Artificial Intelligence (CO-304)
AI project

Facemask Detection

Submitted to: -

Mr. Himanshu Nandanwar

Submitted by:-
Aditya Shrinath Shaw(2K21/SE14)
Katyayani Sharma(2K21/SE/97)

Introduction: -

Face mask detection is a technology that uses computer vision techniques to detect whether a person in an image or video is wearing a face mask. It has gained significant attention and importance during the COVID-19 pandemic as a tool to enforce mask-wearing mandates and help prevent the spread of the virus.

Applications of face mask detection include:

- Public Safety: Monitoring compliance with mask-wearing mandates in public spaces such as airports, hospitals, schools, and workplaces.
- Healthcare: Ensuring that healthcare workers and patients are wearing masks properly in medical settings.
- Security: Integrating with surveillance systems to identify individuals not wearing masks in secure areas.
- Automated Systems: Triggering automated responses, such as access control or alerts, based on mask detection results.

Advantages of face mask detection include:

- Public Health: Helps in enforcing mask-wearing guidelines and reducing the spread of infectious diseases.
- Efficiency: Automates the process of monitoring mask-wearing compliance, reducing the need for manual intervention.
- Safety: Provides an additional layer of safety for individuals in environments where mask-wearing is required.
- Cost-Effectiveness: Can be integrated into existing surveillance systems, minimizing the need for additional infrastructure.

Overall, face mask detection has the potential to play a crucial role in maintaining public health and safety during pandemics and in other settings where mask-wearing is important.

Source code:

Cell 1:-

```
import cv2
import numpy as np
```

Cell 2:-

```
import tensorflow as tf
print(tf.__version__)
```

```
2.15.0
```

Cell 3:-

```
train_dir = "/content/drive/MyDrive/face mask detection/Train"
test_dir = "/content/drive/MyDrive/face mask detection/Test"
valid_dir = "/content/drive/MyDrive/face mask detection/Validation"
```

Cell 4:-

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
```

Cell 5:-

```
target_size=(96,96)
Batch_size = 24
```

Cell 6:-

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range = 40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=target_size,
    batch_size=batch_size,
    color_mode='rgb',
    shuffle=True,
    seed=42,
    class_mode='categorical')
```

```
Found 10009 images belonging to 2 classes.
```

Cell 7:-

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

valid_generator = valid_datagen.flow_from_directory(
    valid_dir,
    target_size=target_size,
    batch_size=batch_size,
    color_mode='rgb',
    shuffle=False,
    class_mode='categorical'
)
```

```
Found 800 images belonging to 2 classes.
```

Cell 8:-

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

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=target_size,
    batch_size=batch_size,
    color_mode='rgb',
    shuffle=False,
    class_mode='categorical'
)
```

```
Found 992 images belonging to 2 classes.
```

Cell 9:-

```
from tensorflow.keras import models, layers
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ModelCheckpoint
```

