```
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=Tru
```

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

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 128 using train_datagen generator
train_generator = train_datagen.flow_from_directory(  # train_datagen
        '/content/drive/MyDrive/NEU_DET/Train_IMAGES',  # This is the source directory for training images
        target_size=(200,200),  # All images will be resized to 200 x 200
        color_mode='rgb',
        batch_size = 64,
        # Specify the classes explicitly
        classes = ['crazing','inclusion','patches','pitted_surface','rolled_in_scale','scratches'],
        # Since we use categorical_crossentropy loss, we need categorical labels
        class_mode='categorical')

# All images will be rescaled by 1./255
test_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 128 using train_datagen generator
test_generator = test_datagen.flow_from_directory(
        '/content/drive/MyDrive/NEU_DET/Test_IMAGES',  # This is the source directory for training images
        target_size=(200,200),  # All images will be resized to 200 x 200
        color_mode='rgb', # grayscale, rgb
        batch_size = 64,  # 128, 64, 32, 24
        # Specify the classes explicitly
        classes = ['crazing','inclusion','patches','pitted_surface','rolled_in_scale','scratches'],
        # Since we use categorical_crossentropy loss, we need categorical labels
        class_mode='categorical')
```

```
Found 1464 images belonging to 6 classes.
Found 336 images belonging to 6 classes.
```

```
import tensorflow as tf

model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 200x 200 with 3 bytes color
    # The first convolution
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(200, 200, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # The second convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The third convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fourth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fifth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a dense layer
    tf.keras.layers.Flatten(),
    # 128 neuron in the fully-connected layer
    tf.keras.layers.Dense(1024, activation='relu'),  # 1024, 128, 64, 32
    # 6 output neurons for 6 classes with the softmax activation
    tf.keras.layers.Dense(6, activation='sigmoid')  # softmax
])

model.summary()
```

```
    Model: "sequential_2"
    _________________________________________________________________
    Layer (type)                 Output Shape              Param #   
    =================================================================
    conv2d_5 (Conv2D)            (None, 198, 198, 16)      160       
    _________________________________________________________________
    max_pooling2d_5 (MaxPooling2 (None, 99, 99, 16)        0         
    _________________________________________________________________
    conv2d_6 (Conv2D)            (None, 97, 97, 32)        4640      
    _________________________________________________________________
    max_pooling2d_6 (MaxPooling2 (None, 48, 48, 32)        0         
    _________________________________________________________________
    conv2d_7 (Conv2D)            (None, 46, 46, 64)        18496     
```

```
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 23, 23, 64)        0

_________________________________________________________________
conv2d_8 (Conv2D)            (None, 21, 21, 64)        36928

_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 10, 10, 64)        0

_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 64)          36928

_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 4, 4, 64)          0

_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0

_________________________________________________________________
dense_5 (Dense)              (None, 1024)              1049600

_________________________________________________________________
dense_6 (Dense)              (None, 6)                 6150
=================================================================
Total params: 1,152,902
Trainable params: 1,152,902
Non-trainable params: 0

_________________________________________________________________
```

```
# Image Detection Using the VGG-19 Convolutional Neural Network

# Build VGG19 structure
from tensorflow.keras.applications import VGG19

base_model = VGG19(weights='imagenet',
                   include_top=False,
                   input_shape=(200,200,3))

print('VGG19 Loaded')
print(base_model.summary())
```

```
VGG19 Loaded
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 200, 200, 3)]     0

_________________________________________________________________
block1_conv1 (Conv2D)        (None, 200, 200, 64)      1792

_________________________________________________________________
block1_conv2 (Conv2D)        (None, 200, 200, 64)      36928
```

```
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 100, 100, 64)      0

_________________________________________________________________
block2_conv1 (Conv2D)        (None, 100, 100, 128)     73856

_________________________________________________________________
block2_conv2 (Conv2D)        (None, 100, 100, 128)     147584

_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 50, 50, 128)       0

_________________________________________________________________
block3_conv1 (Conv2D)        (None, 50, 50, 256)       295168

_________________________________________________________________
block3_conv2 (Conv2D)        (None, 50, 50, 256)       590080

_________________________________________________________________
block3_conv3 (Conv2D)        (None, 50, 50, 256)       590080

_________________________________________________________________
block3_conv4 (Conv2D)        (None, 50, 50, 256)       590080

_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 25, 25, 256)       0

_________________________________________________________________
block4_conv1 (Conv2D)        (None, 25, 25, 512)       1180160

_________________________________________________________________
block4_conv2 (Conv2D)        (None, 25, 25, 512)       2359808

_________________________________________________________________
block4_conv3 (Conv2D)        (None, 25, 25, 512)       2359808

