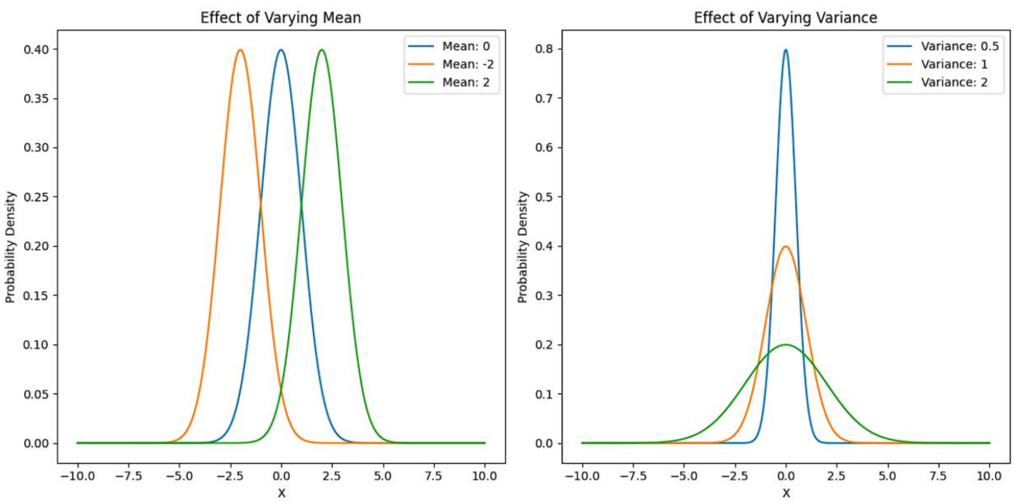
Lab-7

1. Write a Python program that computes the value of the Gaussian distribution at a given vector X. Hence, plot the effect of varying mean and variance to the normal distribution.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
def gaussian_distribution(X, mu, sigma):
    return (1 / (np.sqrt(2 * np.pi * sigma**2))) * np.exp(-((X - mu)**2) / (2 * sigma**2))
X = np.linspace(-10, 10, 1000)
# Example means and variances to plot
means = [0, -2, 2]
variances = [0.5, 1, 2]
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
for mu in means:
    Y = gaussian_distribution(X, mu, 1)  # Keeping variance constant at 1
    plt.plot(X, Y, label=f'Mean: {mu}')
plt.title('Effect of Varying Mean')
plt.xlabel('X')
plt.ylabel('Probability Density')
plt.legend()
plt.subplot(1, 2, 2)
for sigma in variances:
    Y = gaussian_distribution(X, 0, sigma)  # Keeping mean constant at 0
    plt.plot(X, Y, label=f'Variance: {sigma}')
plt.title('Effect of Varying Variance')
plt.xlabel('X')
plt.ylabel('Probability Density')
plt.legend()
plt.tight_layout()
plt.show()
```

Output:



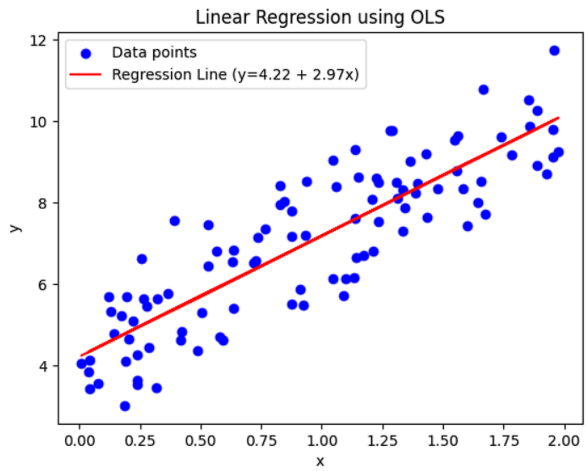
2. Write a python program to implement linear regression.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
x = 2 * np.random.rand(100, 1)
y = 4 + 3 * x + np.random.randn(100, 1)
```

```
def linear_regression(x, y):
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    num = np.sum((x - x_mean) * (y - y_mean))  # Numerator
    denom = np.sum((x - x_mean) ** 2)         # Denominator
    beta_1 = num / denom
    beta_0 = y_mean - beta_1 * x_mean
    return beta_0, beta_1
def predict(x, beta_0, beta_1):
    return beta_0 + beta_1 * x
beta_0, beta_1 = linear_regression(x, y)
y_pred = predict(x, beta_0, beta_1)
plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, y_pred, color='red', label=f'Regression Line (y={beta_0:.2f} + {beta_1:.2f}x)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression using OLS')
plt.legend()
plt.show()
print(f"Intercept (beta_0): {beta_0:.2f}")
print(f"Slope (beta_1): {beta_1:.2f}")
```