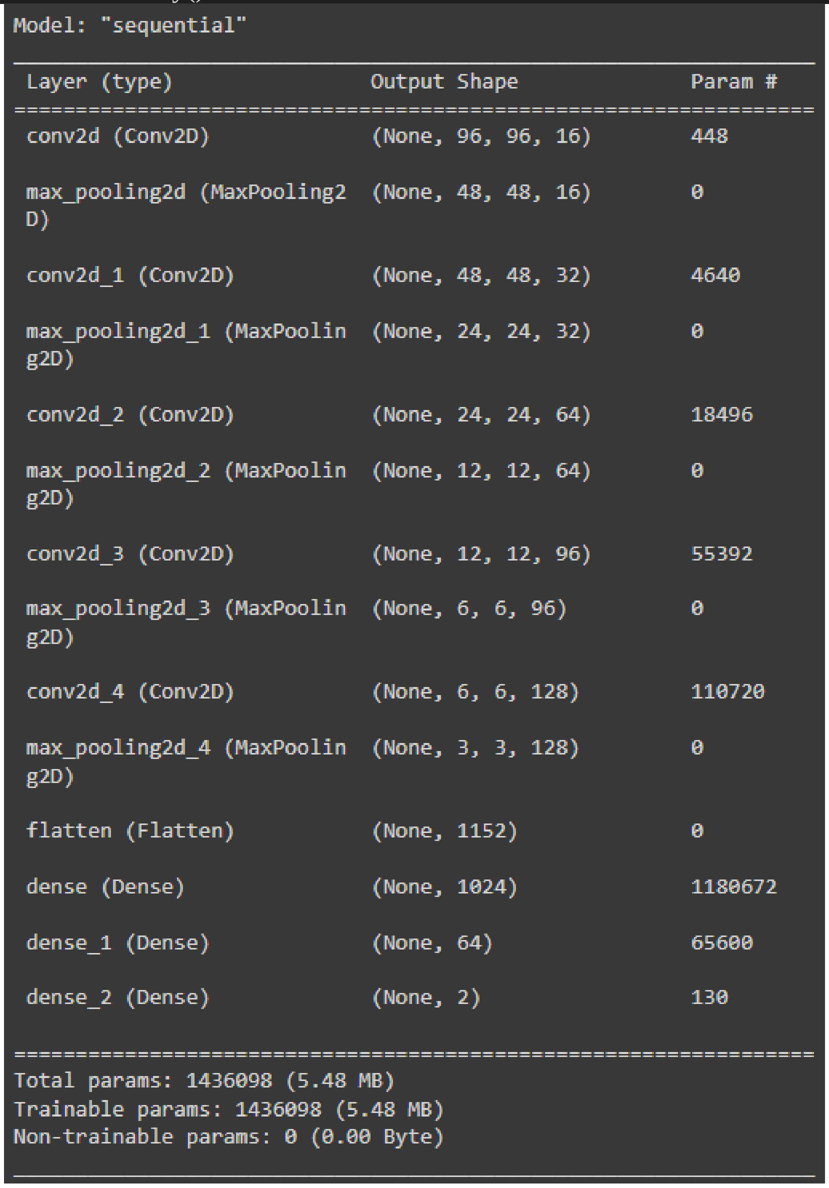
Cell 10:-

```
num_classes = 2 #With mask, Without mask
input_shape = (96,96,3)
```

Cell 11:-

```
model= models.Sequential()
#1st layer
model.add(layers.Conv2D(16,(3,3), activation='relu', padding='same', input_shape=input_shape))
model.add(layers.MaxPool2D((2,2)))
#2nd layer
model.add(layers.Conv2D(32,(3,3), activation='relu', padding='same'))
model.add(layers.MaxPool2D((2,2)))
#3rd layer
```

```
model.add(layers.Conv2D(64,(3,3), activation='relu', padding='same'))
model.add(layers.MaxPool2D((2,2)))
#4th layer
model.add(layers.Conv2D(96,(3,3), activation='relu', padding='same'))
model.add(layers.MaxPool2D((2,2)))
#5th layer
model.add(layers.Conv2D(128,(3,3), activation='relu', padding='same'))
model.add(layers.MaxPool2D((2,2)))

model.add(layers.Flatten())
model.add(layers.Dense(1024))

model.add(layers.Dense(64))

model.add(layers.Dense(num_classes, activation='softmax'))
model.summary()
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 96, 96, 16)        448

 max_pooling2d (MaxPooling2  (None, 48, 48, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 48, 48, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 24, 24, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 12, 12, 96)        55392

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 96)          0
 g2D)

 conv2d_4 (Conv2D)           (None, 6, 6, 128)         110720

 max_pooling2d_4 (MaxPoolin  (None, 3, 3, 128)         0
 g2D)

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 1024)              1180672

 dense_1 (Dense)             (None, 64)                65600

 dense_2 (Dense)             (None, 2)                 130

=================================================================
Total params: 1436098 (5.48 MB)
Trainable params: 1436098 (5.48 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Cell 12:-

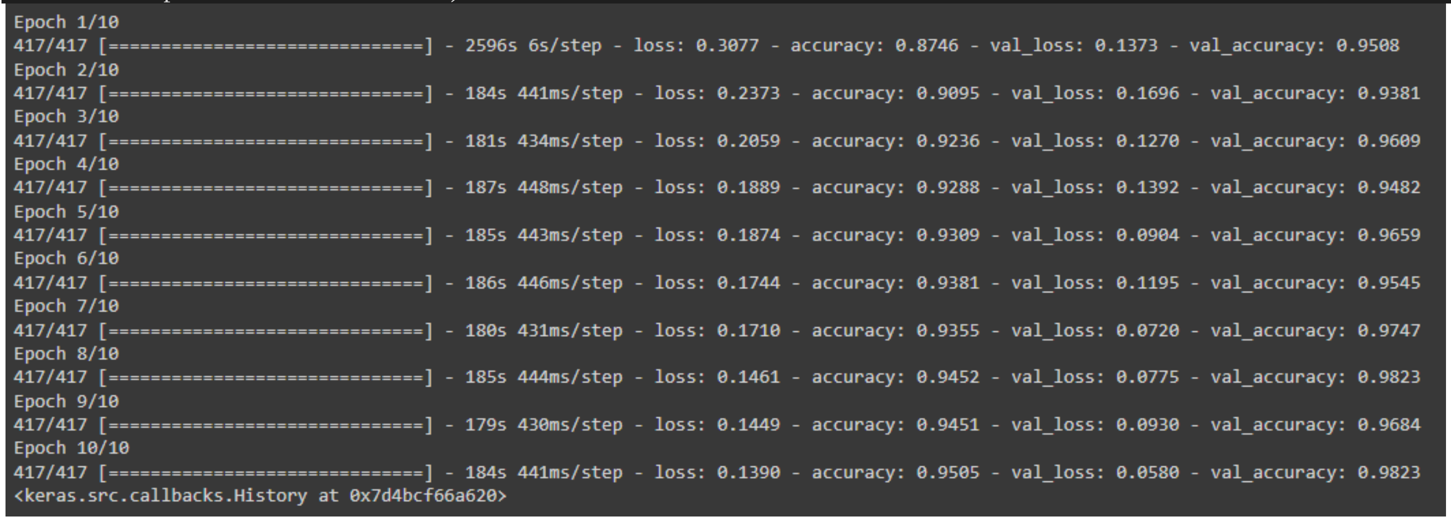
```
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

Cell 13:-

```
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size
num_epochs = 50
```

Cell 14:-

```
model.fit(train_generator, steps_per_epoch=STEP_SIZE_TRAIN, epochs=num_epochs, validation_data=valid_generator,
validation_steps= STEP_SIZE_VALID)
```



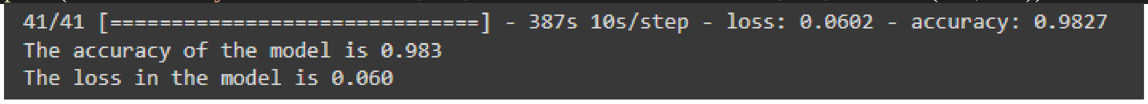
```
Epoch 1/10
417/417 [==============================] - 2596s 6s/step - loss: 0.3077 - accuracy: 0.8746 - val_loss: 0.1373 - val_accuracy: 0.9508
Epoch 2/10
417/417 [==============================] - 184s 441ms/step - loss: 0.2373 - accuracy: 0.9095 - val_loss: 0.1696 - val_accuracy: 0.9381
Epoch 3/10
417/417 [==============================] - 181s 434ms/step - loss: 0.2059 - accuracy: 0.9236 - val_loss: 0.1270 - val_accuracy: 0.9609
Epoch 4/10
417/417 [==============================] - 187s 448ms/step - loss: 0.1889 - accuracy: 0.9288 - val_loss: 0.1392 - val_accuracy: 0.9482
Epoch 5/10
417/417 [==============================] - 185s 443ms/step - loss: 0.1874 - accuracy: 0.9309 - val_loss: 0.0904 - val_accuracy: 0.9659
Epoch 6/10
417/417 [==============================] - 186s 446ms/step - loss: 0.1744 - accuracy: 0.9381 - val_loss: 0.1195 - val_accuracy: 0.9545
Epoch 7/10
417/417 [==============================] - 180s 431ms/step - loss: 0.1710 - accuracy: 0.9355 - val_loss: 0.0720 - val_accuracy: 0.9747
Epoch 8/10
417/417 [==============================] - 185s 444ms/step - loss: 0.1461 - accuracy: 0.9452 - val_loss: 0.0775 - val_accuracy: 0.9823
Epoch 9/10
417/417 [==============================] - 179s 430ms/step - loss: 0.1449 - accuracy: 0.9451 - val_loss: 0.0930 - val_accuracy: 0.9684
Epoch 10/10
417/417 [==============================] - 184s 441ms/step - loss: 0.1390 - accuracy: 0.9505 - val_loss: 0.0580 - val_accuracy: 0.9823
<keras.src.callbacks.History at 0x7d4bcf66a620>
```

