openCV & MediaPipe Cheat Sheet

1. `Basic OpenCV Operations`

- `cv2.imread(filename, flag)` – Load an image (`flag`: 0=grayscale, 1=color, -1=unchanged)
- `cv2.imshow(winname, mat)` – Display an image (`winname`: window name, `mat`: image)
- `cv2.imwrite(filename, img)` – Save an image
- `cv2.waitKey(delay)` – Wait for a key event (`delay`: milliseconds)
- `cv2.destroyAllWindows()` – Close all OpenCV windows

—– 2. Video Processing - `cv2.VideoCapture(source)` – Capture video from camera/file (`source`: 0=webcam, filename=video file) - `cv2.VideoWriter(filename, fourcc, fps, frameSize)` – Save video (`fourcc`: codec, `fps`: frames per second)

—– 3. Color Space Conversion - `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` – Convert to grayscale - `cv2.cvtColor(image, cv2.COLOR_BGR2HSV)` – Convert to HSV - `cv2.cvtColor(image, cv2.COLOR_BGR2RGB)` – Convert to RGB

—– 4. Drawing Functions - `cv2.line(image, start, end, color, thickness)` – Draw a line - `cv2.rectangle(image, pt1, pt2, color, thickness)` – Draw a rectangle - `cv2.circle(image, center, radius, color, thickness)` – Draw a circle - `cv2.putText(image, text, org, font, fontScale, color, thickness)` - Draw Text

—– 5. Image Processing - `cv2.GaussianBlur(image, (ksize, ksize), sigmaX)` – Apply Gaussian blur (`ksize`: kernel size) - `cv2.medianBlur(image, ksize)` – Apply median blur - `cv2.Canny(image, threshold1, threshold2)` – Apply Canny edge detection

—– 6. Morphological Transformations - `cv2.erode(image, kernel, iterations)` – Erosion - `cv2.dilate(image, kernel, iterations)` – Dilation - `cv2.morphologyEx(image, op, kernel)` – Morphological operations

—– 7. Contour Detection - `cv2.findContours(image, mode, method)` – Find contours - `cv2.drawContours(image, contours, contourIdx, color, thickness)` – Draw contours

—– 8. Face & Object Detection - `cv2.CascadeClassifier('path_to_haar_cascade.xml')` – Load a Haar cascade - `detectMultiScale(image, scaleFactor, minNeighbors, minSize)` – Detect objects

—– 9. Feature Detection - `cv2.SIFT_create()` – SIFT feature detection - `cv2.ORB_create()` – ORB feature detection - `cv2.BFMatcher()` – Brute-Force Matcher

—– 10. Thresholding - `cv2.threshold(image, thresh, maxval, type)` – Apply threshold - `cv2.adaptiveThreshold(image, maxValue, adaptiveMethod,` `thresholdType, blockSize, C)` – Adaptive thresholding

—— 11. Histogram Operations - `cv2.calcHist([image], channels, mask, histSize, ranges)` – Calculate histogram - `cv2.equalizeHist(image)` – Histogram equalization

—— 12. Camera Calibration & Perspective Transform - `cv2.undistort(image, cameraMatrix, distCoeffs)` – Remove distortion - `cv2.getPerspectiveTransform(pts1, pts2)` – Perspective transform

—— 13. Optical Flow - `cv2.calcOpticalFlowPyrLK(prevImg, nextImg, prevPts, nextPts, status, err)` – Lucas-Kanade Optical Flow

—— 14. Machine Learning with OpenCV - `cv2.ml.KNearest_create()` – K-Nearest Neighbors - `cv2.ml.SVM_create()` – Support Vector Machine - `cv2.ml.DTrees_create()` – Decision Trees

---

—— MediaPipe Modules ——# 1. Hands Detection - `mp.solutions.hands.Hands(min_detection_confidence, min_tracking_confidence)` – Detect hands - `mp_drawing.draw_landmarks(image, hand_landmarks, mp_hands.HAND_CONNECTIONS)` – Draw hand landmarks

——# 2. Face Detection - `mp.solutions.face_detection.FaceDetection(min_detection_confidence)` – Detect faces - `face_detection.process(image)` – Process the image for face detection

——# 3. Pose Estimation - `mp.solutions.pose.Pose(min_detection_confidence, min_tracking_confidence)` – Detect body pose - `mp_drawing.draw_landmarks(image, pose_landmarks, mp_pose.POSE_CONNECTIONS)` – Draw body pose landmarks

