EXP.NO: 1A DATE:	Solving XOR Problem using Deep Neural Networks-Using Numpy

AIM:

ALGORITHM:

EXP.NO: 1B DATE:	Solving XOR Problem using Deep Neural Networks-Using Tensorflow

AIM:

ALGORITHM:

EXP.NO: 1C DATE:	Solving XOR Problem using Deep Neural Networks-Using Keras

AIM:

ALGORITHM:

EXP.NO: 1D DATE:	Solving XOR Problem using Deep Neural Networks-Using PyTorch

AIM:

ALGORITHM:

EXP.NO:02 DATE:	Character Recognition using CNN

AIM:

ALGORITHM:

EXP.NO: 03 DATE:	Face Recognition using CNN

AIM:

ALGORITHM:

EXP.NO: 04 DATE:	Language Modeling using RNN

AIM:

ALGORITHM:

EXP.NO: 05 DATE:	Sentiment Analysis using LSTM

AIM:

ALGORITHM:

EXP.NO: 06 DATE:	Parts of Speech Tagging using Sequence-to-Sequence Architecture

AIM:

ALGORITHM:

EXP.NO: 07 DATE:	Machine Translation using Encoder-Decoder Model

AIM:

ALGORITHM:

EXP.NO: 08 DATE:	Image Augmentation using GANs

AIM:

ALGORITHM:

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
import sys
# Training data: XOR-like binary classification
X = np.array([
    [0, 1],
    [1, 0],
    [1, 1],
    [0, 0]
])
y = np.array([
    [1],
    [1],
    [0],
    [0]
])
# Network architecture
num_i_units = 2  # input layer size
num_h_units = 2  # hidden layer size
num_o_units = 1  # output layer size
# Hyperparameters
learning_rate = 0.1
reg_param = 0.01
max_iter = 5000
m = X.shape[0]  # Number of training examples
# Random seed for reproducibility
np.random.seed(1)
# Weight and bias initialization
W1 = np.random.randn(num_h_units, num_i_units)
```

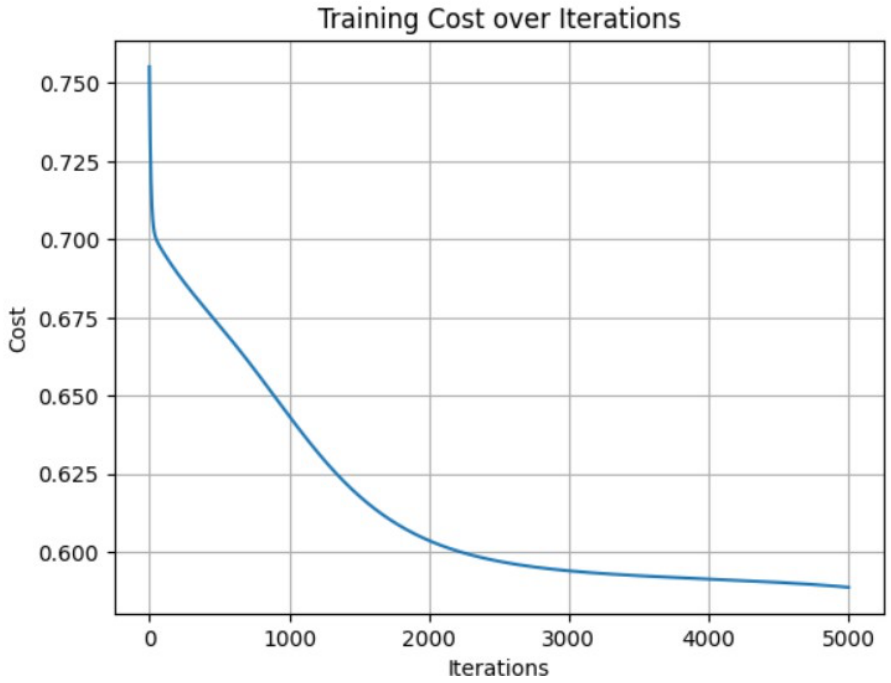
```
W2 = np.random.randn(num_o_units, num_h_units)
B1 = np.random.randn(num_h_units, 1)
B2 = np.random.randn(num_o_units, 1)
# Sigmoid function and derivative
def sigmoid(z, derv=False):
    if derv:
        return z * (1 - z)
    return 1 / (1 + np.exp(-z))
# Cost tracking
cost = np.zeros((max_iter, 1))
# Training function
def train(_W1, _W2, _B1, _B2):
    for i in range(max_iter):
        c = 0
        dW1 = np.zeros_like(_W1)
        dW2 = np.zeros_like(_W2)
        dB1 = np.zeros_like(_B1)
        dB2 = np.zeros_like(_B2)
        for j in range(m):
            a0 = X[j].reshape(-1, 1)
         # Forward pass
            z1 = _W1 @ a0 + _B1
            a1 = sigmoid(z1)
            z2 = _W2 @ a1 + _B2
            a2 = sigmoid(z2)
           # Backward pass
            dz2 = a2 - y[j]
            dW2 += dz2 @ a1.T
            dz1 = (_W2.T @ dz2) * sigmoid(a1, derv=True)
            dW1 += dz1 @ a0.T
```

```
            dB2 += dz2
            dB1 += dz1
            # Compute cost
            c += -y[j] * np.log(a2) - (1 - y[j]) * np.log(1 - a2)
        # Weight update with L2 regularization
        _W1 -= learning_rate * ((dW1 / m) + (reg_param / m) * _W1)
        _W2 -= learning_rate * ((dW2 / m) + (reg_param / m) * _W2)
        _B1 -= learning_rate * (dB1 / m)
        _B2 -= learning_rate * (dB2 / m)
        # Cost per epoch
        cost[i] = (c / m) + (reg_param / (2 * m)) * (np.sum(np.square(_W1)) +
np.sum(np.square(_W2)))
        if i % 500 == 0:
            print(f"Iteration {i}, Cost: {cost[i][0]:.4f}")
        return _W1, _W2, _B1, _B2
# Train the network
W1, W2, B1, B2 = train(W1, W2, B1, B2)
# Plot training cost
plt.plot(range(max_iter), cost)
plt.title("Training Cost over Iterations")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.grid(True)
plt.show()
# Prediction function
def predict(x):
    a0 = x.reshape(-1, 1)
    a1 = sigmoid(W1 @ a0 + B1)
    a2 = sigmoid(W2 @ a1 + B2)
    return int(a2 > 0.5)
# Test the model
```

```
print("\nPredictions:")
for i in range(len(X)):
    print(f"Input: {X[i]} -> Predicted: {predict(X[i])}, Actual: {y[i][0]}")
```

OUTPUT:

```
Iteration 0, Cost: 0.7552
Iteration 500, Cost: 0.6725
Iteration 1000, Cost: 0.6435
Iteration 1500, Cost: 0.6180
Iteration 2000, Cost: 0.6037
Iteration 2500, Cost: 0.5971
Iteration 3000, Cost: 0.5940
Iteration 3500, Cost: 0.5925
Iteration 4000, Cost: 0.5914
Iteration 4500, Cost: 0.5903
```




```
Predictions:
Input: [0 1] -> Predicted: 1, Actual: 1
Input: [1 0] -> Predicted: 1, Actual: 1
Input: [1 1] -> Predicted: 1, Actual: 0
Input: [0 0] -> Predicted: 0, Actual: 0
```

RESULT:

PROGRAM:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
# Training data (XOR-like)
X = np.array([
    [0, 1],
    [1, 0],
    [1, 1],
    [0, 0]
], dtype=np.float32)
y = np.array([
    [1],
    [1],
    [0],
    [0]
], dtype=np.float32)
# Set seed for reproducibility
tf.random.set_seed(1)
# Define the model using Keras Sequential API
model = tf.keras.Sequential([
    tf.keras.layers.Dense(2, activation='sigmoid', input_shape=(2,)),  # Hidden layer
    tf.keras.layers.Dense(1, activation='sigmoid')                     # Output layer
])
# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

```
# Train the model
history = model.fit(X, y, epochs=5000, verbose=0)
# Plot the loss curve
plt.plot(history.history['loss'])
plt.xlabel('Epoch')
plt.ylabel('Binary Cross-Entropy Loss')
plt.title('Training Loss over Epochs')
plt.show()
# Evaluate the model
predictions = model.predict(X)
print("\nPredictions on training data:")
for i, pred in enumerate(predictions):
    print(f"Input: {X[i]} -> Predicted: {pred[0]:.4f} | Actual: {y[i][0]}")
```

OUTPUT:


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step

Predictions on training data:
Input: [0. 1.] -> Predicted: 0.4665 | Actual: 1.0
Input: [1. 0.] -> Predicted: 0.5338 | Actual: 1.0
Input: [1. 1.] -> Predicted: 0.4875 | Actual: 0.0
Input: [0. 0.] -> Predicted: 0.5087 | Actual: 0.0
```

RESULT:

PROGRAM:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
import matplotlib.pyplot as plt
# XOR input data
X = np.array([
    [0, 1],
    [1, 0],
    [1, 1],
    [0, 0]
])
# XOR output labels
y = np.array([
    [1],
    [1],
    [0],
    [0]
])
# Build the Keras model
model = Sequential()
model.add(Dense(2, input_dim=2, activation='sigmoid'))  # Hidden layer with 2 units
model.add(Dense(1, activation='sigmoid'))  # Output layer
# Compile the model
optimizer = SGD(learning_rate=0.1)
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
# Train the model
history = model.fit(X, y, epochs=5000, verbose=0)
```

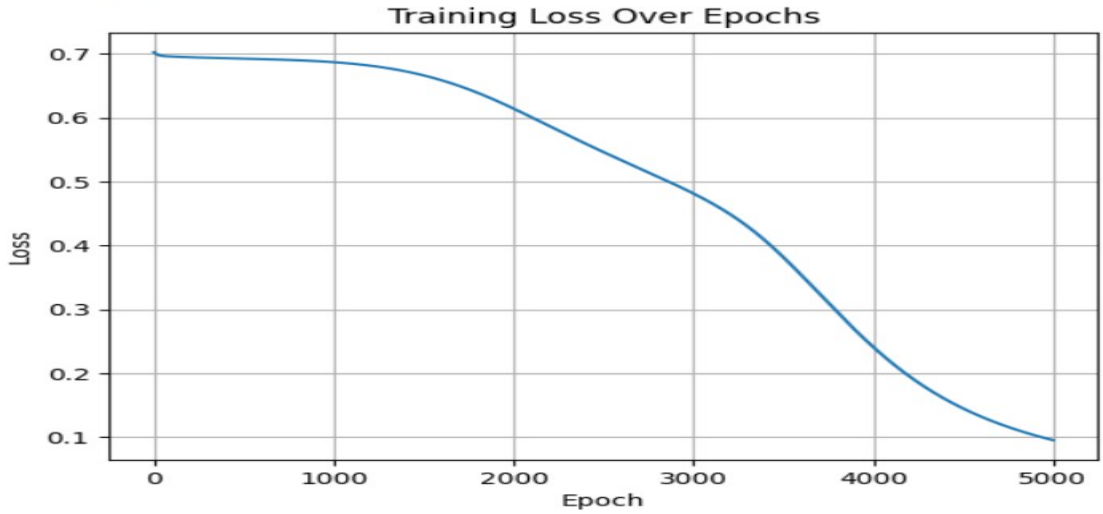
```
# Evaluate the model
loss, accuracy = model.evaluate(X, y)
print(f"\nFinal Loss: {loss:.4f}, Final Accuracy: {accuracy * 100:.2f}%")
# Make predictions
predictions = model.predict(X)
print("\nPredictions:")
for i in range(len(X)):
    print(f"Input: {X[i]} => Predicted: {predictions[i][0]:.4f}")
# Plot the training loss
plt.plot(history.history['loss'])
plt.title('Training Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.show()
```

OUTPUT:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 251ms/step - accuracy: 1.0000 - loss: 0.0954

Final Loss: 0.0954, Final Accuracy: 100.00%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step

Predictions:
Input: [0 1] => Predicted: 0.9186
Input: [1 0] => Predicted: 0.9181
Input: [1 1] => Predicted: 0.1439
Input: [0 0] => Predicted: 0.0542
```



RESULT:

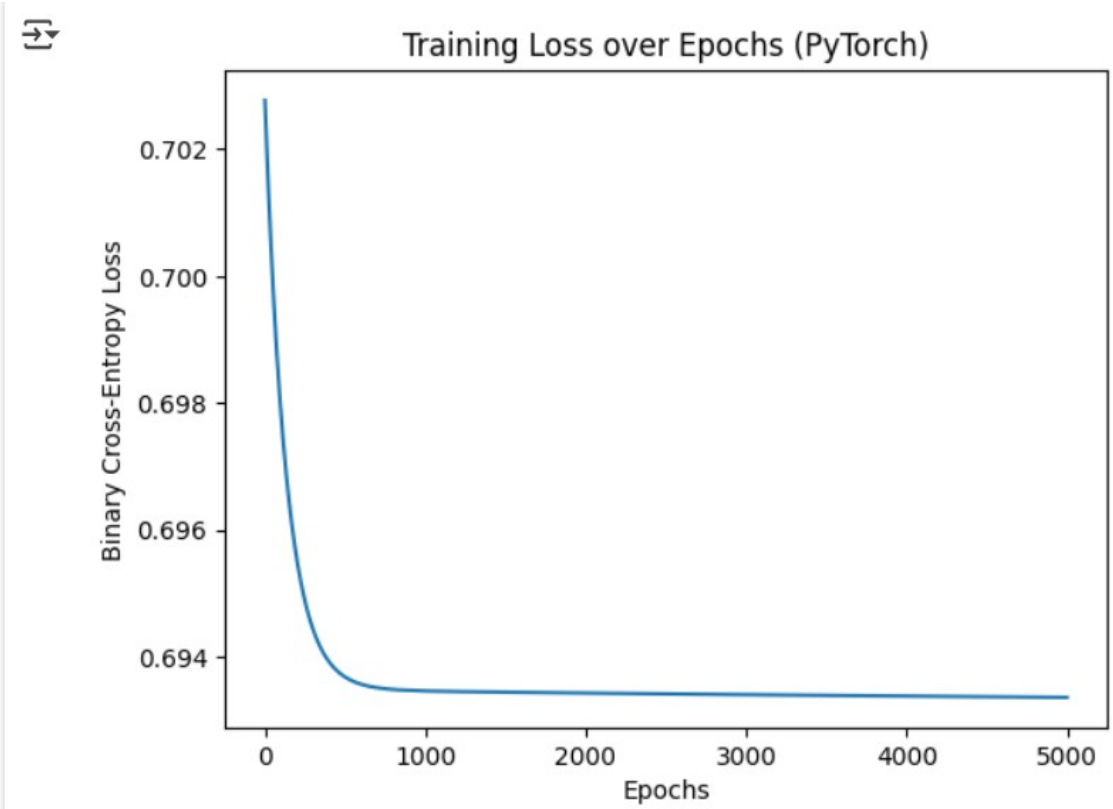
PROGRAM:

```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
# Training data (XOR-like)
X = torch.tensor([
    [0.0, 1.0],
    [1.0, 0.0],
    [1.0, 1.0],
    [0.0, 0.0]
])
y = torch.tensor([
    [1.0],
    [1.0],
    [0.0],
    [0.0]
])
# Set seed for reproducibility
torch.manual_seed(1)
# Define the neural network model
class XORModel(nn.Module):
    def __init__(self):
        super(XORModel, self).__init__()
        self.hidden = nn.Linear(2, 2)  # 2 input → 2 hidden units
        self.output = nn.Linear(2, 1)  # 2 hidden → 1 output
 def forward(self, x):
        x = torch.sigmoid(self.hidden(x))
        x = torch.sigmoid(self.output(x))
        return x
```

```
# Instantiate the model
model = XORModel()
# Define loss function and optimizer
criterion = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
# Training loop
epochs = 5000
loss_history = []
for epoch in range(epochs):
    model.train()
# Forward pass
    outputs = model(X)
    loss = criterion(outputs, y)
 # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
loss_history.append(loss.item())
# Plot training loss
plt.plot(loss_history)
plt.xlabel("Epochs")
plt.ylabel("Binary Cross-Entropy Loss")
plt.title("Training Loss over Epochs (PyTorch)")
plt.show()
# Evaluate the model
model.eval()
with torch.no_grad():
    predictions = model(X)
print("\nPredictions on training data:")
```

```
for i in range(len(X)):
    print(f"Input: {X[i].numpy()} -> Predicted: {predictions[i].item():.4f} | Actual:
{y[i].item()}")
```

