LOW PASS FILTER:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the uploaded image (Ensure the filename is correct)
image = cv2.imread('panda.png', cv2.IMREAD_COLOR)
# Check if the image is loaded correctly
if image is None:
    raise FileNotFoundError("Error: Image 'panda.png' not found. Ensure it is uploaded in the notebook.")
# Convert the image from BGR to RGB for correct display in Matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Apply Gaussian Blur (Low-Pass Filter)
gaussian_blur = cv2.GaussianBlur(image, (15, 15), 0)  # Kernel size: 15x15
# Apply Averaging Filter (Low-Pass Filter)
averaging_blur = cv2.blur(image, (15, 15))  # Kernel size: 15x15
# Apply Median Blur
median_blur = cv2.medianBlur(image, 15)  # Kernel size: 15 (must be an odd number)
# Display the original and filtered images
plt.figure(figsize=(10, 8))
# Original image
plt.subplot(2, 2, 1)
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis("off")
# Gaussian Blur
plt.subplot(2, 2, 2)
plt.imshow(cv2.cvtColor(gaussian_blur, cv2.COLOR_BGR2RGB))
plt.title("Gaussian Blur")
plt.axis("off")
# Averaging Blur
plt.subplot(2, 2, 3)
plt.imshow(cv2.cvtColor(averaging_blur, cv2.COLOR_BGR2RGB))
plt.title("Averaging Blur")
plt.axis("off")
# Median Blur
plt.subplot(2, 2, 4)
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
plt.title("Median Blur")
plt.axis("off")
plt.tight_layout()
plt.show()
```

```
#SURF (SPEEDED-UP ROBUST FEATURES)ALGORITHM
import cv2
import matplotlib.pyplot as plt
# Load the image in grayscale
image = cv2.imread("panda.png", cv2.IMREAD_GRAYSCALE)
# Resize the image (optional, for better visualization)
image = cv2.resize(image, (600, 400))
# Use SIFT instead of SURF
sift = cv2.SIFT_create()
# Detect key points and compute descriptors
keypoints, descriptors = sift.detectAndCompute(image, None)
# Draw the key points on the image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, color=(0, 255, 0)
)
# Display the original image and the image with key points
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap="gray")
plt.axis("off")
# Image with Key Points
plt.subplot(1, 2, 2)
plt.title("Image with Key Points (SIFT)")
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.tight_layout()
plt.show()
```

## #corner and edge

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the image in grayscale
image = cv2.imread("panda.png", cv2.IMREAD_GRAYSCALE)
# Resize the image for consistent processing (optional)
image = cv2.resize(image, (600, 400))
# Step 1: Edge Detection (Canny Edge Detector)
edges = cv2.Canny(image, threshold1=100, threshold2=200)
# Step 2: Corner Detection (Harris Corner Detector)
# Convert image to float32 (required for Harris Corner Detection)
gray_float = np.float32(image)
# Apply Harris Corner Detection
corners = cv2.cornerHarris(src=gray_float, blockSize=2, ksize=3, k=0.04)
# Dilate corner detections for better visualization
corners = cv2.dilate(corners, None)
# Threshold to highlight strong corners (corner strength > threshold)
corner_image = np.copy(image)
corner_image[corners > 0.01 * corners.max()] = 255
# Step 3: Overlay detected corners on the original image
overlay = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
overlay[corners > 0.01 * corners.max()] = [0, 0, 255]  # Red corners
# Display results using Matplotlib
plt.figure(figsize=(12, 8))
# Original Image
plt.subplot(2, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap="gray")
plt.axis("off")
```

```
# Edges
plt.subplot(2, 2, 2)
plt.title("Edges (Canny)")
plt.imshow(edges, cmap="gray")
plt.axis("off")
plt.subplot(2, 2, 3)
plt.title("Corners (Harris)")
plt.imshow(corner_image, cmap="gray")
plt.axis("off")
# Overlay Image
plt.subplot(2, 2, 4)
plt.title("Overlay of Corners on Image")
plt.imshow(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.tight_layout()
plt.show()
```

C)FILTER THREE (For Visualizing Feature Maps of VGG16's First Convolutional Layer)

```
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input from
 keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array from
keras.models import Model
from matplotlib import pyplot from numpy impor
t expand_dims # load the model
model = VGG16()
# redefine model to output right after the first hidden layer
model = Model(inputs=model.inputs, outputs=model.layers[1].output)
 model.summary()
img = load_img("C:\\Users\\3122246002016\\Downloads\\flower.jpg",
target_size=(224, 224)) # convert the image to an array
img = img_to_array(img)
# expand dimensions so that it represents a single
 'sample' img = expand_dims(img, axis=0)
# prepare the image (e.g. scale pixel values for the vgg)
img = preprocess_input(img)
# get feature map for first hidden layer feature_maps = model.predict(img)
# plot all 64 maps in an 8x8 squares square = 8
ix = 1
for _ in range(square): for _ in range(square):
# specify subplot and turn of axis
ax = pyplot.subplot(square, square, ix) ax.set_xticks([])
ax.set_yticks([])
# plot filter channel in grayscale
 pyplot.imshow(feature_maps[0, :, :, ix-1], cmap='gray') ix += 1
# show the figure
pyplot.show()
```

```
# HORIZONTAL FLIP AUGMENTATION
from numpy import expand_dims
from tensorflow.keras.preprocessing.image
 import load_img, img_to_array, ImageDataGenerator
from matplotlib import pyplot
# Load the image
img = load_img('cat.jpeg')
# Convert to numpy array
data = img_to_array(img)
# Expand dimension to one sample
samples = expand_dims(data, 0)
# Create image data augmentation generator
datagen = ImageDataGenerator(horizontal_flip=True)
# Prepare iterator
it = datagen.flow(samples, batch_size=1)
# Generate samples and plot
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = next(it)
    image = batch[0].astype('uint8')
pyplot.imshow(image)
pyplot.show()
#histogram equalized visualition
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('image.jpeg', cv2.IMREAD_GRAYSCALE)
if image is None:
print("Image not found!")
exit()
equalized_image = cv2.equalizeHist(image)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(equalized_image, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')
plt.show()
cv2.imwrite('equalized_image.jpg', equalized_image)
```

```
#image caption generation
import numpy as np
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
import pickle
with open('tokenizer.pkl', 'rb') as f:
tokenizer = pickle.load(f)
max_length = 34
model = load_model('model.h5')
 def extract_features(image_path):
base_model = VGG16()
model_vgg = Model(inputs=base_model.inputs, outputs=base_model.layers[-2].output)
image = load_img(image_path, target_size=(224, 224))
image = img_to_array(image)
image = np.expand_dims(image, axis=0)
image = preprocess_input(image)
feature = model_vgg.predict(image, verbose=0)
return feature
def generate_caption(model, tokenizer, photo, max_length):
in_text = 'startseq'
for _ in range(max_length):
sequence = tokenizer.texts_to_sequences([in_text])[0]
sequence = pad_sequences([sequence], maxlen=max_length)
yhat = model.predict([photo, sequence], verbose=0)
yhat = np.argmax(yhat)
word = None
for w, index in tokenizer.word_index.items():
if index == yhat:
word = w
break
if word is None:
break
in_text += ' ' + word
if word == 'endseq':
 break
final_caption = in_text.replace('startseq', '').replace('endseq', '').strip()
return final_caption
```

```
image_paths = ['image1.png','image2.png','image3.png','image4.png']
for path in image_paths:
photo_feature = extract_features(path)
caption = generate_caption(model, tokenizer, photo_feature, max_length)
print(f"{path}: {caption}")
```

```
HIGH PASS FILTER:
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the uploaded image in grayscale
image = cv2.imread('panda.png', cv2.IMREAD_GRAYSCALE)
# Check if the image is loaded correctly
if image is None:
    raise FileNotFoundError("Error: Image 'panda.png' not found. ")
# Apply High-Pass Filter using Laplacian
laplacian = cv2.Laplacian(image, cv2.CV_64F)  # Use 64-bit float for higher precision
laplacian = np.uint8(np.absolute(laplacian))  # Convert to unsigned 8-bit
sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)  # Derivative in X direction
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)  # Derivative in Y direction
sobel = cv2.magnitude(sobel_x, sobel_y)  # Combine Sobel X and Y
sobel = np.uint8(np.absolute(sobel))  # Convert to unsigned 8-bit
# Display the original and filtered images
plt.figure(figsize=(10, 6))
# Original image
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.axis("off")
# Laplacian High-Pass Filter
plt.subplot(1, 3, 2)
plt.imshow(laplacian, cmap='gray')
plt.title("Laplacian High-Pass Filter")
plt.axis("off")
plt.subplot(1, 3, 3)
plt.imshow(sobel, cmap='gray')
plt.title("Sobel High-Pass Filter")
plt.axis("off")
plt.tight_layout()
plt.show()
```

```
#SHIFT INVARIANT FEATURE TRANSFORM
import cv2
import matplotlib.pyplot as plt
# Load the image in grayscale
image = cv2.imread("panda.png", cv2.IMREAD_GRAYSCALE)
# Resize the image (optional, for better visualization)
image = cv2.resize(image, (600, 400))
# Initialize the SIFT detector
sift = cv2.SIFT_create()
# Detect key points and compute descriptors
keypoints, descriptors = sift.detectAndCompute(image, None)
# Draw the key points on the image
image_with_keypoints = cv2.drawKeypoints(
image, keypoints, None,
 flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, color=(0, 255, 0))
# Display the original image and the image with key points
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap="gray")
plt.axis("off")
# Image with Key Points
plt.subplot(1, 2, 2)
plt.title("Image with Key Points (SIFT)")
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.tight_layout()
plt.show()
```

```
# ONE (To Extract and Visualize Filters from VGG16
from keras.applications.vgg16 import VGG16 from matplotlib import pyplot
# load the model model = VGG16()
# summarize filter shapes for layer in model.layers:
# check for convolutional layer
if 'conv' not in layer.name: continue
# get filter weights
filters, biases = layer.get_weights() print(layer.name, filters.shape)

#B)TWO (To Extract and Visualize Filters from VGG16's Second Convolutional Layer)
from keras.applications.vgg16 import VGG16 from matplotlib import pyplot
# load the model model = VGG16()
# retrieve weights from the second hidden layer
#filters, biases = model.layers[1].get_weights()
# normalize filter values to 0-1 so we can visualize
#them f_min, f_max = filters.min(), filters.max()
filters = (filters - f_min) / (f_max - f_min) # plot first few filters
n_filters, ix = 6, 1
for i in range(n_filters): # get the filter
f = filters[:, :, :, i]
# plot each channel separately for j in range(3):
# specify subplot and turn of axis ax = pyplot.subplot(n_filters, 3, ix) ax.set_xticks([])
ax.set_yticks([])
# plot filter channel in grayscale pyplot.imshow(f[:, :, j], cmap='gray')
ix += 1
# show the figure pyplot.show()
```

```
FILTER FOUR (For Visualizing Feature Maps from Multiple Layers of VGG16)
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
 from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array from keras.models import Model
from matplotlib import pyplot from numpy import expand_dims # load the model
model = VGG16()
# redefine model to output right after the first hidden layer ixs = [2, 5, 9, 13, 17]
outputs = [model.layers[i].output for i in ixs]