——# 4. Holistic Detection (Face, Hands, Pose Together) - `mp.solutions.holistic.Holistic(min_detection` `min_tracking_confidence)` – Detect multiple body parts

——# 5. Selfie Segmentation - `mp.solutions.selfie_segmentation.SelfieSegmentation(model_selection=` – Segment the background

OpenCV Cheatsheet

1. Getting Started

Installation

- **Windows:**

  ```
  pip install opencv-python
  pip install opencv-contrib-python
  ```

- **Linux:**

  ```
  sudo apt update && sudo apt install python3-opencv
  ```

- **Anaconda:**

  ```
  conda install -c conda-forge opencv
  ```

2. Working with Images

2.1 Getting Started

- **Read an Image:**

  ```
  img = cv2.imread('image.jpg')
  ```

- **Display an Image:**

  ```
  cv2.imshow('Image', img)
  cv2.waitKey(0)
  cv2.destroyAllWindows()
  ```

- **Write an Image:**

  ```
  cv2.imwrite('output.jpg', img)
  ```

- **Convert Color Spaces:**

  ```
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  ```

- **Arithmetic Operations:**

  ```
  added = cv2.add(img1, img2)
  subtracted = cv2.subtract(img1, img2)
  ```

- **Bitwise Operations:**

  ```
  bitwise_and = cv2.bitwise_and(img1, img2)
  bitwise_or = cv2.bitwise_or(img1, img2)
  ```

2.2 Image Processing

- **Resizing:**

  ```
  resized = cv2.resize(img, (width, height))
  ```

- **Blurring:**

  ```
  blurred = cv2.GaussianBlur(img, (5,5), 0)
  ```

- **Edge Detection:**

  ```
  edges = cv2.Canny(img, 100, 200)
  ```

- **Histogram Equalization:**

  ```
  equalized = cv2.equalizeHist(gray)
  ```

- **Thresholding:**

  ```
  _, threshold = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
  ```

- **Morphological Operations:**

  ```
  kernel = np.ones((5,5), np.uint8)
  eroded = cv2.erode(img, kernel, iterations=1)
  dilated = cv2.dilate(img, kernel, iterations=1)
  ```

2.3 Feature Detection

- **Hough Line Detection:**

  ```
  lines = cv2.HoughLines(edges, 1, np.pi/180, 200)
  ```

- **Circle Detection:**

  ```
  circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 20)
  ```

- **Corner Detection:**

  ```
  corners = cv2.goodFeaturesToTrack(gray, 25, 0.01, 10)
  ```

2.4 Drawing Functions

- **Draw Shapes:**

  ```
  cv2.line(img, (0, 0), (250, 250), (255, 0, 0), 3)
  cv2.rectangle(img, (50, 50), (200, 200), (0, 255, 0), 2)
  cv2.circle(img, (150, 150), 50, (0, 0, 255), -1)
  ```

- **Draw Text:**

  ```
  cv2.putText(img, 'OpenCV', (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
  ```

3. Working with Videos

3.1 Getting Started

- **Play a Video:**

```
cap = cv2.VideoCapture('video.mp4')
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imshow('Video', frame)
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

3.2 Video Processing

- **Extract Frames:**

```
success, image = cap.read()
count = 0
while success:
    cv2.imwrite(f"frame{count}.jpg", image)
    success, image = cap.read()
    count += 1
```

- **Save Processed Video:**

```
out = cv2.VideoWriter('output.avi', cv2.VideoWriter_fourcc(*'XVID'), 20.0, (640, 480))
out.write(frame)
out.release()
```

4. Applications and Projects

- **Face Detection:**

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_d
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
```

- **Live Webcam Face Detection:**

```
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 3)
    cv2.imshow('Face Detection', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

- **Template Matching:**

  ```
  result = cv2.matchTemplate(img, template, cv2.TM_CCOEFF_NORMED)
  min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
  top_left = max_loc
  ```

- **Vehicle Detection:**

  ```
  car_cascade = cv2.CascadeClassifier('cars.xml')
  cars = car_cascade.detectMultiScale(gray, 1.1, 1)
  ```

This cheatsheet provides essential OpenCV functions for quick reference!

Matplotlib Cheatsheet

Matplotlib Intro

Matplotlib is a Python library for creating static, animated, and interactive visualizations.