OUTPUT:




```
Predictions on training data:
Input: [0. 1.] -> Predicted: 0.5054 | Actual: 1.0
Input: [1. 0.] -> Predicted: 0.4945 | Actual: 1.0
Input: [1. 1.] -> Predicted: 0.5011 | Actual: 0.0
Input: [0. 0.] -> Predicted: 0.4992 | Actual: 0.0
```

RESULT:

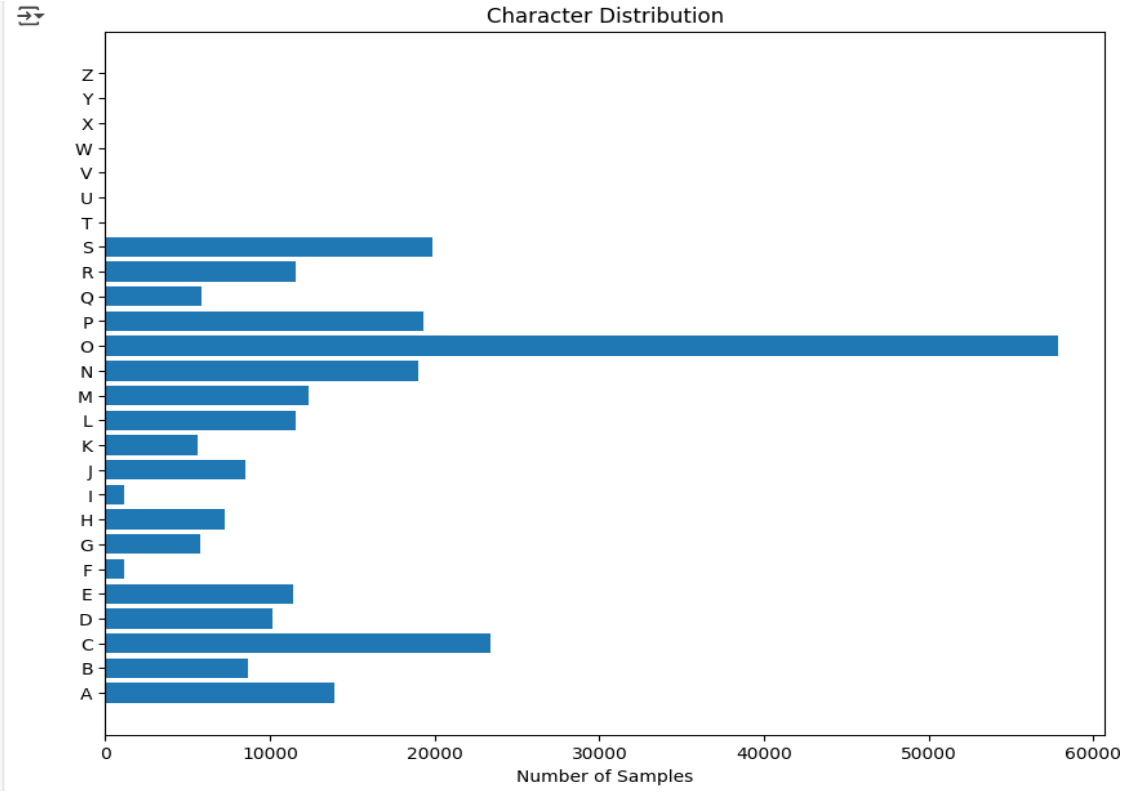
PROGRAM:

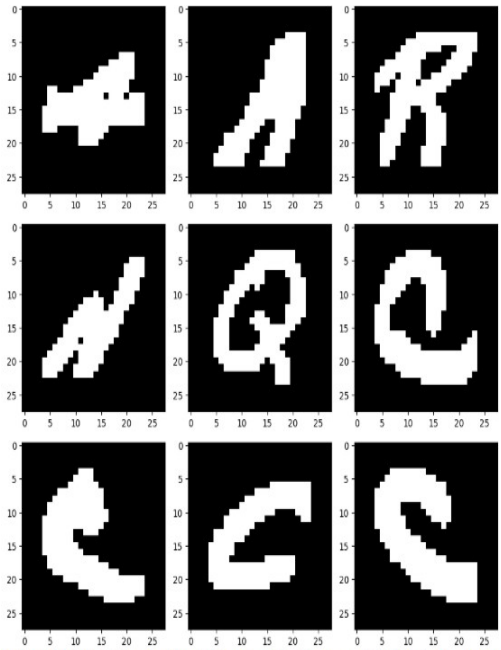
```
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D
from keras.optimizers import Adam
from keras.utils import to_categorical
data = pd.read_csv("A_Z Handwritten Data.csv")
X = data.drop('0', axis=1)
y = data['0'].astype(int)
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=0.2)
train_x = train_x.values.reshape(-1, 28, 28)
test_x = test_x.values.reshape(-1, 28, 28)
word_dict = {i: chr(65 + i) for i in range(26)}
count = np.zeros(26, dtype=int)
for label in y:
    if label < 26:
        count[label] += 1
alphabets = [word_dict[i] for i in range(26)]
plt.figure(figsize=(10, 8))
plt.barh(alphabets, count)
plt.xlabel("Number of Samples")
plt.title("Character Distribution")
plt.show()
samples = shuffle(train_x[:9])
```

```
plt.figure(figsize=(9, 9))
for i in range(9):
    _, img_bin = cv2.threshold(samples[i], 30, 200, cv2.THRESH_BINARY)
    plt.subplot(3, 3, i + 1)
    plt.imshow(img_bin, cmap='gray')
plt.tight_layout()
plt.show()
train_X = train_x.reshape(-1, 28, 28, 1) / 255.0
test_X = test_x.reshape(-1, 28, 28, 1) / 255.0
train_yOHE = to_categorical(train_y, num_classes=26)
test_yOHE = to_categorical(test_y, num_classes=26)
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPool2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2, 2))
    Conv2D(128, kernel_size=(3, 3), activation='relu'),
    MaxPool2D(pool_size=(2, 2))
    Flatten(),
    Dense(64, activation='relu'),
    Dense(128, activation='relu'),
    Dense(26, activation='softmax')
])
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])
print(" Training model...")
history = model.fit(train_X, train_yOHE, epochs=1, validation_data=(test_X, test_yOHE))
print(" Training complete.")
model.save("model_hand.h5")
print("Training Accuracy:", round(history.history['accuracy'][-1] * 100, 2), "%")
print("Validation Accuracy:", round(history.history['val_accuracy'][-1] * 100, 2), "%")
```

```
plt.figure(figsize=(9, 9))
for i in range(9):
    img = test_X[i].reshape(28, 28)
    pred_index = np.argmax(model.predict(test_X[i].reshape(1, 28, 28, 1)))
    true_index = np.argmax(test_yOHE[i])
    pred = word_dict[pred_index]
    actual = word_dict[true_index]
    plt.subplot(3, 3, i + 1)
    plt.imshow(img, cmap='gray')
    plt.title(f"Pred: {pred} | True: {actual}")
    plt.axis('off')
plt.tight_layout()
plt.show()
sample_index = 15  # You can change this to any valid index
sample_image = test_X[sample_index]
true_label_index = np.argmax(test_yOHE[sample_index])
true_label_char = word_dict[true_label_index]
# Show the selected test image
plt.figure(figsize=(4, 4))
plt.imshow(sample_image.reshape(28, 28), cmap='gray')
plt.title(f"Actual Character: {true_label_char}")
plt.axis('off')
plt.show()
# Predict
predicted_index = np.argmax(model.predict(sample_image.reshape(1, 28, 28, 1)))
predicted_char = word_dict[predicted_index]
# Display Prediction
print(" Simulated External Image Prediction")
print(f" Ground Truth: {true_label_char}")
print(f" Model Output: {predicted_char}")
```

