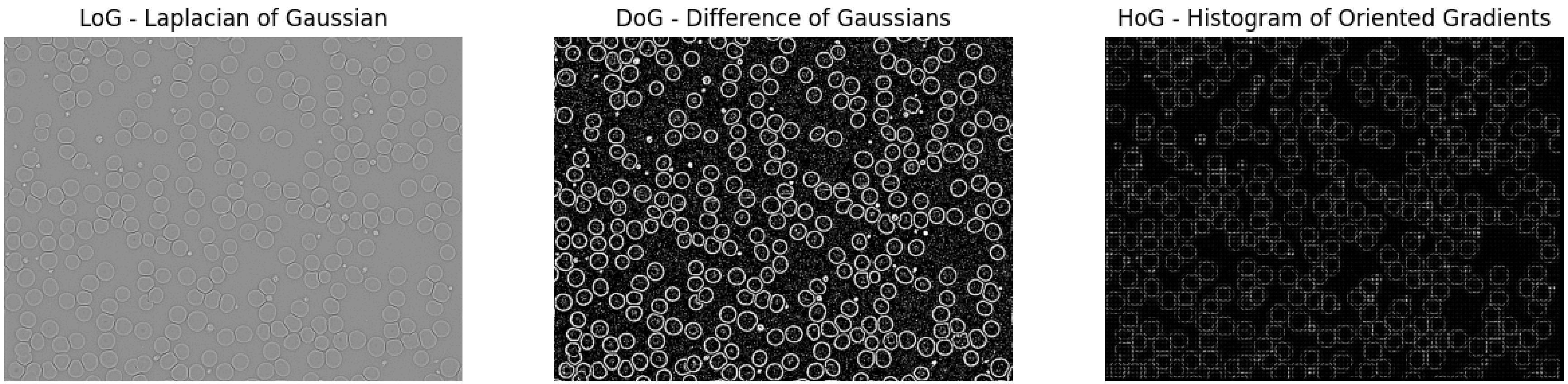
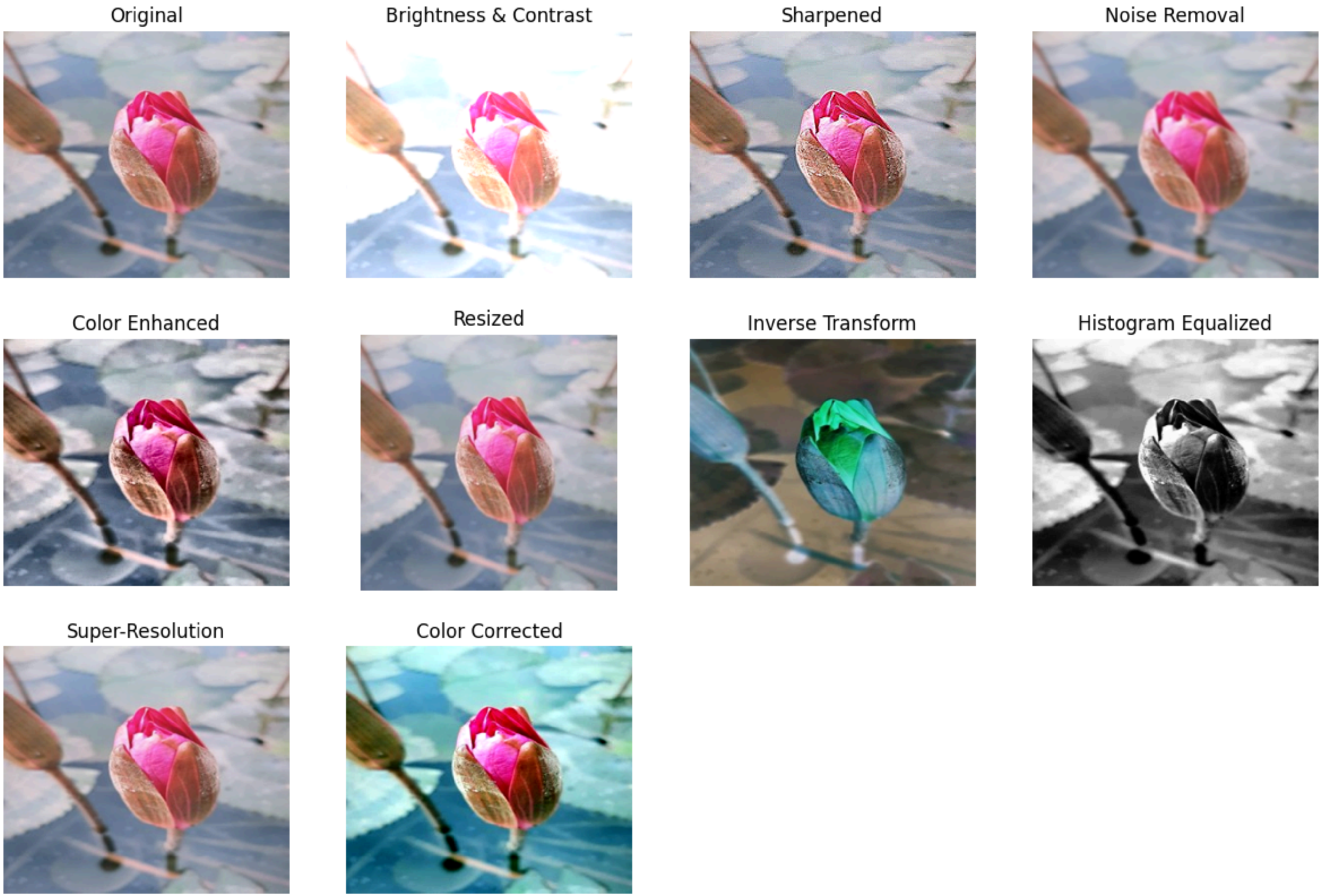
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import exposure

