1. Import The ImageDataGenerator Library

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

2. Configure ImageDataGenerator Class

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.1,
                                   zoom_range = 0.1,
                                   horizontal_flip = True)

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

3. Apply ImageDataGenerator Functionality To Trainset And Testset

```
training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/Intell
                                                 target_size = (224, 224),
                                                 batch_size = 10,
                                                 class_mode = 'categorical')
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/Intelligent
                                            target_size = (224, 224),
                                            batch_size = 10,
                                            class_mode = 'categorical')
```

```
Found 979 images belonging to 3 classes.
Found 171 images belonging to 3 classes.
```

1. Importing The Model Building Libraries

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
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
```

2. Loading The Model

```
IMAGE_SIZE = [224, 224]

train_path = '/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For
valid_path = '/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For
```

```
vgg16 = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

3. Adding Flatten Layer

```
for layer in vgg16.layers:
    layer.trainable = False
```

```
folders = glob('/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/body/training/*
```

```
folders
```

```
['/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/body/training/02-side',
 '/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/body/training/00-front',
 '/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/body/training/01-rear']
```

```
x = Flatten()(vgg16.output)
```

```
len(folders)
```

```
3
```

4. Adding Output Layer

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

5. Creating A Model Object

```
model = Model(inputs=vgg16.input, outputs=prediction)
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (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_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_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_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 3)                 75267

=================================================================
Total params: 14,789,955
Trainable params: 75,267
Non-trainable params: 14,714,688
_________________________________________________________________
```

6. Configure The Learning Process

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

7. Train The Model

```
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=25,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

```
Epoch 1/25
98/98 [==============================] - 606s 6s/step - loss: 1.2827 - accuracy: 0.5649 - val_loss: 0.8292 - val_accuracy: 0.7076
Epoch 2/25
98/98 [==============================] - 601s 6s/step - loss: 0.6301 - accuracy: 0.7467 - val_loss: 1.2482 - val_accuracy: 0.5965
Epoch 3/25
98/98 [==============================] - 601s 6s/step - loss: 0.5073 - accuracy: 0.8039 - val_loss: 0.8174 - val_accuracy: 0.7193
Epoch 4/25
98/98 [==============================] - 601s 6s/step - loss: 0.3564 - accuracy: 0.8621 - val_loss: 0.9245 - val_accuracy: 0.6608
Epoch 5/25
98/98 [==============================] - 599s 6s/step - loss: 0.2951 - accuracy: 0.8917 - val_loss: 1.9934 - val_accuracy: 0.5906
Epoch 6/25
98/98 [==============================] - 638s 7s/step - loss: 0.2557 - accuracy: 0.9152 - val_loss: 0.9176 - val_accuracy: 0.6842
Epoch 7/25
98/98 [==============================] - 607s 6s/step - loss: 0.2083 - accuracy: 0.9367 - val_loss: 0.9594 - val_accuracy: 0.7018
Epoch 8/25
98/98 [==============================] - 600s 6s/step - loss: 0.2184 - accuracy: 0.9122 - val_loss: 1.0329 - val_accuracy: 0.6784
Epoch 9/25
98/98 [==============================] - 602s 6s/step - loss: 0.1320 - accuracy: 0.9581 - val_loss: 1.0539 - val_accuracy: 0.7135
Epoch 10/25
98/98 [==============================] - 599s 6s/step - loss: 0.1131 - accuracy: 0.9622 - val_loss: 1.2113 - val_accuracy: 0.6842
Epoch 11/25
98/98 [==============================] - 597s 6s/step - loss: 0.1001 - accuracy: 0.9745 - val_loss: 0.9917 - val_accuracy: 0.7018
Epoch 12/25
98/98 [==============================] - 598s 6s/step - loss: 0.0954 - accuracy: 0.9745 - val_loss: 1.0601 - val_accuracy: 0.7018
Epoch 13/25
98/98 [==============================] - 594s 6s/step - loss: 0.0695 - accuracy: 0.9816 - val_loss: 1.3700 - val_accuracy: 0.6433
Epoch 14/25
98/98 [==============================] - 599s 6s/step - loss: 0.1414 - accuracy: 0.9653 - val_loss: 1.1607 - val_accuracy: 0.6667
Epoch 15/25
98/98 [==============================] - 600s 6s/step - loss: 0.0905 - accuracy: 0.9796 - val_loss: 1.4014 - val_accuracy: 0.6667
Epoch 16/25
98/98 [==============================] - 601s 6s/step - loss: 0.0797 - accuracy: 0.9775 - val_loss: 1.6741 - val_accuracy: 0.6491
Epoch 17/25
98/98 [==============================] - 602s 6s/step - loss: 0.1042 - accuracy: 0.9745 - val_loss: 1.2824 - val_accuracy: 0.6959
Epoch 18/25
98/98 [==============================] - 600s 6s/step - loss: 0.0831 - accuracy: 0.9785 - val_loss: 1.1667 - val_accuracy: 0.6901
Epoch 19/25
98/98 [==============================] - 603s 6s/step - loss: 0.0826 - accuracy: 0.9704 - val_loss: 1.3747 - val_accuracy: 0.6374
Epoch 20/25
98/98 [==============================] - 600s 6s/step - loss: 0.0536 - accuracy: 0.9837 - val_loss: 1.2074 - val_accuracy: 0.6550
Epoch 21/25
98/98 [==============================] - 597s 6s/step - loss: 0.0716 - accuracy: 0.9796 - val_loss: 1.5491 - val_accuracy: 0.6725
Epoch 22/25
98/98 [==============================] - 599s 6s/step - loss: 0.0457 - accuracy: 0.9918 - val_loss: 1.2930 - val_accuracy: 0.7135
Epoch 23/25
98/98 [==============================] - 601s 6s/step - loss: 0.0526 - accuracy: 0.9928 - val_loss: 1.2576 - val_accuracy: 0.6959
Epoch 24/25
98/98 [==============================] - 601s 6s/step - loss: 0.0421 - accuracy: 0.9908 - val_loss: 1.3347 - val_accuracy: 0.7193
Epoch 25/25
98/98 [==============================] - 597s 6s/step - loss: 0.0597 - accuracy: 0.9826 - val_loss: 1.4728 - val_accuracy: 0.6725
```

8. Save The Model

```
from tensorflow.keras.models import load_model

model.save('/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Model/body.h5')
```

9. Test The Model

```
from tensorflow.keras.models import load_model
import cv2
from skimage.transform import resize
```

```
model = load_model('/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Model/body.h5')
```

```
def detect(frame):
  img = cv2.resize(frame,(224,224))
  img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

  if(np.max(img)>1):
    img = img/255.0
  img = np.array([img])
  prediction = model.predict(img)
  label = ["front","rear","side"]
  preds = label[np.argmax(prediction)]
  return preds
```

```
import numpy as np
```

```
data = "/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/body/training/00-front/0005.JPEG
image = cv2.imread(data)
print(detect(image))
```

```
1/1 [==============================] - 1s 638ms/step
front
```

1. Import The ImageDataGenerator Library

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

IMAGE PRE PROCESSING

2. Configure ImageDataGenerator Class

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.1,
                                   zoom_range = 0.1,
                                   horizontal_flip = True)

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

3. Apply ImageDataGenerator Functionality To Trainset And Testset

```
training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companie
                                                 target_size = (224, 224),
                                                 batch_size = 10,
                                                 class_mode = 'categorical')
test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dat
                                            target_size = (224, 224),
                                            batch_size = 10,
                                            class_mode = 'categorical')
```

```
Found 979 images belonging to 3 classes.
Found 171 images belonging to 3 classes.
```

MODEL BUILDING

1. Importing The Model Building Libraries

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
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
```

2. Loading The Model

```
IMAGE_SIZE = [224, 224]

train_path = '/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/level/training'
valid_path = '/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/level/validation'
```

```
vgg16 = 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 [==============================] - 0s 0us/step
```

3. Adding Flatten Layer

```
for layer in vgg16.layers:
    layer.trainable = False
```

```
folders = glob('/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/level/training/*')
```

```
folders
```

```
['/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/level/training/02-moderate',
 '/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/level/training/03-severe',
 '/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/level/training/01-minor']
```

```
x = Flatten()(vgg16.output)
```

```
len(folders)
```

```
3
```

4. Adding Output Layer

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

5. Creating A Model Object

```
model = Model(inputs=vgg16.input, outputs=prediction)
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (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_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_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_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0
```

6. Configure The Learning Process

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

7. Train The Model

```
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=25,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