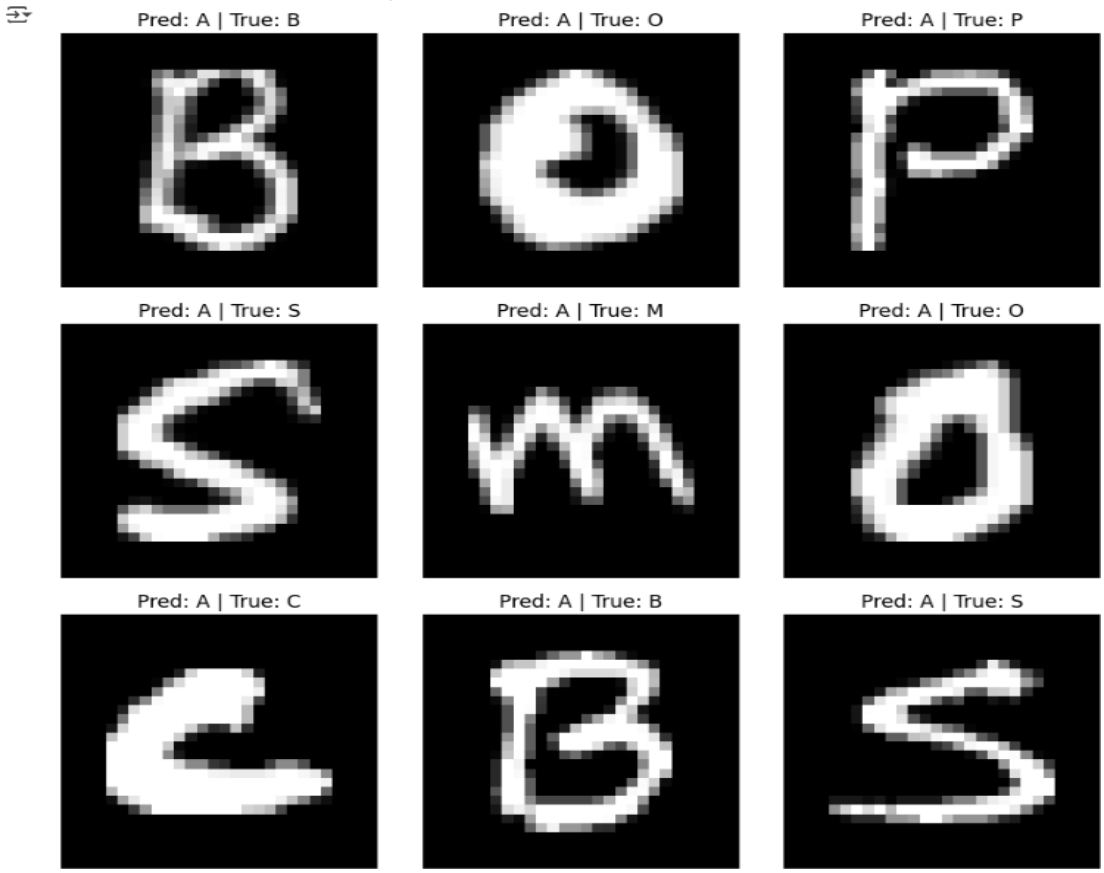
OUTPUT:

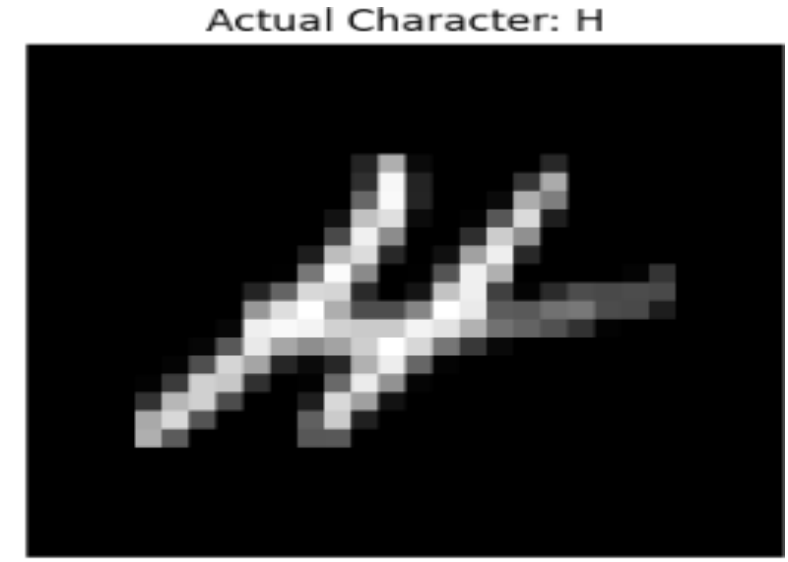




```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
⏳ Training model...
6357/6357 ━━━━━━━━━━━━━━━━━━━━ 288s 45ms/step - accuracy: 0.9075 - loss: nan - val_accuracy: 0.0546 - val_loss: nan
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model
✅ Training complete.
Training Accuracy: 91.05 %
Validation Accuracy: 5.46 %
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```