_________________________________________________________________
block4_conv4 (Conv2D)        (None, 25, 25, 512)       2359808

_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 12, 12, 512)       0

_________________________________________________________________
block5_conv1 (Conv2D)        (None, 12, 12, 512)       2359808

_________________________________________________________________
block5_conv2 (Conv2D)        (None, 12, 12, 512)       2359808

_________________________________________________________________
block5_conv3 (Conv2D)        (None, 12, 12, 512)       2359808

_________________________________________________________________
block5_conv4 (Conv2D)        (None, 12, 12, 512)       2359808

_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 6, 6, 512)         0
=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0

_________________________________________________________________
None
```

```
# Image Detection Using the VGG-19 Convolutional Neural Network
# Image Detection using the VGG 19 Convolutional Neural Network
import tensorflow as tf

model = tf.keras.models.Sequential(base_model)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))  # 4096
model.add(tf.keras.layers.Dense(128, activation='relu'))  # 4096, 2048, 1024, 512
model.add(tf.keras.layers.Dense(6, activation='sigmoid'))  # softmax, sigmoid

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 6, 6, 512)         20024384
_________________________________________________________________
flatten (Flatten)            (None, 18432)             0
_________________________________________________________________
dense (Dense)                (None, 128)               2359424
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 774
=================================================================
Total params: 22,401,094
Trainable params: 22,401,094
Non-trainable params: 0
_________________________________________________________________
```

```
#compile model using accuracy to measure model performance
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history = model.fit(
        train_generator,
        validation_data = test_generator,
        epochs = 30)   #50
```

```
Epoch 1/30
23/23 [==============================] - 49s 2s/step - loss: 2.1309 - accuracy: 0.1578 - val_loss: 1.7615 - val_accuracy: 0
Epoch 2/30
23/23 [==============================] - 23s 1s/step - loss: 1.6923 - accuracy: 0.2445 - val_loss: 1.4523 - val_accuracy: 0
```

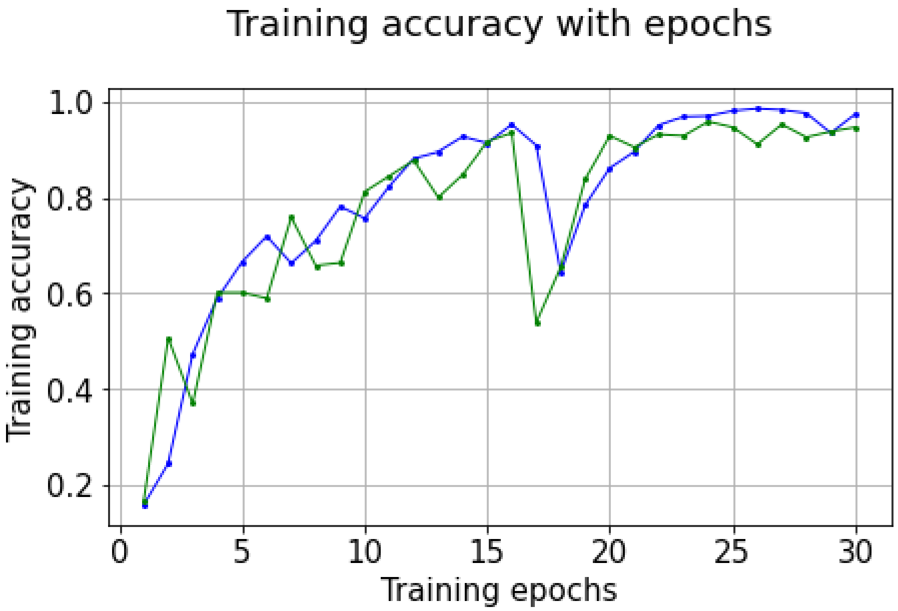
```
23/23 [==============================] - 23s 1s/step - loss: 1.6923 - accuracy: 0.2443 - val_loss: 1.4323 - val_accuracy: 0
Epoch 3/30
23/23 [==============================] - 23s 1s/step - loss: 1.2542 - accuracy: 0.4727 - val_loss: 1.1918 - val_accuracy: 0
Epoch 4/30
23/23 [==============================] - 24s 1s/step - loss: 0.9422 - accuracy: 0.5902 - val_loss: 0.9092 - val_accuracy: 0
Epoch 5/30
23/23 [==============================] - 24s 1s/step - loss: 0.7927 - accuracy: 0.6660 - val_loss: 0.9854 - val_accuracy: 0
Epoch 6/30
23/23 [==============================] - 23s 1s/step - loss: 0.6576 - accuracy: 0.7193 - val_loss: 0.9015 - val_accuracy: 0
Epoch 7/30
23/23 [==============================] - 24s 1s/step - loss: 0.8092 - accuracy: 0.6626 - val_loss: 0.6737 - val_accuracy: 0
Epoch 8/30
23/23 [==============================] - 23s 1s/step - loss: 0.6541 - accuracy: 0.7097 - val_loss: 0.8050 - val_accuracy: 0
Epoch 9/30
23/23 [==============================] - 24s 1s/step - loss: 0.4754 - accuracy: 0.7814 - val_loss: 1.1737 - val_accuracy: 0
Epoch 10/30
23/23 [==============================] - 23s 1s/step - loss: 0.5743 - accuracy: 0.7568 - val_loss: 0.8689 - val_accuracy: 0
Epoch 11/30
23/23 [==============================] - 23s 1s/step - loss: 0.3998 - accuracy: 0.8238 - val_loss: 0.4751 - val_accuracy: 0
Epoch 12/30
23/23 [==============================] - 23s 1s/step - loss: 0.3052 - accuracy: 0.8818 - val_loss: 0.3136 - val_accuracy: 0
Epoch 13/30
23/23 [==============================] - 23s 1s/step - loss: 0.2781 - accuracy: 0.8955 - val_loss: 0.8807 - val_accuracy: 0
Epoch 14/30
23/23 [==============================] - 23s 1s/step - loss: 0.2340 - accuracy: 0.9269 - val_loss: 0.6109 - val_accuracy: 0
Epoch 15/30
23/23 [==============================] - 23s 1s/step - loss: 0.2480 - accuracy: 0.9133 - val_loss: 0.3665 - val_accuracy: 0
Epoch 16/30
23/23 [==============================] - 23s 1s/step - loss: 0.1480 - accuracy: 0.9522 - val_loss: 0.2748 - val_accuracy: 0
Epoch 17/30
23/23 [==============================] - 23s 1s/step - loss: 0.2746 - accuracy: 0.9085 - val_loss: 2.2972 - val_accuracy: 0
Epoch 18/30
23/23 [==============================] - 23s 1s/step - loss: 1.3448 - accuracy: 0.6441 - val_loss: 0.7872 - val_accuracy: 0
Epoch 19/30
23/23 [==============================] - 23s 1s/step - loss: 0.5159 - accuracy: 0.7848 - val_loss: 0.4659 - val_accuracy: 0
Epoch 20/30
23/23 [==============================] - 23s 1s/step - loss: 0.3117 - accuracy: 0.8620 - val_loss: 0.3381 - val_accuracy: 0
Epoch 21/30
23/23 [==============================] - 23s 1s/step - loss: 0.2789 - accuracy: 0.8962 - val_loss: 0.6158 - val_accuracy: 0
Epoch 22/30
23/23 [==============================] - 23s 1s/step - loss: 0.1485 - accuracy: 0.9508 - val_loss: 0.5221 - val_accuracy: 0
Epoch 23/30
23/23 [==============================] - 23s 1s/step - loss: 0.1102 - accuracy: 0.9686 - val_loss: 0.6738 - val_accuracy: 0
Epoch 24/30
23/23 [==============================] - 23s 1s/step - loss: 0.0927 - accuracy: 0.9699 - val_loss: 0.2562 - val_accuracy: 0
Epoch 25/30

23/23 [==============================] - 23s 1s/step - loss: 0.0693 - accuracy: 0.9809 - val_loss: 0.3652 - val_accuracy: 0
```

```
23/23 [==============================] - 23s 1s/step - loss: 0.0693 - accuracy: 0.9809 - val_loss: 0.3632 - val_accuracy: 0
Epoch 26/30
23/23 [==============================] - 23s 1s/step - loss: 0.0517 - accuracy: 0.9857 - val_loss: 0.5310 - val_accuracy: 0
Epoch 27/30
23/23 [==============================] - 23s 1s/step - loss: 0.0541 - accuracy: 0.9829 - val_loss: 0.2720 - val_accuracy: 0
Epoch 28/30
23/23 [==============================] - 23s 1s/step - loss: 0.0976 - accuracy: 0.9761 - val_loss: 0.3028 - val_accuracy: 0
Epoch 29/30
23/23 [==============================] - 23s 1s/step - loss: 0.2315 - accuracy: 0.9344 - val_loss: 0.3036 - val_accuracy: 0
Epoch 30/30
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(7,4))
plt.plot([i+1 for i in range(30)],history.history['accuracy'],'-o',c='b',lw=1,markersize=2)
plt.plot([i+1 for i in range(30)],history.history['val_accuracy'],'-o',c='g',lw=1,markersize=2)
plt.grid(True)
plt.title("Training accuracy with epochs\n",fontsize=18)
plt.xlabel("Training epochs",fontsize=15)
plt.ylabel("Training accuracy",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```


Training accuracy with epochs
Training accuracy
Training epochs