Program 1

Implement the following for the CIFAR-10 (Canadian Institute for Advanced Research) dataset:

a. using KNN,

b. using 3-layer neural network

Program: # Using KNN

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
# Load the CIFAR-10 dataset (You may need to download it separately)
# Load and preprocess the data
cifar10 = datasets.fetch_openml(name="CIFAR_10_small")
X = cifar10.data.astype('float32')
y = cifar10.target.astype('int')
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Create and train a K-Nearest Neighbors (KNN) classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
# Evaluate the classifier
accuracy = knn.score(X_test, y_test)
print(f"KNN Accuracy: {accuracy:.2f}")
```

Output: KNN Accuracy: 0.30

Code: # 3-layer neural network

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, datasets
from torch.utils.data import DataLoader
# Set random seed for reproducibility
torch.manual_seed(42)
# Define a simple 3-layer neural network for Softmax classification
class SimpleNet(nn.Module):
    def __init__(self, input_size, num_classes):
        super(SimpleNet, self).__init__()
        self.fc1 = nn.Linear(input_size, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, num_classes)
    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the input
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
# Load and preprocess the CIFAR-10 dataset using torchvision
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
cifar10_train = datasets.CIFAR10(root='./data', train=True, transform=transform, download=True)
cifar10_test = datasets.CIFAR10(root='./data', train=False, transform=transform, download=True)
# Set data loaders
train_loader = DataLoader(cifar10_train, batch_size=64, shuffle=True)
test_loader = DataLoader(cifar10_test, batch_size=64)
# Initialize the model and optimizer
model = SimpleNet(input_size=32*32*3, num_classes=10)
```

```python
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
criterion = nn.CrossEntropyLoss()
# Training loop
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
# Evaluate the model on the test set
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
accuracy = 100 * correct / total
print(f"3-Layer Neural Network Accuracy: {accuracy:.2f}%")
```

Output:

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:03<00:00, 42993818.54it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
3-Layer Neural Network Accuracy: 53.74%
```

Applications:

Text mining, Agriculture, Finance, Medical and Facial recognition

Program 2

Train a Deep learning model to classify a given image using the pre-trained model.

Code:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms
import numpy as np
from PIL import Image
# Define data transformations for preprocessing
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
# Load a pre-trained ResNet-50 model
model = models.resnet50(pretrained=True)
num_features = model.fc.in_features
# Replace the final fully connected layer for binary classification
model.fc = nn.Linear(num_features, 2)
# Define a loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
# Load an image to classify
image_path = '/…../Path to Cat1.jpg'
image = Image.open(image_path)
image = data_transforms['train'](image)
```

```python
image = image.unsqueeze(0)  # Add batch dimension
# Load the pre-trained model's weights
#model.load_state_dict(torch.load('/content/drive/MyDrive/20CSEL76-LAB Programs/resnet50-0676ba61.pth'))
# Save the model state dictionary to a file
torch.save(model.state_dict(), 'resnet50_pretrained_weights.pth')
# Set the model to evaluation mode
model.eval()
# Make predictions
with torch.no_grad():
    outputs = model(image)
    _, predicted = torch.max(outputs, 1)
# Map class indices to class labels
class_labels = ['cat', 'dog']
predicted_class = class_labels[predicted.item()]
print(f'The image is classified as: {predicted_class}')
```

Output: The image is classified as: cat

Applications:

Translation, chatbots and other natural language processing applications

Program 3

Train a CNN using Tensorflow and Keras

Code:

```
import tensorflow as tf
from tensorflow import keras
# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
# Preprocess the data
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
# Define a simple CNN model
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation="relu", input_shape=(32, 32, 3)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
# Compile the model
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
Epoch 1/10
625/625 [==============================] - 42s 63ms/step - loss: 1.5269 - accuracy: 0.4552 - val_loss: 1.3100 - val_accuracy: 0.5384
Epoch 2/10
625/625 [==============================] - 36s 58ms/step - loss: 1.2210 - accuracy: 0.5699 - val_loss: 1.2296 - val_accuracy: 0.5697
Epoch 3/10
625/625 [==============================] - 37s 59ms/step - loss: 1.1088 - accuracy: 0.6115 - val_loss: 1.1191 - val_accuracy: 0.6138
Epoch 4/10
625/625 [==============================] - 35s 57ms/step - loss: 1.0218 - accuracy: 0.6432 - val_loss: 1.1003 - val_accuracy: 0.6196
Epoch 5/10
625/625 [==============================] - 36s 57ms/step - loss: 0.9418 - accuracy: 0.6746 - val_loss: 1.0600 - val_accuracy: 0.6335
Epoch 6/10
625/625 [==============================] - 39s 62ms/step - loss: 0.8775 - accuracy: 0.6961 - val_loss: 1.0440 - val_accuracy: 0.6437
Epoch 7/10
625/625 [==============================] - 39s 62ms/step - loss: 0.8259 - accuracy: 0.7135 - val_loss: 1.0736 - val_accuracy: 0.6319
Epoch 8/10
625/625 [==============================] - 39s 62ms/step - loss: 0.7702 - accuracy: 0.7320 - val_loss: 1.0551 - val_accuracy: 0.6459
Epoch 9/10
625/625 [==============================] - 36s 58ms/step - loss: 0.7217 - accuracy: 0.7499 - val_loss: 1.0430 - val_accuracy: 0.6486
Epoch 10/10
625/625 [==============================] - 35s 56ms/step - loss: 0.6729 - accuracy: 0.7691 - val_loss: 1.0607 - val_accuracy: 0.6454
313/313 [==============================] - 4s 12ms/step - loss: 1.0715 - accuracy: 0.6375
Test accuracy: 63.75%
```

Applications:

Image Classification, Object Recognition, facial recognition, medical image analysis and self-driving cars.

Program 4

Improve the Deep learning model (ex. CNN model of Question 3) by tuning hyperparameters.

Code:

```
import tensorflow as tf
from tensorflow import keras
# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
# Preprocess the data
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
# Data Augmentation
datagen = keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(x_train)
# Define a deeper CNN model
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation="relu", input_shape=(32, 32, 3)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(64, (3, 3), activation="relu"),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(128, (3, 3), activation="relu"),
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
```

```
])
# Compile the model
model.compile(optimizer=keras.optimizers.Adam(lr=0.001),
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
# Train the model with data augmentation
model.fit(datagen.flow(x_train, y_train, batch_size=64), epochs=20, validation_data=(x_test, y_test))
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

Output:

```
Epoch 12/20
782/782 [==============================] - 30s 39ms/step - loss: 0.8465 - accuracy: 0.1000 - val_loss: 0.9558 - val_accuracy: 0.1101
Epoch 13/20
782/782 [==============================] - 29s 38ms/step - loss: 0.8291 - accuracy: 0.1001 - val_loss: 0.8560 - val_accuracy: 0.0989
Epoch 14/20
782/782 [==============================] - 29s 37ms/step - loss: 0.8048 - accuracy: 0.1008 - val_loss: 0.8534 - val_accuracy: 0.1231
Epoch 15/20
782/782 [==============================] - 30s 38ms/step - loss: 0.7884 - accuracy: 0.1002 - val_loss: 0.8889 - val_accuracy: 0.0705
Epoch 16/20
782/782 [==============================] - 29s 38ms/step - loss: 0.7736 - accuracy: 0.1004 - val_loss: 0.8086 - val_accuracy: 0.1083
Epoch 17/20
782/782 [==============================] - 30s 38ms/step - loss: 0.7621 - accuracy: 0.1012 - val_loss: 0.7450 - val_accuracy: 0.1027
Epoch 18/20
782/782 [==============================] - 29s 37ms/step - loss: 0.7445 - accuracy: 0.1008 - val_loss: 0.8912 - val_accuracy: 0.1573
Epoch 19/20
782/782 [==============================] - 29s 37ms/step - loss: 0.7377 - accuracy: 0.0999 - val_loss: 0.7452 - val_accuracy: 0.1124
Epoch 20/20
782/782 [==============================] - 30s 38ms/step - loss: 0.7220 - accuracy: 0.1017 - val_loss: 0.8895 - val_accuracy: 0.0903
313/313 [==============================] - 1s 3ms/step - loss: 0.8895 - accuracy: 0.0903
Test accuracy: 9.03%
```

Applications:

Increasing model efficiency, robustness

Program 5

Implement an Object detection using a Convolution Neural Network variant

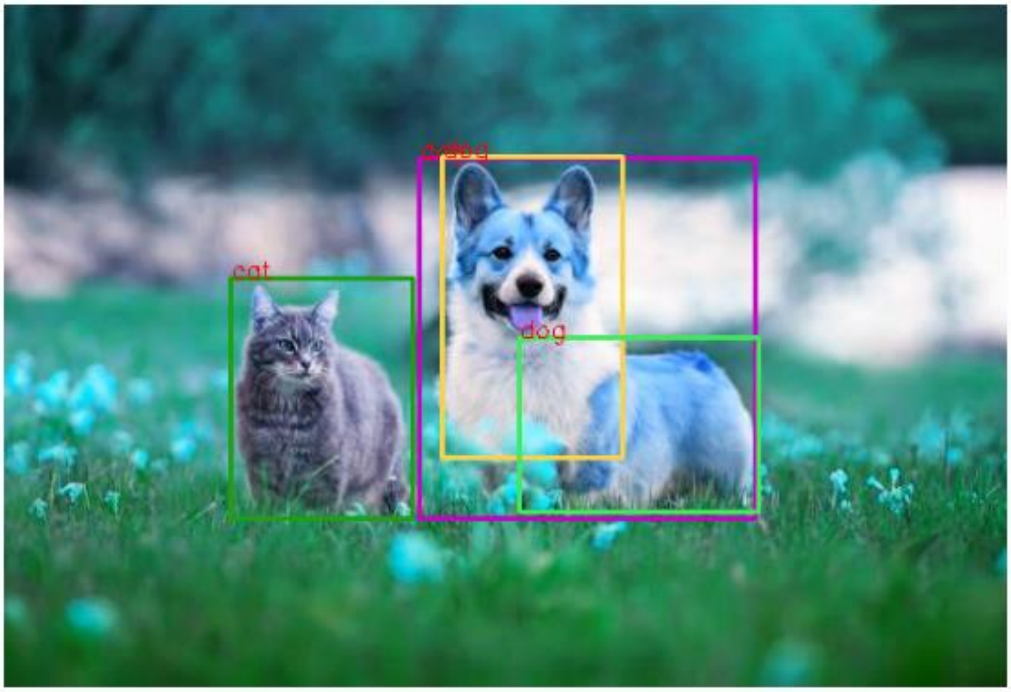
Code:

```
from PIL import Image
import cv2
import matplotlib.pyplot as plt
import numpy as np
import torch
import torchvision.transforms as T
import torchvision
from google.colab import drive
drive.mount('/content/drive')
# Download the pre-trained model from Pytorch repository
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
print("type(model): \t", type(model), "\n")
print("model: \n", model, "\n")
# Set the model to evaluation mode; this step is IMPORTANT
model.eval()
# Getting the list of all categories used to train the Faster R-CNN network
# Predictions from this list are supposed to be returned
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N/A',
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket',
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl',
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book',
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]
print("Faster R-CNN network train on ", len(COCO_INSTANCE_CATEGORY_NAMES), " object
categories")
THRESHOLD = 0.8
test_image_path ='/content/drive/MyDrive/VAP_GENAI_upendra/Object Detection
Datasets/dog_cat/val/images/cat_dog.jpg'
test_image = Image.open(test_image_path)
print("type(test_image): \t", test_image, "\n")
# transforming the test image to a Pytorch tensor
transform = T.Compose([T.ToTensor()])
test_image_tensor = transform(test_image)
print("type(test_image_tensor): \t", type(test_image_tensor), "\n")
print("test_image_tensor.size(): \t", test_image_tensor.size(), "\n")
```

```
print("test_image_tensor: \n", test_image_tensor, "\n")
# Make predictions on the test image using pre-trained model
predictions = model([test_image_tensor])
print("type(predictions): \t", type(predictions), "\n")
print("len(predictions): \t", len(predictions), "\n")
print("predictions: \n", predictions, "\n")
# Getting the actual predictions
prediction = predictions[0]
print("type(prediction): \t", type(prediction), "\n")
for key in prediction.keys():
    print(key, "\t", prediction[key])
# Extract the individual prediction components
bounding_boxes = prediction['boxes']
print("bounding_boxes: \n", bounding_boxes, "\n")
# Detach the bounding boxes from the computation graph
bounding_boxes = bounding_boxes.detach()
print("bounding_boxes: \n", bounding_boxes, "\n")
print("Number of bounding boxes (objects) detected: \t", bounding_boxes.size()[0], "\n")
# Extract the individual prediction components
class_labels = prediction['labels']
print("class_labels: \t\t\t\t\t", class_labels, "\n")
print("Number of bounding boxes (objects) detected: \t", class_labels.size()[0], "\n")
# Now  visualizing the detection results
# Swapping the axes to convret the dimension in the format (H, W, Channels)
test_image = cv2.imread(test_image_path)
test_image = cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB)
print("Unannotated test image")
plt.imshow(test_image)
plt.show()
from google.colab.patches import cv2_imshow
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 0.5
font_color = (0, 0, 255)  # White text color
font_thickness = 1
for box_index in range(bounding_boxes.size()[0]):
    print("Showing box: ", box_index+1, "\n")
    x1, y1, x2, y2 = bounding_boxes[box_index]
    random_color = list(np.random.choice(range(256), size=3))
    x1 = int(x1)
    x2 = int(x2)
    y1 = int(y1)
    y2 = int(y2)
    cv2.rectangle(test_image, (x1, y1), (x2, y2), (int(random_color[0]), int(random_color[1]),
int(random_color[2])), 2)
    cv2.putText(test_image,
str(COCO_INSTANCE_CATEGORY_NAMES[int(class_labels[box_index])]), (x1, y1), font,
font_scale, font_color, font_thickness)
```

