidden layer
         model.add(Dense(units=128, activation="relu", input_shape=(64, 64, 3)))

In [75]: #add output layer
         model.add(Dense(units=63,activation="softmax"))
```

# 5.Compile the model (loss fucntion,accuracy,optimizer)

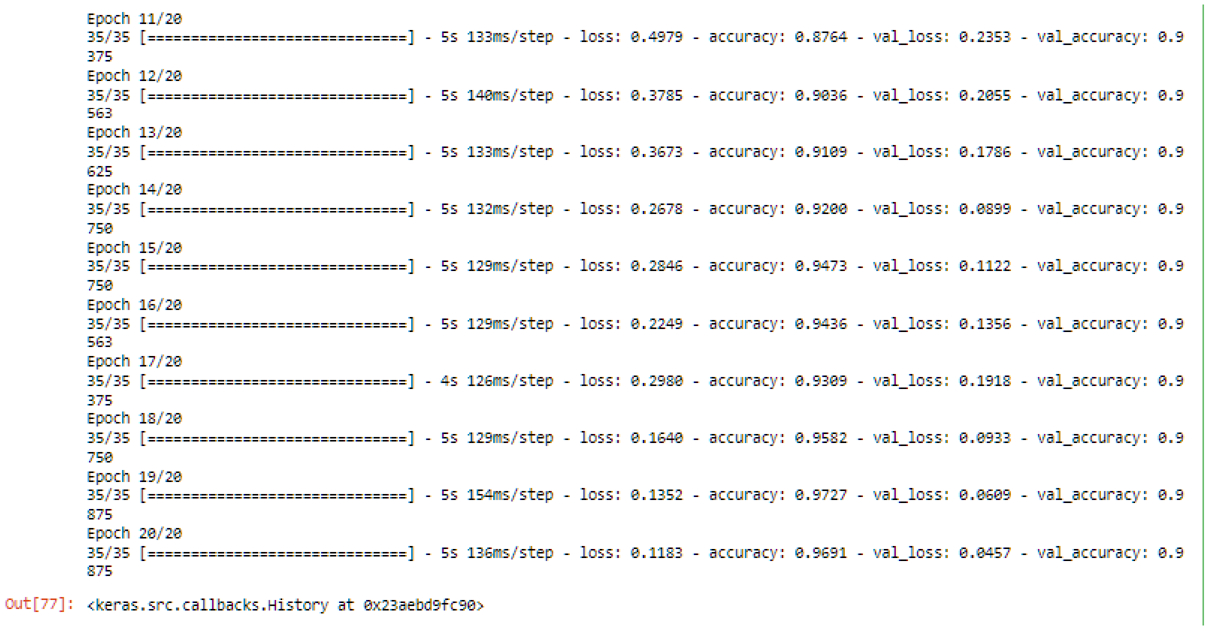


```
In [76]: #Compile the model (loss fucntion,accuracy,optimizer)
         model.compile(loss="categorical_crossentropy",optimizer="adam",metrics="accuracy")
```

# 6.fit model (x_train,steps_per epoch,epochs,validation_data,validation_steps)

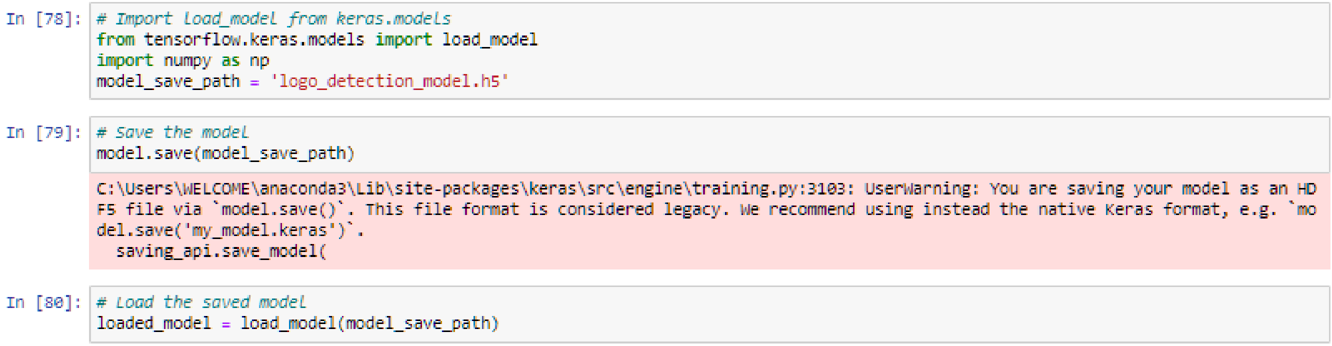


