Lab-7

# 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)

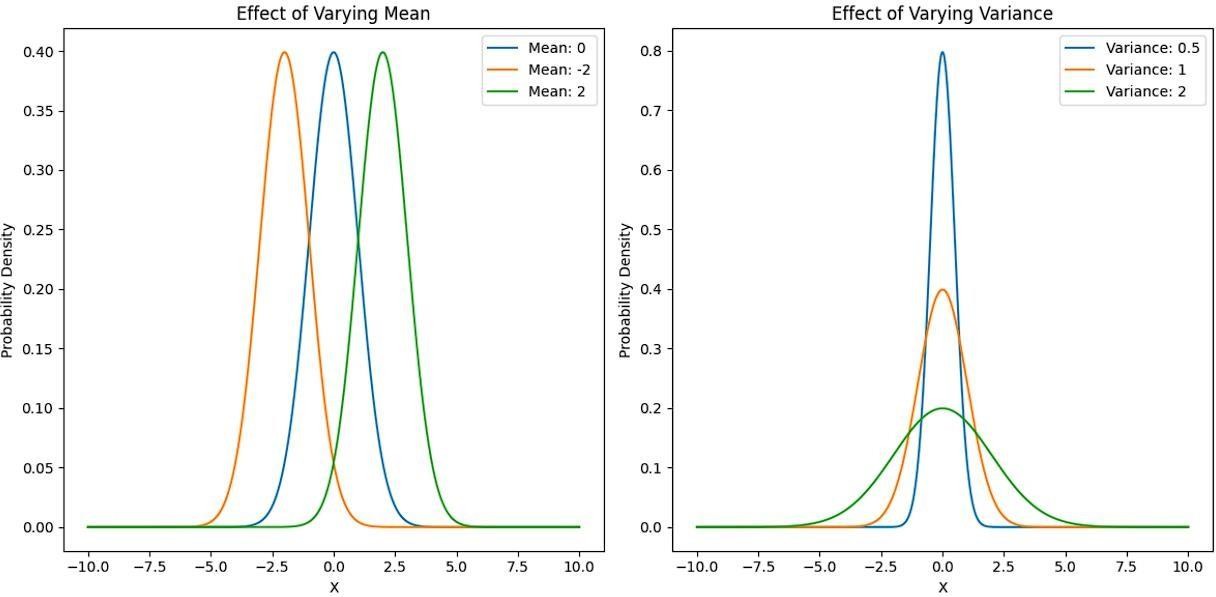
for sigma in variances:

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:



1. **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:

1. **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))

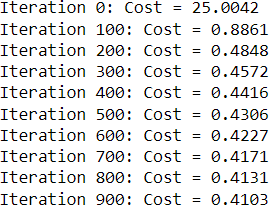
plt.subplot(1, 2, 1) plt.plot(range(n\_iterations), cost\_history, 'b-') plt.title('Cost Function over Iterations') plt.xlabel('Iterations')

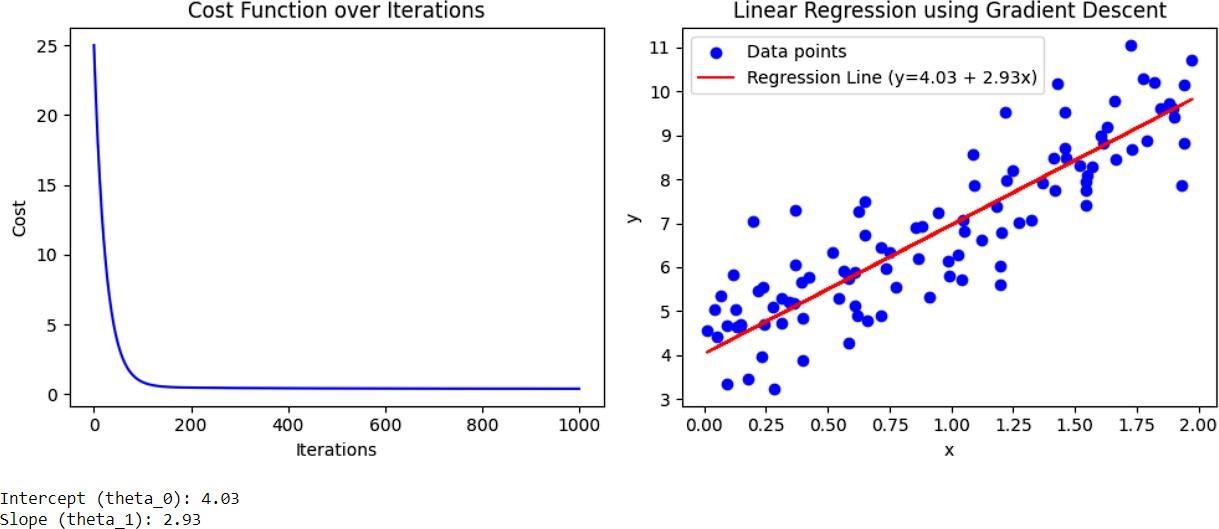
plt.ylabel('Cost') plt.subplot(1, 2, 2) plt.scatter(x, y, color='blue', label='Data points')

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}") **Output:**





# Write a python program to classify different flower images using MLP. Source Code:

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 =

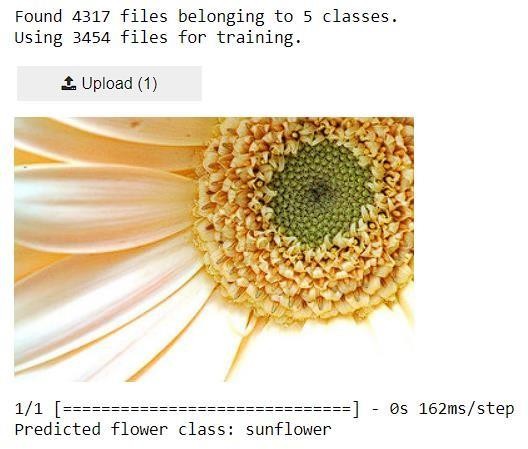
np.argmax(prediction) return class\_names[predicted\_class] def on\_upload\_change(change):

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:



1. **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. **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)



# 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:



1. **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:

