# SAM

```
! pip install metaseg -q
from IPython.display import Image
Image('/content/Prostate.png')
!pip install matplotlib-venn
!apt-get -qq install -y libfluidsynth1
!apt-get -qq install -y libarchive-dev && pip install -U libarchive
import libarchive
!apt-get -qq install -y graphviz && pip install pydot
import pydot
!pip install cartopy
import cartopy
pip install metaseg
!pip install fal_serverless
!pip install metaseg
from metaseg import SegAutoMaskPredictor
autoseg_image = SegAutoMaskPredictor().image_predict(
    source='/content/Prostate.png',
    model_type="vit_l", # vit_l, vit_h, vit_b
    points_per_side=16,
    points_per_batch=64,
    min_area=0,
    output_path="output.png",
    show=False,
    save=True,
)
from IPython.display import Image
Image('output.png')
from IPython.display import Image
Image('/content/Prostate.png')
```

# Forward Diffusion

```
from scipy.stats import norm
import random
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-3, 3, num = 100) #Input x
sigma = 1 # Standard Deviation
mean = 0 # Mean
print(x)
constant = np.sqrt(2.0*np.pi*sigma**2)
pdf_normal_distribution = np.exp(-
((x - mean)**2/(2.0*sigma**2)))/constant
fig, ax = plt.subplots(figsize=(10, 5));
ax.plot(x, pdf_normal_distribution);
ax.set_ylim(0);
ax.set_title('Normal Distribution', size = 20);
ax.set_ylabel('Probability Density', size = 20)
import os
import scipy
import urllib
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from scipy.stats import norm, multivariate_normal
import plotly
import plotly.graph_objects as go
from plotly.subplots import make_subplots
def forward_diffusion_process(img_prev, beta, t):
    beta_t = beta[t].reshape(-1, 1, 1)
        #Calculate mean and variance
    mu = np.sqrt((1.0 - beta_t)) * img_prev
    sigma = np.sqrt(beta_t)
        #Obtain image at timestep t using equation
    img_t = mu + sigma * np.random.randn(*img_prev.shape)
    return img_t
#Input
img = Image.open("/content/Prostate.png")
IMG_SIZE = (128, 128)
img = img.resize(size=IMG_SIZE)
img_curr = np.asarray(img.copy(), dtype=np.float32) / 255.
#Parameters
timesteps = 100
beta_start = 0.0001
beta_end = 0.05
beta = np.linspace(beta_start, beta_end, num=timesteps, dtype=np.float
32)
print("Beta: ", beta)
processed_images = []
# Run the forward process to obtain img after t timesteps
for t in range(timesteps):
    img_curr = forward_diffusion_process(img_prev=img_curr, beta=beta,
 t=t)
    if t%20==0 or t==timesteps - 1:
        sample = (img_curr.clip(0, 1) * 255.0).astype(np.uint8)
        processed_images.append(sample)
#Plot and see samples at different timesteps
_, ax = plt.subplots(1, len(processed_images), figsize=(12, 5))

for i, sample in enumerate(processed_images):
    ax[i].imshow(sample)
    ax[i].set_title(f"Timestep: {i*20}")
    ax[i].axis("off")
    ax[i].grid(False)

plt.suptitle("forward_diffusion_process", y=0.75)
plt.show()
```

## Image generator

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.models import *
from tensorflow.keras.preprocessing import image
from tensorflow.keras.utils import load_img

import tensorflow
import tensorflow as tf
# from tf.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.preprocessing.image import ImageDataGenerator

# Total Generated number
total_number = 20

data_gen = ImageDataGenerator(rescale=1. / 255, shear_range=0.2,
                              zoom_range=0.2, horizontal_flip=True)
# !unzip '/content/Desktop.zip'
# Create image to tensor
img = load_img("/content/drive/MyDrive/G/generated_0_523..png",
grayscale=False)
arr = img_to_array(img)
tensor_image = arr.reshape((1, ) + arr.shape)

for i, _ in enumerate(data_gen.flow(x=tensor_image,
                                    batch_size=1,
                                    save_to_dir="/content",
                                    save_prefix="generated",
                                    save_format=".png")):
    if i > total_number:
        break
```

## Image enhancement with Keras epochs Increasing image resolution

```
import numpy as np

from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Activation, Reshape
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import img_to_array,
load_img
```

```
# Define the dimensions of the input image
input_height = 123
input_width = 217

# Define the number of training epochs
epochs = 1000

# Load and preprocess the input image
input_image = load_img('/content/ProstateBad.jpeg',
target_size=(input_height, input_width), color_mode='grayscale')
input_image = img_to_array(input_image) / 255.0  # Normalize pixel
values to the range [0, 1]
input_image = np.expand_dims(input_image, axis=0)

# Repeat the grayscale image across three channels
input_image = np.repeat(input_image, 3, axis=-1)

# Define the model architecture
model = Sequential()
model.add(Conv2D(64, (3, 3), padding='same',
input_shape=(input_height, input_width, 3)))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(3, (3, 3), padding='same'))
model.add(Activation('sigmoid'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