Output:




```
Intercept (beta_0): 4.22
Slope (beta_1): 2.97
```

3. Write a python program to implement gradient descent.

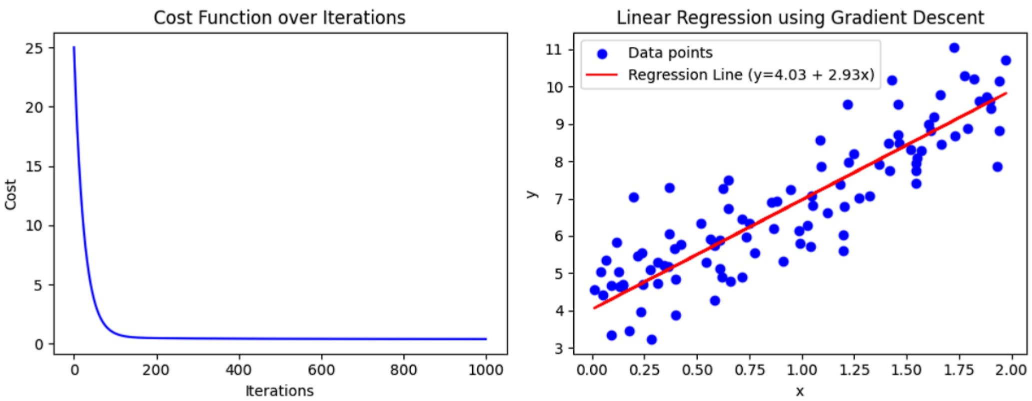
Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
x = 2 * np.random.rand(100, 1)
y = 4 + 3 * x + np.random.randn(100, 1)
def gradient_descent(x, y, learning_rate=0.01, n_iterations=1000):
    m = len(y)
    theta_0 = 0  # Initial intercept
    theta_1 = 0  # Initial slope
    cost_history = []  # Keep track of the cost function for each iteration
    for iteration in range(n_iterations):
        y_pred = theta_0 + theta_1 * x
        d_theta_0 = (1 / m) * np.sum(y_pred - y)
        d_theta_1 = (1 / m) * np.sum((y_pred - y) * x)
        theta_0 = theta_0 - learning_rate * d_theta_0
        theta_1 = theta_1 - learning_rate * d_theta_1
        cost = (1 / (2 * m)) * np.sum((y_pred - y) ** 2)
        cost_history.append(cost)
```

```
        if iteration % 100 == 0:
            print(f"Iteration {iteration}: Cost = {cost:.4f}")
    return theta_0, theta_1, cost_history
learning_rate = 0.01
n_iterations = 1000
theta_0, theta_1, cost_history = gradient_descent(x, y, learning_rate, n_iterations)
y_pred = theta_0 + theta_1 * x
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(range(n_iterations), cost_history, 'b-')
plt.title('Cost Function over Iterations')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.subplot(1, 2, 2)
plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, y_pred, color='red', label=f'Regression Line (y={theta_0:.2f} + {theta_1:.2f}x)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression using Gradient Descent')
plt.legend()
plt.tight_layout()
plt.show()
print(f"Intercept (theta_0): {theta_0:.2f}")
print(f"Slope (theta_1): {theta_1:.2f}")
```

<u>Output:</u>

```
Iteration 0: Cost = 25.0042
Iteration 100: Cost = 0.8861
Iteration 200: Cost = 0.4848
Iteration 300: Cost = 0.4572
Iteration 400: Cost = 0.4416
Iteration 500: Cost = 0.4306
Iteration 600: Cost = 0.4227
Iteration 700: Cost = 0.4171
Iteration 800: Cost = 0.4131
Iteration 900: Cost = 0.4103
```