```
Epoch 1/25
98/98 [==============================] - 615s 6s/step - loss: 1.2465 - accuracy: 0.5516 - val_loss: 1.0659 - val_accuracy: 0.5731
Epoch 2/25
98/98 [==============================] - 604s 6s/step - loss: 0.6654 - accuracy: 0.7549 - val_loss: 1.0368 - val_accuracy: 0.6316
Epoch 3/25
98/98 [==============================] - 604s 6s/step - loss: 0.5950 - accuracy: 0.7630 - val_loss: 1.1309 - val_accuracy: 0.6257
Epoch 4/25
98/98 [==============================] - 601s 6s/step - loss: 0.4964 - accuracy: 0.8069 - val_loss: 1.1262 - val_accuracy: 0.6082
Epoch 5/25
98/98 [==============================] - 603s 6s/step - loss: 0.3559 - accuracy: 0.8672 - val_loss: 1.1408 - val_accuracy: 0.6316
Epoch 6/25
98/98 [==============================] - 604s 6s/step - loss: 0.2425 - accuracy: 0.9152 - val_loss: 1.1566 - val_accuracy: 0.5789
Epoch 7/25
98/98 [==============================] - 604s 6s/step - loss: 0.1964 - accuracy: 0.9367 - val_loss: 1.1200 - val_accuracy: 0.6199
Epoch 8/25
98/98 [==============================] - 598s 6s/step - loss: 0.2119 - accuracy: 0.9203 - val_loss: 1.1181 - val_accuracy: 0.6316
Epoch 9/25
98/98 [==============================] - 597s 6s/step - loss: 0.1111 - accuracy: 0.9622 - val_loss: 1.3554 - val_accuracy: 0.5614
Epoch 10/25
98/98 [==============================] - 595s 6s/step - loss: 0.1394 - accuracy: 0.9438 - val_loss: 1.2256 - val_accuracy: 0.6082
Epoch 11/25
98/98 [==============================] - 598s 6s/step - loss: 0.1167 - accuracy: 0.9602 - val_loss: 1.3020 - val_accuracy: 0.6374
Epoch 12/25
98/98 [==============================] - 598s 6s/step - loss: 0.0823 - accuracy: 0.9755 - val_loss: 1.3000 - val_accuracy: 0.6550
Epoch 13/25
98/98 [==============================] - 602s 6s/step - loss: 0.1062 - accuracy: 0.9632 - val_loss: 1.2962 - val_accuracy: 0.6433
Epoch 14/25
98/98 [==============================] - 599s 6s/step - loss: 0.0717 - accuracy: 0.9775 - val_loss: 1.3089 - val_accuracy: 0.6491
Epoch 15/25
98/98 [==============================] - 598s 6s/step - loss: 0.0692 - accuracy: 0.9826 - val_loss: 1.2885 - val_accuracy: 0.6023
Epoch 16/25
98/98 [==============================] - 595s 6s/step - loss: 0.0449 - accuracy: 0.9898 - val_loss: 1.7932 - val_accuracy: 0.5673
Epoch 17/25
98/98 [==============================] - 609s 6s/step - loss: 0.0522 - accuracy: 0.9867 - val_loss: 1.2697 - val_accuracy: 0.6433
Epoch 18/25
98/98 [==============================] - 607s 6s/step - loss: 0.0386 - accuracy: 0.9969 - val_loss: 1.5100 - val_accuracy: 0.6023
Epoch 19/25
98/98 [==============================] - 595s 6s/step - loss: 0.0381 - accuracy: 0.9939 - val_loss: 1.2199 - val_accuracy: 0.6784
Epoch 20/25
98/98 [==============================] - 596s 6s/step - loss: 0.0196 - accuracy: 1.0000 - val_loss: 1.2907 - val_accuracy: 0.6433
Epoch 21/25
98/98 [==============================] - 597s 6s/step - loss: 0.0394 - accuracy: 0.9928 - val_loss: 1.2678 - val_accuracy: 0.6491
Epoch 22/25
98/98 [==============================] - 595s 6s/step - loss: 0.0377 - accuracy: 0.9908 - val_loss: 1.4709 - val_accuracy: 0.6316
Epoch 23/25
98/98 [==============================] - 595s 6s/step - loss: 0.0387 - accuracy: 0.9918 - val_loss: 1.3320 - val_accuracy: 0.6257
Epoch 24/25
98/98 [==============================] - 596s 6s/step - loss: 0.0279 - accuracy: 0.9949 - val_loss: 1.6355 - val_accuracy: 0.6433
Epoch 25/25
98/98 [==============================] - 603s 6s/step - loss: 0.0271 - accuracy: 0.9939 - val_loss: 1.3182 - val_accuracy: 0.6608
```

8. Save The Model

```
from tensorflow.keras.models import load_model

model.save('/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Model/level.h5')
```

9. Test The Model

```
from tensorflow.keras.models import load_model
import cv2
from skimage.transform import resize
```

```
model = load_model('/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Model/level.h5')
```

```
def detect(frame):
  img = cv2.resize(frame,(224,224))
  img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

  if(np.max(img)>1):
    img = img/255.0
  img = np.array([img])
  prediction = model.predict(img)
  label = ["minor","moderate","severe"]
  preds = label[np.argmax(prediction)]
  return preds
```

```
import numpy as np
```

```
data = "/content/drive/MyDrive/Intelligent Vehicle Damage Assessment & Cost Estimator For Insurance Companies/Dataset/level/validation/01-minor/0010.J
image = cv2.imread(data)
print(detect(image))
```

```
1/1 [==============================] - 1s 728ms/step
minor
```