Matplotlib Get Started

Install Matplotlib using:

```
pip install matplotlib
```

Importing Matplotlib:

```
import matplotlib.pyplot as plt
```

Matplotlib Pyplot

Pyplot is a module in Matplotlib providing a MATLAB-like interface.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```

Matplotlib Plotting

Basic plotting:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.show()
```

Matplotlib Markers

Markers highlight data points in a plot.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30], marker='o')
plt.show()
```

Matplotlib Line

Customize line styles and colors.

```
plt.plot([1, 2, 3, 4], [10, 20, 25, 30], linestyle='dashed', color='red')
plt.show()
```

Matplotlib Labels

Adding labels to axes.

```
plt.xlabel('X Axis Label')
plt.ylabel('Y Axis Label')
plt.title('Title of the Graph')
plt.show()
```

Matplotlib Grid

Enable grid in the plot.

```
plt.grid(True)
plt.show()
```

Matplotlib Subplot

Create multiple subplots in a single figure.

```
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot([1, 2, 3, 4], [10, 20, 25, 30])
axs[1, 1].plot([1, 2, 3, 4], [30, 25, 20, 10])
plt.show()
```

Matplotlib Scatter

Scatter plot for visualizing point data.

```
plt.scatter([1, 2, 3, 4], [10, 20, 25, 30])
plt.show()
```

Matplotlib Bars

Bar chart representation.

```
plt.bar([1, 2, 3, 4], [10, 20, 25, 30])
plt.show()
```

Matplotlib Histograms

Histogram to visualize frequency distribution.

```
import numpy as np
data = np.random.randn(1000)
plt.hist(data, bins=30)
plt.show()
```

Matplotlib Pie Charts

Pie chart for categorical data.

```
labels = ['A', 'B', 'C', 'D']
sizes = [10, 20, 30, 40]
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.show()
```

NumPy Cheatsheet

NumPy Tutorial

NumPy HOME

NumPy (Numerical Python) is a library for numerical computing in Python. It provides support for arrays, matrices, and mathematical operations.

NumPy Intro

- Core library for numerical computing.
- Provides high-performance multidimensional arrays.
- Supports mathematical functions, linear algebra, and more.
- Uses C-based optimizations for speed.

NumPy Getting Started

- Install NumPy using pip:

  ```
  pip install numpy
  ```

- Importing NumPy:

  ```
  import numpy as np
  ```

NumPy Arrays

Creating Arrays

- Create a NumPy array:

  ```
  arr = np.array([1, 2, 3, 4])
  ```

- Create an array with a specific shape:

  ```
  arr = np.zeros((2, 3))  # 2x3 matrix of zeros
  ```

Array Indexing

- Access elements:

  ```
  arr[0]  # First element
  ```

- Accessing multiple elements:

  ```
  arr[1:3]  # Slice from index 1 to 2
  ```

Array Slicing

- Select a subarray:

  ```
  arr[:, 1]  # All rows, second column
  ```

Data Types

- Check data type:

```
arr.dtype
```

- Convert data type:

```
arr.astype(float)
```

Copy vs View

- Copy creates a new object:

```
new_arr = arr.copy()
```

- View refers to the same object:

```
view_arr = arr.view()
```

Array Shape

- Check shape:

```
arr.shape
```

Array Reshape

- Change shape without modifying data:

```
arr.reshape(3, 2)  # Convert to 3x2
```

Array Iterating

- Loop through elements:

```
for x in np.nditer(arr):
    print(x)
```

Array Join

- Concatenate arrays:

```
np.concatenate((arr1, arr2), axis=0)
```

Array Split

- Split an array:

```
np.array_split(arr, 3)
```

Array Search

- Find an element:

  ```
  np.where(arr == 4)
  ```

Array Sort

- Sort array elements:

  ```
  np.sort(arr)
  ```

Array Filter

- Filter elements based on a condition:

  ```
  arr[arr > 5]
  ```

NumPy Random

Random Intro

- Generate random numbers using NumPy's random module:

  ```
  np.random.rand(5)  # 5 random values
  ```

Data Distribution

- Generate normal distribution:

  ```
  np.random.normal(0, 1, 10)  # Mean=0, StdDev=1, 10 values
  ```

Random Permutation

- Shuffle array elements:

  ```
  np.random.permutation(arr)
  ```

Seaborn Module

- Seaborn is used for statistical visualizations:

  ```
  import seaborn as sns
  sns.histplot(data)
  ```

Probability Distributions

Normal Distribution

- Generate normally distributed values:

  ```
  np.random.normal(0, 1, 1000)
  ```

Binomial Distribution

- Binomial probability simulation:

  `np.random.binomial(n=10, p=0.5, size=100)`

Poisson Distribution

- Generate Poisson-distributed values:

  `np.random.poisson(lam=3, size=1000)`

Uniform Distribution

- Generate uniform values between 0 and 1:

  `np.random.uniform(0, 1, 100)`

Logistic Distribution

- Used for binary classification modeling:

  `np.random.logistic(0, 1, 100)`

Multinomial Distribution

- Simulate multi-outcome events:

  `np.random.multinomial(10, [0.2, 0.3, 0.5])`

Exponential Distribution

- Used for time until an event occurs:

  `np.random.exponential(scale=2, size=100)`

Chi-Square Distribution

- Used in statistical tests:

  `np.random.chisquare(df=2, size=100)`

Rayleigh Distribution

- Used in signal processing:

  `np.random.rayleigh(scale=2, size=100)`

Pareto Distribution

- Used in economics and finance:

  `np.random.pareto(a=3, size=100)`

Zipf Distribution

- Used in linguistics and social sciences:

  ```
  np.random.zipf(a=2, size=100)
  ```

NumPy ufunc (Universal Functions)

Ufunc Introduction

- NumPy universal functions (ufuncs) operate on ndarrays:

  ```
  np.add(arr1, arr2)
  ```

  Common Ufuncs & Parameters

- `np.add(x1, x2)`: Element-wise addition.
- `np.subtract(x1, x2)`: Element-wise subtraction.
- `np.multiply(x1, x2)`: Element-wise multiplication.
- `np.divide(x1, x2)`: Element-wise division.
- `np.round(x, decimals=0)`: Round elements.
- `np.floor(x)`: Round down.
- `np.ceil(x)`: Round up.
- `np.log(x)`, `np.log10(x)`: Logarithms.
- `np.sum(a, axis=None)`: Sum elements.
- `np.prod(a, axis=None)`: Product of elements.
- `np.diff(a, n=1)`: Discrete differences.
- `np.lcm(x1, x2)`: Least common multiple.
- `np.gcd(x1, x2)`: Greatest common divisor.
- `np.sin(x)`, `np.cos(x)`, `np.tan(x)`: Trigonometric functions.
- `np.sinh(x)`, `np.cosh(x)`: Hyperbolic functions.
- `np.union1d(arr1, arr2)`: Union of sets.
- `np.intersect1d(arr1, arr2)`: Intersection of sets.

This cheatsheet summarizes key NumPy functions and probability distributions.

TensorFlow with OpenCV Cheatsheet

Installing Dependencies

```
pip install tensorflow opencv-python numpy
```

Importing Required Libraries

```
import tensorflow as tf
import cv2
import numpy as np
```

Loading a Pre-Trained Model

```
model = tf.keras.models.load_model('model.h5')
```

Capturing an Image

```
cap = cv2.VideoCapture(0)
ret, frame = cap.read()
cap.release()
cv2.imwrite('captured_image.jpg', frame)
```

Preprocessing Image

```
image = cv2.imread('captured_image.jpg')
image = cv2.resize(image, (224, 224))
image = image / 255.0
image = np.expand_dims(image, axis=0)
image = np.array(image, dtype=np.float32)
```

Making Predictions

```
predictions = model.predict(image)
print(predictions)
```

Face Detection

```
face_cascade =
cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
image_gray = cv2.cvtColor(image[0] * 255,
cv2.COLOR_BGR2GRAY).astype(np.uint8)
faces = face_cascade.detectMultiScale(image_gray,
scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
for (x, y, w, h) in faces:
     cv2.rectangle(image[0], (x, y), (x+w, y+h), (255,
0, 0), 2)
cv2.imshow('Face Detection',
image[0].astype(np.uint8))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Object Detection

```
net =
cv2.dnn.readNetFromTensorflow('ssd_mobilenet_v2.pb',
'ssd_mobilenet_v2.pbtxt')
blob = cv2.dnn.blobFromImage(image[0] * 255,
size=(300, 300), swapRB=True, crop=False)
net.setInput(blob)
output = net.forward()
```

Running Model on Video

```
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    if not ret:
        break
     processed_frame = cv2.resize(frame, (224, 224)) /
255.0
     processed_frame = np.expand_dims(processed_frame,
axis=0)
         processed_frame = np.array(processed_frame,
dtype=np.float32)
    prediction = model.predict(processed_frame)
       cv2.imshow('Live Video', (processed_frame[0] *
255).astype(np.uint8))
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Saving Processed Image

```
cv2.imwrite('output.jpg', (processed_frame[0] *
255).astype(np.uint8))
```