Assignment 3

Task 1

```
# Import necessary libraries

import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

# -----------------------
# Task 1 - Data Loading and Preprocessing
# -----------------------
# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features: sepal length, sepal width, petal length, petal
width
y = iris.target  # Target: species (0, 1, 2)

# Split into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the input features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# One-hot encode the target labels
y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)

# -----------------------
# Task 2 - Neural Network Construction
# -----------------------
# Build the neural network
model = Sequential()
```

```
model.add(Dense(8, input_shape=(4,), activation='relu'))  # Hidden layer
with 8 neurons
model.add(Dense(3, activation='softmax'))  # Output layer for 3 classes

# -----------------------
# Task 3 - Model Compilation and Training
# -----------------------
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train_encoded, epochs=100, batch_size=5, verbose=0)

# -----------------------
# Task 4 - Model Evaluation
# -----------------------
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test_encoded, verbose=0)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
test_samples = [
    [5.1, 3.5, 1.4, 0.2],
    [6.0, 2.2, 4.0, 1.0],
    [6.9, 3.1, 5.4, 2.1]
]
```

```
# ---- Predict custom test samples ----
# Standardized test samples using the same scaler used during training
test_samples_scaled = scaler.transform(test_samples)

# Predict using the trained model
predictions = model.predict(test_samples_scaled)

# Decode predictions to class labels
predicted_classes = np.argmax(predictions, axis=1)

# Map numerical labels to flower names
```

```
species_names = iris.target_names
predicted_species = [species_names[i] for i in predicted_classes]

# Display the results
for i, sample in enumerate(test_samples):
    print(f"Input: {sample} -> Predicted Species: {predicted_species[i]}")
```

```
Input: [5.1, 3.5, 1.4, 0.2] -> Predicted Species: setosa
Input: [6.0, 2.2, 4.0, 1.0] -> Predicted Species: versicolor
Input: [6.9, 3.1, 5.4, 2.1] -> Predicted Species: virginica
```

INSIGHTS OF TASK 1

Learn how to train a model of neural networks using tensorflow library
Learn how to split the data into train data and test data using train_test_split method of tensorflow
Learn how to fit model on train data and find the accuracy of the model on test data
Learn how neural network works and how to train them, how to give weights to the initial layer of the neural networks using gradient descent, importance of the non linear activation function and how to find the weights of the further layers of the model and then how to do backtracking to minimize the cost function to increase the model accuracy

Task 2

```
!pip install annoy
from google.colab import drive
drive.mount('/content/drive')
dataset_path = '/content/drive/MyDrive/kagglecatsanddogs_5340/PetImages'

from google.colab import files
uploaded = files.upload()




import zipfile
import os
```

```python
# Unzip the file
with zipfile.ZipFile("PetImages.zip", 'r') as zip_ref:
    zip_ref.extractall("PetImages")

# Verify extraction
print("Files extracted to:", os.listdir("PetImages"))
import os
import torch
from torchvision import models, transforms
from PIL import Image
from annoy import AnnoyIndex
import numpy as np
import matplotlib.pyplot as plt

# Use ResNet18 pre-trained on ImageNet
model = models.resnet18(pretrained=True)
model = torch.nn.Sequential(*list(model.children())[:-1])  # Remove final
classification layer
model.eval()  # Set model to evaluation mode

# Preprocessing for input images
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Image paths
image_paths = []
for folder in ['cat1', 'dog1']:
    folder_path = f'/content/PetImages/PetImages/{folder}'
    for fname in os.listdir(folder_path):
        if fname.lower().endswith(('.jpg', '.png', '.jpeg')):
            image_paths.append(os.path.join(folder_path, fname))

# Extract features and save to Annoy index
feature_dim = 512  # ResNet18 outputs 512-dim feature vectors
index = AnnoyIndex(feature_dim, 'angular')
features = []

for i, path in enumerate(image_paths):
```

```python
    try:
        img = Image.open(path).convert('RGB')
        img_tensor = transform(img).unsqueeze(0)
        with torch.no_grad():
            feature = model(img_tensor).squeeze().numpy()
        index.add_item(i, feature)
        features.append(feature)
    except Exception as e:
        print(f"Skipped {path} due to error: {e}")

index.build(10)  # Number of trees
index.save('image_features.ann')

# Function to show image
def show_image(image_path):
    img = Image.open(image_path)
    plt.imshow(img)
    plt.axis('off')
    plt.show()

# Search example
query_img_path = image_paths[0]
query_img = Image.open(query_img_path).convert('RGB')
query_tensor = transform(query_img).unsqueeze(0)
with torch.no_grad():
    query_feat = model(query_tensor).squeeze().numpy()

# Find top 5 similar images
similar_idxs = index.get_nns_by_vector(query_feat, 10)

# Display results
print("Query Image:")
show_image(query_img_path)

print("Similar Images:")
for idx in similar_idxs:
    print(image_paths[idx])
    show_image(image_paths[idx])
```

Query Image:

Similar Images:


Overview of image similarity search

Image Similarity Search Using Feature Detection

We have developed an image similarity search system that uses **feature detection** with pre-trained deep learning models and **Spotify's Annoy library**. This tool helps find images that look similar to a given query image from a dataset.

How Feature Detection Works

Feature detection is the process of identifying important patterns or characteristics in an image. In our project, we use a **pre-trained convolutional neural network (CNN)** called **ResNet18** to extract feature vectors.
These vectors are like unique digital signatures that describe the content of the image—such as its shapes, edges, and textures.

Pipeline of the Project

1. **Dataset Loading:**
   Images are first resized and preprocessed to a standard format.

2. **Feature Extraction:**
   Each image is passed through the CNN, which gives a 512-dimensional feature vector.

3. **Indexing:**
   These vectors are stored in a special structure using the **Annoy** library, which helps in fast searching.

4. **Query Search:**
   When a new image is given, its feature vector is compared with the stored ones to find the most similar images.

Why Use Pre-trained Models?

Pre-trained models like **ResNet18** have already learned useful patterns from large datasets like **ImageNet**.
This means we don't have to train them from scratch, and they work well for tasks like feature extraction.
They save time and give better results due to their generalization ability.

Applications of This System

- Visual product search in **e-commerce** platforms
- Detecting **duplicate images**
- **Content-based image retrieval** in media libraries or photo apps