```
cv2_imshow(test_image)
print("\n\n")
```

Output:



Applications:

Object detection and Region Proposal Network (RPN)

Program 6

Perform Sentiment Analysis using RNN.

Code:

```
from keras.datasets import imdb
from keras import Sequential
from keras.layers import Embedding, Dense, Dropout
from keras.layers import Activation, SimpleRNN
vocabulary_size = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words = vocabulary_size)
print('Loaded dataset with {} training samples, {} test samples'.format(len(X_train), len(X_test)))
print('---review---')
print(X_train[6])
print('---label---')
print(y_train[6])
word2id = imdb.get_word_index()
id2word = {i: word for word, i in word2id.items()}
print('---review with words---')
print([id2word.get(i, ' ') for i in X_train[6]])
print('---label---')
print(y_train[6])
print('Maximum review length: {}'.format(
len(max((X_train + X_test), key=len))))
print('Minimum review length: {}'.format(
len(min((X_test + X_test), key=len))))
from keras.preprocessing import sequence
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
embedding_size=32
model=Sequential()
model.add(Embedding(vocabulary_size, embedding_size, input_length=max_words))
model.add(SimpleRNN(100))
```

```
model.add(Dense(1, activation='sigmoid'))
print(model.summary())
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
batch_size = 64
num_epochs = 1
X_valid, y_valid = X_train[:batch_size], y_train[:batch_size]
X_train2, y_train2 = X_train[batch_size:], y_train[batch_size:]
model.fit(X_train2, y_train2, validation_data=(X_valid, y_valid), batch_size=batch_size,
epochs=num_epochs)
scores = model.evaluate(X_test, y_test, verbose=0)
print('Test accuracy:', scores[1])
```

Output: Test accuracy: 0.6672000288963318

Application:

Speech recognition, voice recognition, time series prediction, and natural language processing

Program 7

Design a Chatbot using bi-directional LSTMs

Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Sample training data
training_data = [
    "Hello",
    "Hi",
    "How are you?",
    "I'm doing well, thanks!",
    "What's your name?",
    "I'm a chatbot.",
    "Tell me a joke",
    "Why don't scientists trust atoms?",
    "Because they make up everything!",
    "exit"
]
# Tokenize the training data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(training_data)
total_words = len(tokenizer.word_index) + 1
# Create input sequences and labels
input_sequences = []
for line in training_data:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
```

```
        input_sequences.append(n_gram_sequence)
# Pad sequences
max_sequence_length = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='pre')
# Create inputs and labels
input_sequences = np.array(input_sequences)
X, y = input_sequences[:, :-1], input_sequences[:, -1]
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
# Build the model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Embedding(total_words, 64, input_length=max_sequence_length - 1))
model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20)))
model.add(tf.keras.layers.Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Train the model
model.fit(X, y, epochs=100, verbose=1)
# Function to generate a response
def generate_response(input_text):
    input_seq = tokenizer.texts_to_sequences([input_text])[0]
    input_seq = pad_sequences([input_seq], maxlen=max_sequence_length - 1, padding='pre')
    predicted_probs = model.predict(input_seq)[0]
    predicted_id = np.argmax(predicted_probs)
    # Check if predicted_id is within a valid range
    if predicted_id >= 1 and predicted_id <= len(tokenizer.word_index):
        predicted_word = list(tokenizer.word_index.keys())[predicted_id - 1]
    else:
        predicted_word = "UNKNOWN_WORD"
    return predicted_word
# Chat with the bot
user_input = "Hello"
```

```python
while user_input != "exit":
    response = generate_response(user_input)
    print(f"User: {user_input}")
    print(f"Chatbot: {response}")
    user_input = input("You: ")
```

Output:

```
1/1 [==============================] - 0s 16ms/step - loss: 1.4513 - accuracy: 0.8500
Epoch 100/100
1/1 [==============================] - 0s 19ms/step - loss: 1.4191 - accuracy: 0.8500
1/1 [==============================] - 1s 877ms/step
User: Hello
Chatbot: a
You: how are you
1/1 [==============================] - 0s 22ms/step
User: how are you
Chatbot: you
You: im fine
1/1 [==============================] - 0s 21ms/step
User: im fine
Chatbot: a
You: what is a
1/1 [==============================] - 0s 26ms/step
User: what is a
Chatbot: chatbot
You: what's your name?
1/1 [==============================] - 0s 24ms/step
User: what's your name?
Chatbot: name
You: exit
```

Applications:

Sentiment analysis, language modeling, speech recognition, and video analysis.