```
In [77]: #fit model (x_train,steps_per epoch,epochs,validation_data,validation_steps)
         model.fit(x_train,epochs=20,validation_data=x_test,validation_steps=10)

         Epoch 1/20
         35/35 [==============================] - 7s 153ms/step - loss: 4.4404 - accuracy: 0.0382 - val_loss: 3.8278 - val_accuracy: 0.1
         063
         Epoch 2/20
         35/35 [==============================] - 5s 135ms/step - loss: 3.7564 - accuracy: 0.0982 - val_loss: 3.3238 - val_accuracy: 0.1
         625
         Epoch 3/20
         35/35 [==============================] - 4s 127ms/step - loss: 3.2750 - accuracy: 0.2018 - val_loss: 2.6804 - val_accuracy: 0.3
         625
         Epoch 4/20
         35/35 [==============================] - 5s 129ms/step - loss: 2.6715 - accuracy: 0.3291 - val_loss: 1.9903 - val_accuracy: 0.4
         750
         Epoch 5/20
         35/35 [==============================] - 4s 126ms/step - loss: 2.0501 - accuracy: 0.4927 - val_loss: 1.4668 - val_accuracy: 0.6
         438
         Epoch 6/20
         35/35 [==============================] - 4s 124ms/step - loss: 1.5363 - accuracy: 0.5782 - val_loss: 0.9986 - val_accuracy: 0.7
         812
         Epoch 7/20
         35/35 [==============================] - 5s 140ms/step - loss: 1.2048 - accuracy: 0.6982 - val_loss: 0.8251 - val_accuracy: 0.7
         563
         Epoch 8/20
         35/35 [==============================] - 5s 148ms/step - loss: 0.9840 - accuracy: 0.7600 - val_loss: 0.5739 - val_accuracy: 0.8
         750
         Epoch 9/20
         35/35 [==============================] - 5s 135ms/step - loss: 0.7371 - accuracy: 0.8200 - val_loss: 0.4428 - val_accuracy: 0.9
         062
         Epoch 10/20
         35/35 [==============================] - 5s 135ms/step - loss: 0.6804 - accuracy: 0.8418 - val_loss: 0.2265 - val_accuracy: 0.9
         375
```

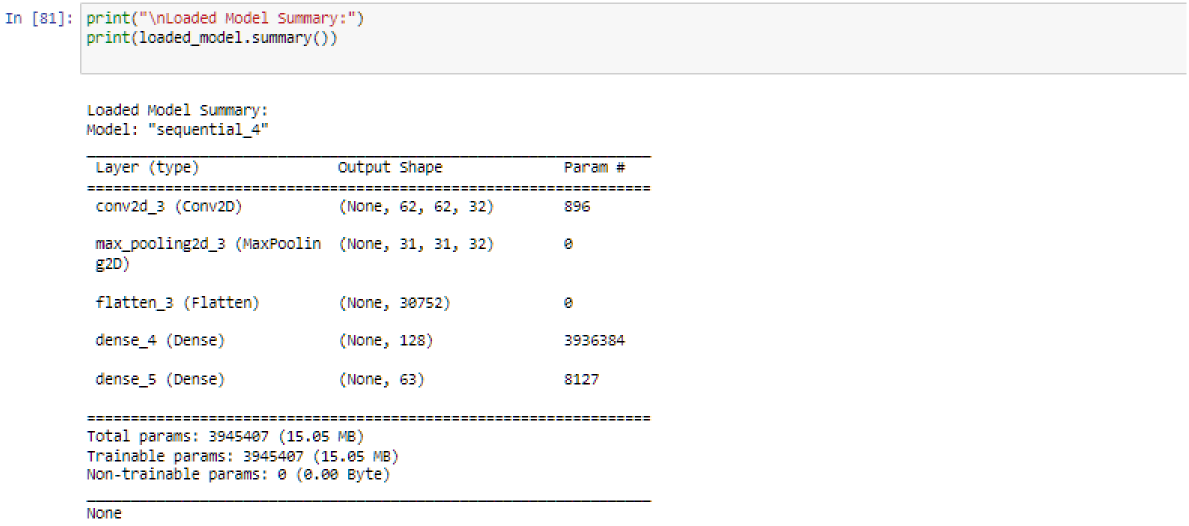