Cell 15:-

```
model.save('facemask_cnn.keras')
```

Cell 16:-

```
loss, acc = model.evaluate(test_generator, steps=STEP_SIZE_TEST)
print("The accuracy of the model is {:.3f}\nThe loss in the model is {:.3f}".format(acc,loss))
```



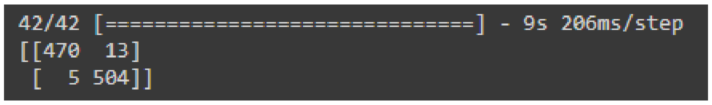
```
41/41 [==============================] - 387s 10s/step - loss: 0.0602 - accuracy: 0.9827
The accuracy of the model is 0.983
The loss in the model is 0.060
```

Cell 17:-

```
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
```

Cell 18:-

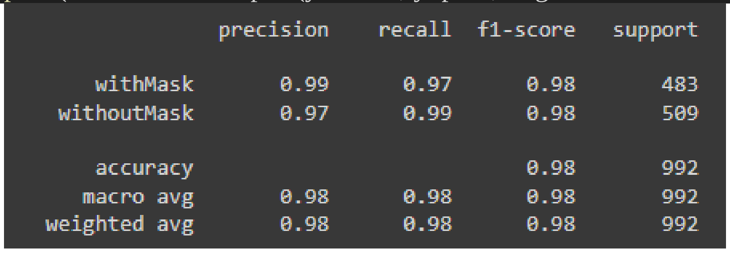
```
preds= model.predict(test_generator)
y_pred= np.argmax(preds, axis = 1)
y_actual = test_generator.classes
cm = confusion_matrix(y_actual, y_pred)
print(cm)
```



```
42/42 [==============================] - 9s 206ms/step
[[470  13]
 [  5 504]]
```

Cell 19:-

```
labels = ['withMask', 'withoutMask']
print(classification_report(y_actual, y_pred, target_names=labels))
```

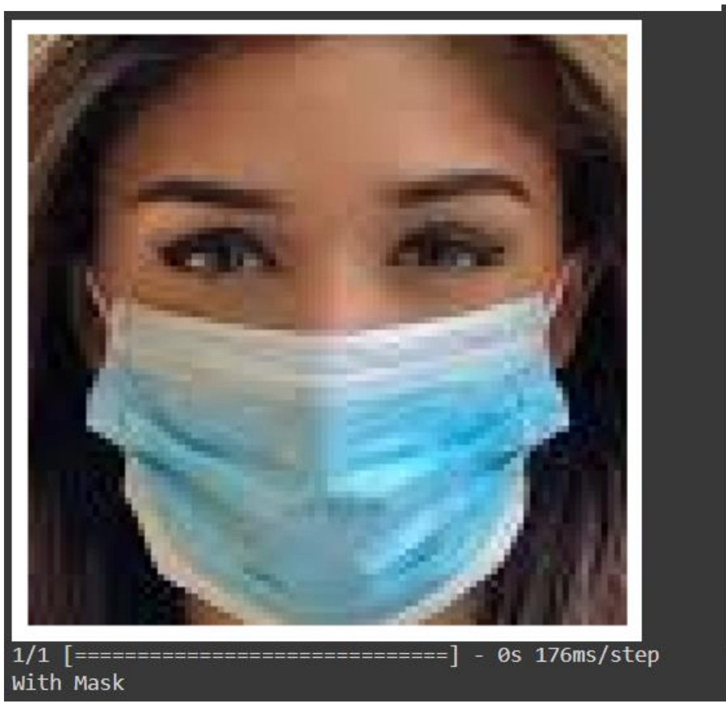


```
              precision    recall  f1-score   support

    withMask       0.99      0.97      0.98       483
 withoutMask       0.97      0.99      0.98       509

    accuracy                           0.98       992
   macro avg       0.98      0.98      0.98       992
weighted avg       0.98      0.98      0.98       992
```