```
Intercept (theta_0): 4.03
Slope (theta_1): 2.93
```

4. Write a python program to classify different flower images using MLP.

<u>Source Code:</u>

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import io
import ipywidgets as widgets
from IPython.display import display
```

```
batch_size = 32
img_height = 180
img_width = 180
data_dir = r"C:\Users\anupa\OneDrive\Desktop\Assignment_Sem-3\Python\flowers"  # Replace with your dataset path
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
class_names = train_ds.class_names
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
model = models.Sequential([
    layers.Flatten(input_shape=(img_height, img_width, 3)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(5, activation='softmax')  # 5 flower classes
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

def preprocess_image(image):
    image = image.resize((img_height, img_width))  # Resize the image
    image_array = np.array(image) / 255.0  # Normalize the image
    image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension
    return image_array
def predict_flower(image):
    processed_image = preprocess_image(image)
    prediction = model.predict(processed_image)
    predicted_class = np.argmax(prediction)
    return class_names[predicted_class]
def on_upload_change(change):
    uploaded_file = change['new']
    if uploaded_file:
        img = Image.open(io.BytesIO(uploaded_file[0]['content']))
        display(img)
        flower_class = predict_flower(img)
        print(f"Predicted flower class: {flower_class}")
uploader = widgets.FileUpload(accept='image/*', multiple=False)
uploader.observe(on_upload_change, names='value')
display(uploader)
```

Output:

```
Found 4317 files belonging to 5 classes.
Using 3454 files for training.
```

Upload (1)

```
1/1 [==============================] - 0s 162ms/step
Predicted flower class: sunflower
```

5. Write a python program to classify different flower images using the SVM classifier.

Source Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from PIL import Image
import io
import ipywidgets as widgets
from IPython.display import display, clear_output
import os
def load_and_preprocess_data(data_dir, img_height, img_width):
    images = []
    labels = []
    class_names = sorted(os.listdir(data_dir))
    for class_index, class_name in enumerate(class_names):
        class_dir = os.path.join(data_dir, class_name)
        for img_name in os.listdir(class_dir):
            img_path = os.path.join(class_dir, img_name)
            img = image.load_img(img_path, target_size=(img_height, img_width))
            img_array = image.img_to_array(img)
            img_array = preprocess_input(img_array)
            images.append(img_array)
            labels.append(class_index)
        return np.array(images), np.array(labels), class_names
def extract_features(model, images):
    features = model.predict(images)
    return features
data_dir = r"C:\Users\anupa\OneDrive\Desktop\Assignment_Sem-3\Python\flowers"  # Replace with your dataset path
img_height = 180
img_width = 180
images, labels, class_names = load_and_preprocess_data(data_dir, img_height, img_width)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
base_model.trainable = False
features = extract_features(base_model, images)
features_flat = features.reshape(features.shape[0], -1)  # Flatten the features
X_train, X_test, y_train, y_test = train_test_split(features_flat, labels, test_size=0.2, random_state=42)
svm = SVC(kernel='linear')  # Using a linear kernel for SVM
svm.fit(X_train, y_train)
def preprocess_image(img):
    img = img.resize((img_height, img_width))  # Resize the image
    img_array = np.array(img)
    img_array = preprocess_input(img_array)  # Normalize the image
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array
def predict_flower(img):
    processed_image = preprocess_image(img)
    features = extract_features(base_model, processed_image)
    features_flat = features.reshape(features.shape[0], -1)  # Flatten the features
    prediction = svm.predict(features_flat)
    return class_names[prediction[0]]
def on_upload_change(change):
    uploaded_file = change['new']
    if uploaded_file:
        img = Image.open(io.BytesIO(uploaded_file[0]['content']))
        clear_output(wait=True)
```

```
        display(img)
        flower_class = predict_flower(img)
        print(f"Predicted flower class: {flower_class}")
uploader = widgets.FileUpload(accept='image/*', multiple=False)
uploader.observe(on_upload_change, names='value')
display(uploader)
```

Output:

```
1/1 [==============================] - 0s 207ms/step
Predicted flower class: dandelion
```

6. Write a python program to classify different flower images using CNN.

Source Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing import image
from PIL import Image
import numpy as np
import io
import ipywidgets as widgets
from IPython.display import display, clear_output
import os
def create_model(num_classes):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(180, 180, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
def load_and_preprocess_data(data_dir, img_height, img_width):
    train_ds = tf.keras.preprocessing.image_dataset_from_directory(
        data_dir,
        validation_split=0.2,
        subset="training",
        seed=123,
        image_size=(img_height, img_width),
        batch_size=32
    )
    val_ds = tf.keras.preprocessing.image_dataset_from_directory(
        data_dir,
```

```
        validation_split=0.2,
        subset="validation",
        seed=123,
        image_size=(img_height, img_width),
        batch_size=32
    )
    normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
    train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
    val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
    try:
        class_names = train_ds.class_names
    except AttributeError:
        class_names = [d for d in os.listdir(data_dir) if os.path.isdir(os.path.join(data_dir, d))]
    return train_ds, val_ds, class_names
data_dir = r"C:\Users\anupa\OneDrive\Desktop\Assignment_Sem-3\Python\flowers"  # Replace with your dataset path
img_height = 180
img_width = 180
train_ds, val_ds, class_names = load_and_preprocess_data(data_dir, img_height, img_width)
print(f"Class names: {class_names}")
num_classes = len(class_names)
model = create_model(num_classes)

model.summary()
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=3  # Reduce the number of epochs for demonstration
)
model.save('flower_classifier.h5')
model = tf.keras.models.load_model('flower_classifier.h5')
def preprocess_image(img):
    img = img.resize((img_height, img_width))  # Resize the image
    img_array = np.array(img) / 255.0  # Normalize the image
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array
def predict_flower(img):
    processed_image = preprocess_image(img)
    prediction = model.predict(processed_image)
    predicted_class = np.argmax(prediction)
    return class_names[predicted_class]
def on_upload_change(change):
    uploaded_file = change['new']
    if uploaded_file:
        img = Image.open(io.BytesIO(uploaded_file[0]['content']))
        clear_output(wait=True)
        display(img)
        flower_class = predict_flower(img)
        print(f"Predicted flower class: {flower_class}")
uploader = widgets.FileUpload(accept='image/*', multiple=False)
uploader.observe(on_upload_change, names='value')
display(uploader)
```

Output:

```
1/1 [==============================] - 0s 191ms/step
Predicted flower class: rose
```

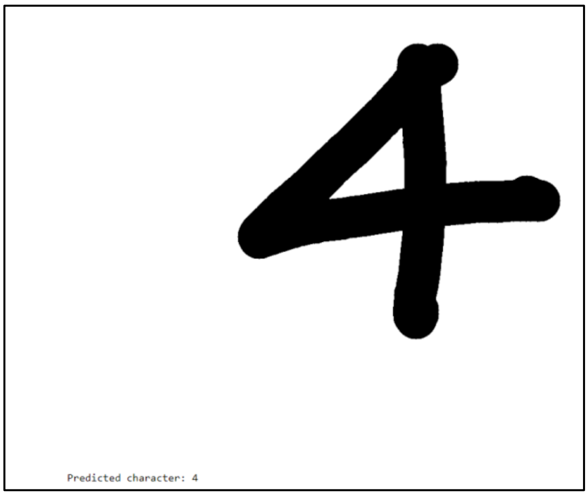
7. Write a python program to classify different handwritten character images using the SVM classifier.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from sklearn import datasets, svm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from PIL import Image
from IPython.display import display, clear_output
import io
digits = datasets.load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
clf = svm.SVC(kernel='linear', probability=True)
clf.fit(X_train, y_train)
def preprocess_image(img):
   img = img.convert('L').resize((8, 8))
   img_array = np.array(img)
   img_array = (img_array / 255.0) * 16
   img_array = 16 - img_array
   img_array = img_array.flatten().reshape(1, -1)
   img_array = scaler.transform(img_array)
   return img_array
def predict_character(img):
   img_array = preprocess_image(img)
   prediction = clf.predict(img_array)
   return prediction[0]
def on_upload_change(change):
   uploaded_file = change['new']
   if uploaded_file:
      img = Image.open(io.BytesIO(uploaded_file[0]['content']))
      clear_output(wait=True)
      display(img)
      predicted_character = predict_character(img)
            print(f"Predicted character: {predicted_character}")
uploader = widgets.FileUpload(accept='image/*', multiple=False)
uploader.observe(on_upload_change, names='value')
display(uploader)
```

Output:



8. Write a python program to classify different face images using CNN.

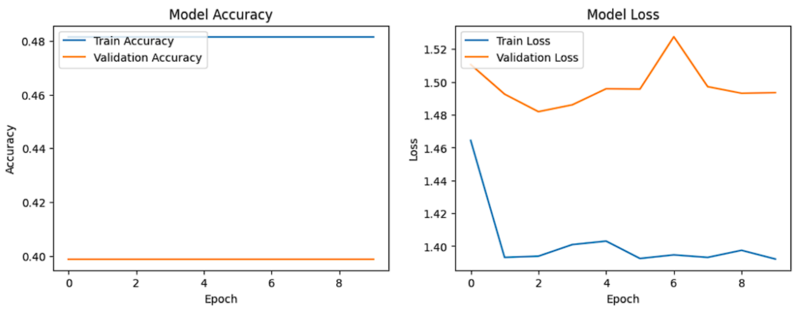
Source Code:

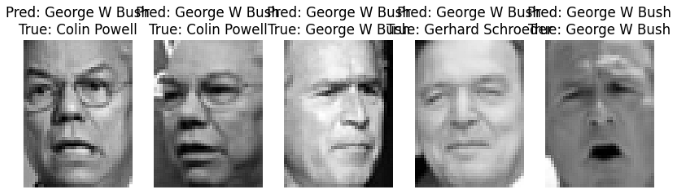
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from sklearn.datasets import fetch_lfw_people
lfw_people = fetch_lfw_people(min_faces_per_person=100, resize=0.4)
X = lfw_people.images  # Face images (grayscale)
y = lfw_people.target  # Labels (people)
target_names = lfw_people.target_names  # Names of people
X = X / 255.0
X = X[..., np.newaxis]
label_binarizer = LabelBinarizer()
y = label_binarizer.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(len(target_names), activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.2f}")
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)
print(classification_report(y_true_classes, y_pred_classes, target_names=target_names))
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='upper left')
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper left')
plt.show()
n_samples = 5
plt.figure(figsize=(10, 5))
for i in range(n_samples):
    plt.subplot(1, n_samples, i + 1)
    plt.imshow(X_test[i].reshape(X_test.shape[1], X_test.shape[2]), cmap='gray')
    plt.title(f"Pred: {target_names[y_pred_classes[i]]}\nTrue: {target_names[y_true_classes[i]]}")
    plt.axis('off')
plt.show()
```