```
         Epoch 11/20
         35/35 [==============================] - 5s 133ms/step - loss: 0.4979 - accuracy: 0.8764 - val_loss: 0.2353 - val_accuracy: 0.9
         375
         Epoch 12/20
         35/35 [==============================] - 5s 140ms/step - loss: 0.3785 - accuracy: 0.9036 - val_loss: 0.2055 - val_accuracy: 0.9
         563
         Epoch 13/20
         35/35 [==============================] - 5s 133ms/step - loss: 0.3673 - accuracy: 0.9109 - val_loss: 0.1786 - val_accuracy: 0.9
         625
         Epoch 14/20
         35/35 [==============================] - 5s 132ms/step - loss: 0.2678 - accuracy: 0.9200 - val_loss: 0.0899 - val_accuracy: 0.9
         750
         Epoch 15/20
         35/35 [==============================] - 5s 129ms/step - loss: 0.2846 - accuracy: 0.9473 - val_loss: 0.1122 - val_accuracy: 0.9
         750
         Epoch 16/20
         35/35 [==============================] - 5s 129ms/step - loss: 0.2249 - accuracy: 0.9436 - val_loss: 0.1356 - val_accuracy: 0.9
         563
         Epoch 17/20
         35/35 [==============================] - 4s 126ms/step - loss: 0.2980 - accuracy: 0.9309 - val_loss: 0.1918 - val_accuracy: 0.9
         375
         Epoch 18/20
         35/35 [==============================] - 5s 129ms/step - loss: 0.1640 - accuracy: 0.9582 - val_loss: 0.0933 - val_accuracy: 0.9
         750
         Epoch 19/20
         35/35 [==============================] - 5s 154ms/step - loss: 0.1352 - accuracy: 0.9727 - val_loss: 0.0609 - val_accuracy: 0.9
         875
         Epoch 20/20
         35/35 [==============================] - 5s 136ms/step - loss: 0.1183 - accuracy: 0.9691 - val_loss: 0.0457 - val_accuracy: 0.9
         875

Out[77]: <keras.src.callbacks.History at 0x23aebd9fc90>
```

# 7.Save the model

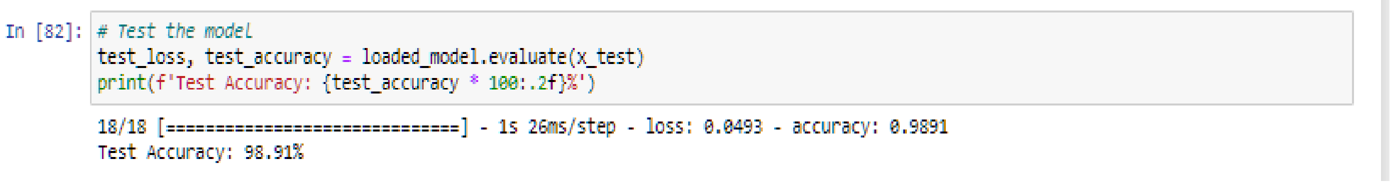