# Load Image
image = cv2.imread("/content/Picture1.jpg", cv2.IMREAD_GRAYSCALE)

# --- 1. LoG (Laplacian of Gaussian) ---
log = cv2.GaussianBlur(image, (3, 3), 0)
log = cv2.Laplacian(log, cv2.CV_64F)

# --- 2. DoG (Difference of Gaussians) ---
gauss1 = cv2.GaussianBlur(image, (5, 5), 1)
gauss2 = cv2.GaussianBlur(image, (5, 5), 2)
dog = gauss1 - gauss2

# --- 3. HoG (Histogram of Oriented Gradients) ---
fd, hog_image = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)
hog_image = exposure.rescale_intensity(hog_image, in_range=(0, 10))

# --- Visualization ---
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].imshow(log, cmap="gray")
axs[0].set_title("LoG - Laplacian of Gaussian")
axs[1].imshow(dog, cmap="gray")
axs[1].set_title("DoG - Difference of Gaussians")
axs[2].imshow(hog_image, cmap="gray")
axs[2].set_title("HoG - Histogram of Oriented Gradients")

for ax in axs:
    ax.axis("off")
plt.show()
```




```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread("/content/Picture2.jpg")

# Convert to grayscale (for certain operations)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# --- 1. Adjust Brightness & Contrast ---
alpha = 1.5  # Contrast (1.0-3.0)
beta = 30    # Brightness (-100 to 100)
bright_contrast = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

# --- 2. Sharpening ---
kernel_sharp = np.array([[0, -1, 0],
                          [-1, 5, -1],
                          [0, -1, 0]])
sharpened = cv2.filter2D(image, -1, kernel_sharp)

# --- 3. Noise Removal (Gaussian Blur) ---
noise_removed = cv2.GaussianBlur(image, (5, 5), 0)

# --- 4. Color Enhancement ---
lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
l, a, b = cv2.split(lab)
clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
l = clahe.apply(l)
enhanced_color = cv2.merge([l, a, b])
```

```
enhanced_color = cv2.cvtColor(enhanced_color, cv2.COLOR_LAB2BGR)

# --- 5. Resizing & Scaling ---
resized = cv2.resize(image, (400, 400), interpolation=cv2.INTER_CUBIC)

# --- 6. Inverse Transform (Negative) ---
inverse_transform = cv2.bitwise_not(image)

# --- 7. Histogram Equalization ---
equalized_gray = cv2.equalizeHist(gray)

# --- 8. Super-Resolution (Using OpenCV DNN) ---
super_res_image = cv2.resize(image, (image.shape[1] * 4, image.shape[0] * 4), interpolation=cv2.INTER_CUBIC)

# --- 9. Color Correction (White Balance) ---
wb = cv2.xphoto.createSimpleWB()
color_corrected = wb.balanceWhite(image)

# --- Visualization ---
titles = ["Original", "Brightness & Contrast", "Sharpened", "Noise Removal",
          "Color Enhanced", "Resized", "Inverse Transform", "Histogram Equalized",
          "Super-Resolution", "Color Corrected"]
images = [image, bright_contrast, sharpened, noise_removed, enhanced_color,
          resized, inverse_transform, equalized_gray, super_res_image, color_corrected]

plt.figure(figsize=(15, 10))
for i in range(len(images)):
    plt.subplot(3, 4, i+1)
    plt.imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))
    plt.title(titles[i])
    plt.axis("off")
plt.show()
```



TASK 2