# Train the model
model.fit(input_image, input_image, epochs=epochs, verbose=1)

# Generate the enhanced image
enhanced_image = model.predict(input_image)

# Rescale the enhanced image back to the range [0, 255]
enhanced_image = enhanced_image.squeeze() * 255.0
enhanced_image = enhanced_image.astype(np.uint8)

# Save the enhanced image
enhanced_image = enhanced_image.reshape((input_height, input_width,
3))
keras.preprocessing.image.save_img('enhanced_image2.jpg',
enhanced_image)
```

## Increase quality of image using CV

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

# Load the color image
image = cv2.imread('/content/ProstateBad.jpeg')

# Convert the color image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Resize the grayscale image to the desired dimensions
# desired_width = 217
# desired_height = 123
gray_image = cv2.resize(gray_image, (desired_width, desired_height))

# Convert the grayscale image to 8-bit unsigned integer
gray_image = np.uint8(gray_image)

# Apply various image enhancement techniques
enhanced_image = gray_image.copy()

# Apply noise reduction techniques (e.g., Gaussian blur)
enhanced_image = cv2.GaussianBlur(enhanced_image, (3, 3), 0)

# Adjust the image contrast using histogram equalization
enhanced_image = cv2.equalizeHist(enhanced_image)

# Increase image sharpness using unsharp masking
enhanced_image = cv2.GaussianBlur(enhanced_image, (3, 3), 0)
enhanced_image = cv2.addWeighted(enhanced_image, 1.5, gray_image, -
0.5, 0)

# Apply dynamic range adjustment (e.g., contrast stretching)
min_intensity = np.min(enhanced_image)
max_intensity = np.max(enhanced_image)
enhanced_image = (enhanced_image - min_intensity) / (max_intensity -
min_intensity) * 255

# Reduce compression artifacts using JPEG deblocking
enhanced_image = cv2.fastNlMeansDenoising(gray_image, None, 10, 7, 21)

# Apply artistic interpretation or style transfer techniques
# (e.g., using pre-trained neural networks)

# Display the enhanced image
cv2_imshow(enhanced_image)
```

# Cancer Detection with EfficientNetB3

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.models import *
from tensorflow.keras.preprocessing import image

!unzip '/content/Project 1 DWI.zip'

train_datagen = image.ImageDataGenerator(
    rotation_range=15,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    width_shift_range=0.1,
    height_shift_range=0.1)
val_datagen= image.ImageDataGenerator(
    rotation_range=15,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    width_shift_range=0.1,
    height_shift_range=0.1)
test_datagen= image.ImageDataGenerator(
    rotation_range=15,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    width_shift_range=0.1,
    height_shift_range=0.1)

train_generator = train_datagen.flow_from_directory(
    '/content/train',
    target_size = (224,224),
    batch_size = 8,
    class_mode = 'categorical')
test_generator = test_datagen.flow_from_directory(
    '/content/valid',
    target_size = (224,224),
    batch_size = 8,
    shuffle=True,
    class_mode = 'categorical')
validation_generator = test_datagen.flow_from_directory(
    '/content/test',
    target_size = (224,224),
    batch_size = 8,
    shuffle=True,
    class_mode = 'categorical')

base_model = tf.keras.applications.EfficientNetB3(weights='imagenet',
input_shape=(224,224,3), include_top=False)

for layer in base_model.layers:
    layer.trainable=True
model = Sequential()
model.add(base_model)
model.add(GaussianNoise(0.25))
model.add(GlobalAveragePooling2D())
model.add(Dense(1024,activation='relu'))
model.add(BatchNormalization())
model.add(GaussianNoise(0.25))
model.add(Dropout(0.25))
model.add(Dense(2, activation='sigmoid'))
model.summary()

model.compile(loss='categorical_crossentropy', optimizer=tf.keras.opti
mizers.Adam(learning_rate=0.000001),metrics=['accuracy','AUC','Precisi
on','Recall'])

from keras.callbacks import ModelCheckpoint, EarlyStopping
es=EarlyStopping(monitor='val_loss',patience=3)
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=validation_generator
    )

model.evaluate(train_generator)

model.evaluate(validation_generator)

imag = tf.keras.utils.load_img('/content/Test Internet CancerDWI.jpg')
imag

from PIL import Image
img=Image.open("/content/Test Internet CancerDWI.jpg")
w,h=img.size    # w=Width and h=Height
print("Width =",w,end="\t")
print("Height =",h)
imag = imag.resize((224, 224))
from PIL import Image
w,h=imag.size    # w=Width and h=Height
print("Width =",w,end="\t")
print("Height =",h)

from tensorflow.keras.utils import load_img
from keras.preprocessing import image
img = tf.keras.utils.load_img('/content/Test Internet CancerDWI.jpg',t
arget_size=(224,224))
# imag = tf.keras.utils.load_img('/content/Test Internet CancerDWI.jpg
')
imaga = np.expand_dims(imag,axis=0)
ypred = model.predict(imaga)
print(ypred)
a=np.argmax(ypred,-1)
if a==0:
  op="Cancer"
elif a==1:
  op="Normal"
plt.imshow(img)
print("THE UPLOADED IMAGE IS SUSPECTED AS: "+str(op))

tf.keras.models.save_model(model,'mymodel.vahid')

!pip install streamlit
```