Output:





10. Write a python program to classify breast cancer from histopathological images using VGG-16 and DenseNet-201 CNN architectures

Source Code:

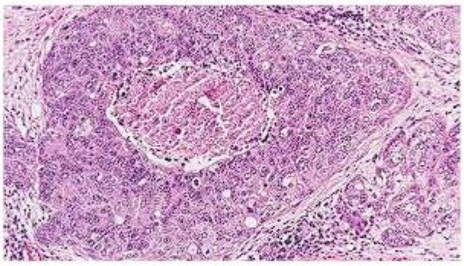
```
import tensorflow as tf
from tensorflow.keras.applications import VGG16, DenseNet201
from tensorflow.keras.applications.vgg16 import preprocess_input as vgg16_preprocess
from tensorflow.keras.applications.densenet import preprocess_input as densenet_preprocess
from tensorflow.keras.preprocessing import image
import numpy as np
from PIL import Image
import io
import ipywidgets as widgets
from IPython.display import display, clear_output
vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
densenet_model = DenseNet201(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
def build_custom_model(base_model, num_classes):
    model = tf.keras.Sequential([
        base_model,
        tf.keras.layers.GlobalAveragePooling2D(),
```

```
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
num_classes = 2
class_names = ['Benign', 'Malignant']
vgg16_custom_model = build_custom_model(vgg16_model, num_classes)
densenet_custom_model = build_custom_model(densenet_model, num_classes)
def preprocess_image(img, model_type):
    img = img.resize((224, 224))  # Resize image to 224x224 pixels
    img_array = image.img_to_array(img)  # Convert to numpy array
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    if model_type == 'vgg16':
        img_array = vgg16_preprocess(img_array)
    elif model_type == 'densenet':
        img_array = densenet_preprocess(img_array)

    return img_array
def predict_vgg16(img):
    processed_image = preprocess_image(img, model_type='vgg16')
    prediction = vgg16_custom_model.predict(processed_image)
    predicted_class = np.argmax(prediction)
    return class_names[predicted_class]

def predict_densenet(img):
    processed_image = preprocess_image(img, model_type='densenet')
    prediction = densenet_custom_model.predict(processed_image)
    predicted_class = np.argmax(prediction)
    return class_names[predicted_class]
def on_upload_change(change):
    uploaded_file = change['new']
    if uploaded_file:
        img = Image.open(io.BytesIO(uploaded_file[0]['content']))
        clear_output(wait=True)
        display(img)
        vgg16_prediction = predict_vgg16(img)
        densenet_prediction = predict_densenet(img)
        print(f"VGG-16 Prediction: {vgg16_prediction}")
        print(f"DenseNet-201 Prediction: {densenet_prediction}")
uploader = widgets.FileUpload(accept='image/*', multiple=False)
uploader.observe(on_upload_change, names='value')
display(uploader)
```

Output:



```
1/1 [==============================] - 1s 522ms/step
1/1 [==============================] - 6s 6s/step
VGG-16 Prediction: Benign
DenseNet-201 Prediction: Malignant
```