```
# @title TASK 2
```

```
#Load Dataset
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time

# Load CIFAR-100 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar100.load_data()

# Normalize pixel values to [0, 1]
train_images, test_images = train_images / 255.0, test_images / 255.0

# CIFAR-100 has 100 classes
num_classes = 100

# Convert labels to categorical
train_labels = tf.keras.utils.to_categorical(train_labels, num_classes)
test_labels = tf.keras.utils.to_categorical(test_labels, num_classes)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169001437/169001437 ━━━━━━━━━━━━━━━━━━━━ 19s 0us/step
```

```
#Alexnet Model
def alexnet_model(input_shape=(32, 32, 3), num_classes=100):
    model = models.Sequential([
        tf.keras.layers.Resizing(128, 128, interpolation="bilinear"),
        layers.Conv2D(96, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(256, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(384, (3, 3), activation='relu'),
        layers.Conv2D(384, (3, 3), activation='relu'),
        layers.Conv2D(256, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        layers.Flatten(),
        layers.Dense(4096, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(4096, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
```

```
    ])
    return model

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

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
#VGG16
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten

# Load VGG16 (without the classification layer)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Add custom layers
x = Flatten()(base_model.output)
x = Dense(512, activation='relu')(x)
x = Dense(num_classes, activation='softmax')(x)

# Create VGG16 model
vgg16 = Model(inputs=base_model.input, outputs=x)
vgg16.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
epochs = 5
batch_size = 64

# Train AlexNet
start_time = time.time()
alexnet.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_data=(test_images, test_labels))
alexnet_time = time.time() - start_time
```

```
Epoch 1/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 228s 251ms/step - accuracy: 0.0085 - loss: 4.6290 - val_accuracy: 0.0100 - val_loss: 4.6052
```

```
Epoch 2/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 233s 241ms/step - accuracy: 0.0095 - loss: 4.6055 - val_accuracy: 0.0100 - val_loss: 4.6052
Epoch 3/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 202s 241ms/step - accuracy: 0.0109 - loss: 4.6055 - val_accuracy: 0.0100 - val_loss: 4.6052
Epoch 4/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 202s 241ms/step - accuracy: 0.0100 - loss: 4.6054 - val_accuracy: 0.0100 - val_loss: 4.6052
Epoch 5/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 192s 229ms/step - accuracy: 0.0094 - loss: 4.6055 - val_accuracy: 0.0100 - val_loss: 4.6052
```

```
# Train VGG16
start_time = time.time()
vgg16.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_data=(test_images, test_labels))
vgg16_time = time.time() - start_time
```

```
Epoch 1/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 61s 63ms/step - accuracy: 0.0077 - loss: 4.6127 - val_accuracy: 0.0100 - val_loss: 4.6052
Epoch 2/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 66s 53ms/step - accuracy: 0.0097 - loss: 4.6055 - val_accuracy: 0.0100 - val_loss: 4.6052
Epoch 3/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 52ms/step - accuracy: 0.0091 - loss: 4.6055 - val_accuracy: 0.0100 - val_loss: 4.6052
Epoch 4/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 40s 51ms/step - accuracy: 0.0091 - loss: 4.6055 - val_accuracy: 0.0100 - val_loss: 4.6052
Epoch 5/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 42s 52ms/step - accuracy: 0.0095 - loss: 4.6055 - val_accuracy: 0.0100 - val_loss: 4.6052
```

```
# Evaluate AlexNet
alexnet_loss, alexnet_acc = alexnet.evaluate(test_images, test_labels)
print(f"AlexNet - Accuracy: {alexnet_acc:.4f}, Loss: {alexnet_loss:.4f}, Time: {alexnet_time:.2f} sec")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 17s 41ms/step - accuracy: 0.0109 - loss: 4.6052
AlexNet - Accuracy: 0.0100, Loss: 4.6052, Time: 1081.88 sec
```

```
# Evaluate VGG16
vgg16_loss, vgg16_acc = vgg16.evaluate(test_images, test_labels)
print(f"VGG16 - Accuracy: {vgg16_acc:.4f}, Loss: {vgg16_loss:.4f}, Time: {vgg16_time:.2f} sec")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.0106 - loss: 4.6052
VGG16 - Accuracy: 0.0100, Loss: 4.6052, Time: 251.78 sec
```

```
import pandas as pd

results = pd.DataFrame({
    "Model": ["AlexNet", "VGG16"],
    "Accuracy": [alexnet_acc, vgg16_acc],
    "Loss": [alexnet_loss, vgg16_loss],
```

```
        "Training Time (s)": [alexnet_time, vgg16_time]
    })

    print(results)

    # Plot results
    sns.barplot(x="Model", y="Accuracy", data=results)
    plt.title("Model Accuracy Comparison")
    plt.show()
```

```
         Model  Accuracy      Loss  Training Time (s)
    0  AlexNet      0.01  4.605178        1081.882729
    1    VGG16      0.01  4.605182         251.783153
```