model = Model(inputs=model.inputs, outputs=outputs) # load the img of required shape
img = load_img("C:\\Users\\3122246002016\\Downloads\\flower.jpg",
 target_size=(224, 224)) # convert the image to an array
img = img_to_array(img)
# expand dimensions so that it represents a single
 'sample' img = expand_dims(img, axis=0)
# prepare the image (e.g. scale pixel values for the vgg) img = preprocess_input(img)
# get feature map for first hidden layer
feature_maps = model.predict(img) # plot the output from each block square = 8
for fmap in feature_maps:
# plot all 64 maps in an 8x8 squares ix = 1
for _ in range(square): for _ in range(square):
# specify subplot and turn of axis
ax = pyplot.subplot(square, square, ix) ax.set_xticks([])
ax.set_yticks([])
# plot filter channel in grayscale pyplot.imshow(fmap[0, :, :, ix-1], cmap='gray') ix += 1
# show the figure
pyplot.show()
```

```
from numpy import expand_dims
from tensorflow.keras.preprocessing.image
import load_img, img_to_array, ImageDataGenerator
from matplotlib import pyplot as plt

# === CONFIGURATION ===
mode = "vertical"  # Choose either "vertical" or "horizontal"

# === Load the image ===
img = load_img('cat.jpeg')  # Replace with your image path
data = img_to_array(img)
samples = expand_dims(data, 0)

# === Define augmentation based on mode ===
if mode == "vertical":
    # VERTICAL SHIFT AUGMENTATION
    datagen = ImageDataGenerator(height_shift_range=0.5)
elif mode == "horizontal":
    # HORIZONTAL SHIFT AUGMENTATION
    datagen = ImageDataGenerator(width_shift_range=[-200, 200])
else:
    raise ValueError("Invalid mode. Choose 'vertical' or 'horizontal'.")

# === Prepare iterator ===
it = datagen.flow(samples, batch_size=1)

# === Generate and plot 9 augmented images ===
for i in range(9):
    plt.subplot(330 + 1 + i)
    batch = next(it)
    image = batch[0].astype('uint8')
    plt.imshow(image)
    plt.axis('off')

plt.suptitle(f"{mode.capitalize()} Shift Augmentation", fontsize=14)
plt.tight_layout()
plt.show()
```

```
#MULTI OBJECT DETECTION
import cv2
import numpy as np
import os
from IPython.display import display
from PIL import Image
# Paths
img_path = "C:\\Users\\3122246002016\\Downloads\\image.jpg"
proto = "C:\\Users\\3122246002016\\Downloads\\MobileNetSSD_deploy.prototxt"
model = "C:\\Users\\3122246002016\\Downloads\\MobileNetSSD_deploy.caffemodel"
# Check files
for path in [img_path, proto, model]:
    if not os.path.exists(path): raise FileNotFoundError(f"{path} not found")
# Load model & image
net = cv2.dnn.readNetFromCaffe(proto, model)
img = cv2.imread(img_path)
h, w = img.shape[:2]
resized = cv2.resize(img, (800, 600))
x_scale, y_scale = 800/w, 600/h

# Blob & detection
blob = cv2.dnn.blobFromImage(cv2.resize(img, (300, 300)), 0.007843, (300, 300), 127.5)
net.setInput(blob)
detections = net.forward()
# Labels & colors
CLASSES = ["background", "aeroplane", "bicycle", "bird",
 "boat", "bottle", "bus", "car", "cat", "chair",
           "cow", "diningtable", "dog", "horse", "motorbike",
"person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]
colors = np.random.uniform(0, 255, size=(len(CLASSES), 3))
# Draw boxes
for i in range(detections.shape[2]):
    conf = detections[0, 0, i, 2]
    if conf > 0.2:
        idx = int(detections[0, 0, i, 1])
        box = (detections[0, 0, i, 3:7] * np.array([w, h, w, h])).astype("int")
        sx, sy, ex, ey = (box * [x_scale, y_scale, x_scale, y_scale]).astype("int")
        sx, sy, ex, ey = np.clip([sx, sy, ex, ey], 0, [799, 599, 799, 599])
        label = f"{CLASSES[idx]}: {conf*100:.2f}%"
        print("[INFO]", label)
        cv2.rectangle(resized, (sx, sy), (ex, ey), colors[idx], 2)
        cv2.putText(resized, label, (sx, sy - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, colors[idx], 2)
# Show image
```