```
In [78]: # Import load_model from keras.models
         from tensorflow.keras.models import load_model
         import numpy as np
         model_save_path = 'logo_detection_model.h5'

In [79]: # Save the model
         model.save(model_save_path)

         C:\Users\WELCOME\anaconda3\Lib\site-packages\keras\src\engine\training.py:3103: UserWarning: You are saving your model as an HD
         F5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `mo
         del.save('my_model.keras')`.
           saving_api.save_model(

In [80]: # Load the saved model
         loaded_model = load_model(model_save_path)
```



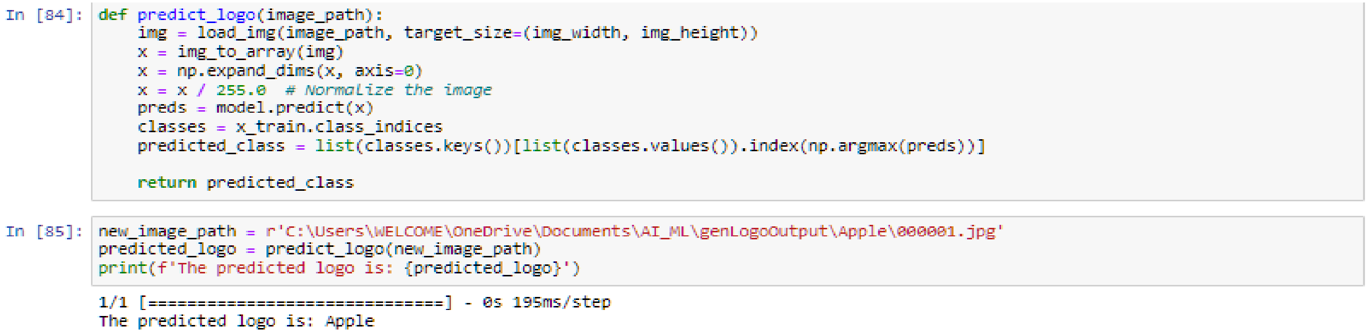
```
In [81]: print("\nLoaded Model Summary:")
         print(loaded_model.summary())


         Loaded Model Summary:
         Model: "sequential_4"
         _________________________________________________________________
          Layer (type)                Output Shape              Param #
         =================================================================
          conv2d_3 (Conv2D)           (None, 62, 62, 32)        896

          max_pooling2d_3 (MaxPoolin  (None, 31, 31, 32)        0
          g2D)

          flatten_3 (Flatten)         (None, 30752)             0

          dense_4 (Dense)             (None, 128)               3936384

          dense_5 (Dense)             (None, 63)                8127

         =================================================================
         Total params: 3945407 (15.05 MB)
         Trainable params: 3945407 (15.05 MB)
         Non-trainable params: 0 (0.00 Byte)
         _________________________________________________________________
         None
```

# 8.Test the model



```
In [82]: # Test the model
         test_loss, test_accuracy = loaded_model.evaluate(x_test)
         print(f'Test Accuracy: {test_accuracy * 100:.2f}%')

         18/18 [==============================] - 1s 26ms/step - loss: 0.0493 - accuracy: 0.9891
         Test Accuracy: 98.91%
```

# 9.Predict the model



```
In [84]: def predict_logo(image_path):
             img = load_img(image_path, target_size=(img_width, img_height))
             x = img_to_array(img)
             x = np.expand_dims(x, axis=0)
             x = x / 255.0  # Normalize the image
             preds = model.predict(x)
             classes = x_train.class_indices
             predicted_class = list(classes.keys())[list(classes.values()).index(np.argmax(preds))]

             return predicted_class

In [85]: new_image_path = r'C:\Users\WELCOME\OneDrive\Documents\AI_ML\genLogoOutput\Apple\000001.jpg'
         predicted_logo = predict_logo(new_image_path)
         print(f'The predicted logo is: {predicted_logo}')

         1/1 [==============================] - 0s 195ms/step
         The predicted logo is: Apple
```