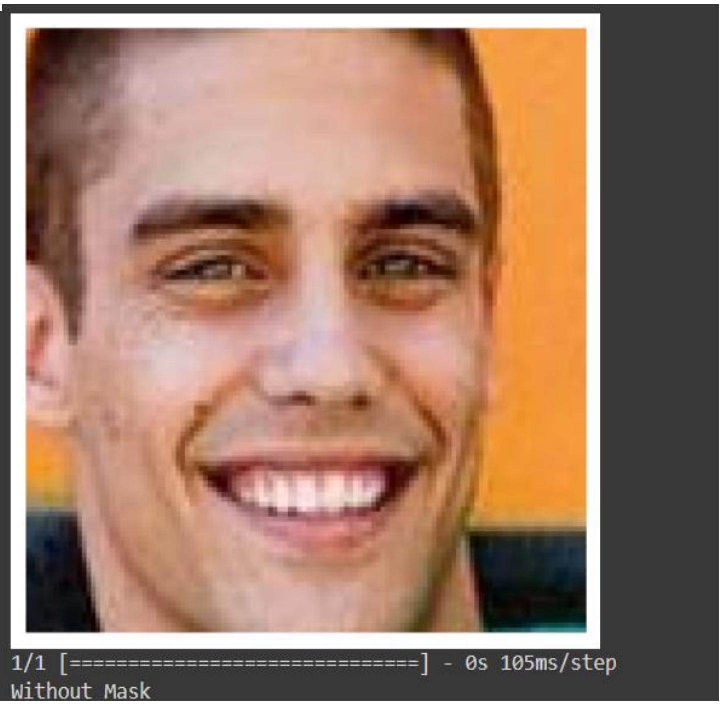
Interface to Access the model:-

```
import cv2
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
# Load the model
model = tf.keras.models.load_model('/content/drive/MyDrive/face mask
detection/fm_Part2_cnn.keras')
# Function to preprocess image
def preprocess_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (96, 96))
    img = img / 255.0
    img = np.expand_dims(img, axis=0)
    return img
# Load and preprocess the image
image_path = '/content/drive/MyDrive/face mask detection/mask2.png'
image = preprocess_image(image_path)
#display the image
img = mpimg.imread(image_path)
plt.imshow(img)
plt.axis('off')  # Turn off axis numbers
plt.show()

# Make predictions
predictions = model.predict(image)
# Print the prediction
if predictions[0][0]> 0.5:
    print("Without Mask")
else:
    print("With Mask")
```

Output: -



```
1/1 [==============================] - 0s 176ms/step
With Mask
```



```
1/1 [==============================] - 0s 105ms/step
Without Mask
```

Libraries used: -

NumPy:

NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy's arrays are more efficient than Python's built-in lists for numerical operations. It also offers tools for integrating C/C++ and Fortran code, and it can be used to perform a variety of computations such as linear algebra, statistical operations, and Fourier analysis. NumPy is a foundational library for many other Python libraries in the data science and machine learning ecosystem.

Tensorflow:

TensorFlow is an open-source machine learning framework developed by Google for building and training neural network models. It provides a comprehensive ecosystem of tools, libraries, and community resources for machine learning development. TensorFlow supports both deep learning and traditional machine learning models and can be used for a wide range of tasks, including classification, regression, clustering, and more.

One of the key features of TensorFlow is its flexibility and scalability. It allows users to define and train complex models using high-level APIs like Keras or to customize models and training loops at a lower level using TensorFlow's core API. TensorFlow also supports distributed computing, allowing models to be trained across multiple GPUs or TPUs for faster performance.

TensorFlow is widely used in both academia and industry for research and production applications, making it one of the most popular and powerful machine learning frameworks available today.

Keras:

Keras is a high-level neural networks API, written in Python, that simplifies the development and deployment of deep learning models. It can run on TensorFlow, CNTK, or Theano. Released in March 2015 by François Chollet, Keras enables fast experimentation with various deep neural networks.

Key features of Keras include its simplicity and ease of use, allowing users to build and train neural networks with just a few lines of code. It offers pre-built models like convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which can be easily customized. Keras also provides functions for data preprocessing, such as image augmentation and sequence padding, streamlining the preparation of training data.

Its modularity and flexibility enable the construction of neural networks as a series of interconnected layers, facilitating experimentation with different architectures. Keras includes built-in functions for regularization, such as dropout and L1/L2 weight regularization, which can prevent overfitting.

Keras offers tools for monitoring and visualizing the training process, helping users track metrics like accuracy and loss, and visualize training progress. It is portable, running on various hardware platforms, and supports both CPU and GPU acceleration. Additionally, Keras can be integrated with other machine learning and deep learning libraries, making it versatile for diverse tasks.

While Keras is user-friendly and suitable for many applications, it may not be ideal for advanced research requiring fine-grained control over architecture and training processes. Despite this limitation, Keras remains a powerful tool for building and deploying deep learning models efficiently.

Matplotlib:

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is widely used for generating plots, charts, histograms, and other graphical representations of data. Matplotlib's pyplot module provides a MATLAB-like interface for creating plots quickly and easily.

With Matplotlib, you can create a wide variety of plots, including line plots, scatter plots, bar plots, histograms, and more. It also supports 3D plotting and can be used to create complex, publication-quality figures.

One of the key features of Matplotlib is its customization options. You can easily customize the appearance of your plots by changing colors, line styles, markers, fonts, and more. Matplotlib also supports LaTeX formatting for mathematical expressions, making it ideal for scientific publications.

Matplotlib can be used interactively within Jupyter notebooks or as a standalone library in Python scripts. It is highly compatible with NumPy and pandas, making it easy to visualize data from these libraries.

Overall, Matplotlib is a powerful and flexible library for creating high-quality visualizations in Python.

OpenCV:

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. It provides tools for image and video analysis, including various algorithms for object detection, feature extraction, image processing, and more. OpenCV is written in C++ and has bindings for Python, Java, and MATLAB/Octave.

One of the key features of OpenCV is its wide range of supported platforms, including Windows, Linux, macOS, Android, and iOS, making it highly versatile for developing computer vision applications across different devices. It also offers support for multiple programming languages, allowing developers to choose the language that best suits their needs.

OpenCV is used in a variety of fields, including robotics, augmented reality, facial recognition, and medical image analysis. Its comprehensive set of functions and algorithms, combined with its ease of use and cross-platform compatibility, make it a popular choice for both academic research and industrial applications in the field of computer vision.

Scikit-Learn:

Scikit-learn is a popular machine learning library in Python that provides simple and efficient tools for data mining and data analysis. It is built on top of NumPy, SciPy, and matplotlib and is designed to interoperate with other Python libraries such as NumPy and pandas.

Scikit-learn provides a wide range of supervised and unsupervised learning algorithms through a consistent interface. It includes algorithms for classification, regression, clustering, dimensionality reduction, model selection, and preprocessing. Some of the most commonly used algorithms include support vector machines (SVM), random forests, k-nearest neighbors (KNN), and principal component analysis (PCA).

One of the key features of scikit-learn is its ease of use and its well-organized and easy-to-understand API. It provides sensible defaults for many parameters, making it easy for beginners to get started with machine learning. At the same time, it also allows for fine-tuning of parameters for more experienced users.

Scikit-learn also includes tools for model evaluation, including functions for cross-validation, hyperparameter tuning, and model selection. It provides built-in metrics for evaluating the performance of models, such as accuracy, precision, recall, and F1-score.

Overall, scikit-learn is a powerful and versatile library that is widely used in the machine learning community for both academic research and industrial applications. Its simplicity, efficiency, and comprehensive documentation make it a valuable tool for anyone looking to work with machine learning in Python.