```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
🔍 Simulated External Image Prediction
🧠 Ground Truth: H
✅ Model Output: A
```

RESULT:

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import cv2
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.utils import to_categorical
# 1. Load Dataset
lfw_people = fetch_lfw_people(min_faces_per_person=100, resize=1.0, color=True)
X = lfw_people.images  # Shape: (n_samples, 125, 94, 3)
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = len(target_names)
# Limit to 100 samples/class for balance
mask = np.zeros(y.shape, dtype=bool)
for label in np.unique(y):
    label_idx = np.where(y == label)[0][:100]
    mask[label_idx] = True
X = X[mask]
y = y[mask]
X = X / 255.0  # Normalize
y_cat = to_categorical(y, num_classes=n_classes)
# 2. Display One Face per Class (with Names)
print("Displaying one sample from each class:")
plt.figure(figsize=(15, 5))
```

```
for i, name in enumerate(target_names):
    idx = np.where(y == i)[0][0]
    plt.subplot(2, 4, i + 1)
    plt.imshow(X[idx])
    plt.title(name)
    plt.axis('off')
plt.tight_layout()
plt.show()
# 3. Save George W Bush Images
george_class_index = list(target_names).index("George W Bush")
george_images = X[y == george_class_index][:5]
george_save_path = "george_faces"
os.makedirs(george_save_path, exist_ok=True)
for i, img in enumerate(george_images):
    img_bgr = (img * 255).astype(np.uint8)
    cv2.imwrite(f"{george_save_path}/george_{i+1}.jpg", cv2.cvtColor(img_bgr,
cv2.COLOR_RGB2BGR))
print(f" Saved 5 George W Bush images to '{george_save_path}/'")
# 4. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y_cat, test_size=0.2, stratify=y, random_state=42)
# 5. CNN Model
model = Sequential([
    Input(shape=X.shape[1:]),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
```

```
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(n_classes, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# 6. Train Model
print("Training started...")
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=20,
    batch_size=32,
    verbose=1
)
print(" Training complete.")
# 7. Accuracy Plot
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.legend()
plt.grid(True)
plt.show()
# 8. Confusion Matrix
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=target_names, yticklabels=target_names)
```

```
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()
# 9. Predict from George Saved Image
external_img_path = f"{george_save_path}/george_1.jpg"
external_img = cv2.imread(external_img_path)
external_img_rgb = cv2.cvtColor(external_img, cv2.COLOR_BGR2RGB)
external_img_rgb = cv2.resize(external_img_rgb, (94, 125)) / 255.0
external_input = np.expand_dims(external_img_rgb, axis=0)
external_pred = model.predict(external_input)
external_pred_label = target_names[np.argmax(external_pred)]
plt.imshow(external_img_rgb)
plt.title(f"External George W Bush Image Prediction: {external_pred_label}")
plt.axis('off')
plt.show()
# 10. Predict from Test Set Sample
idx = 0
test_img = X_test[idx]
true_label = target_names[np.argmax(y_test[idx])]
pred_label = target_names[np.argmax(model.predict(np.expand_dims(test_img, axis=0)))]
plt.imshow(test_img)
plt.title(f"Test Image - True: {true_label} | Pred: {pred_label}")
plt.axis('off')
plt.show()
```

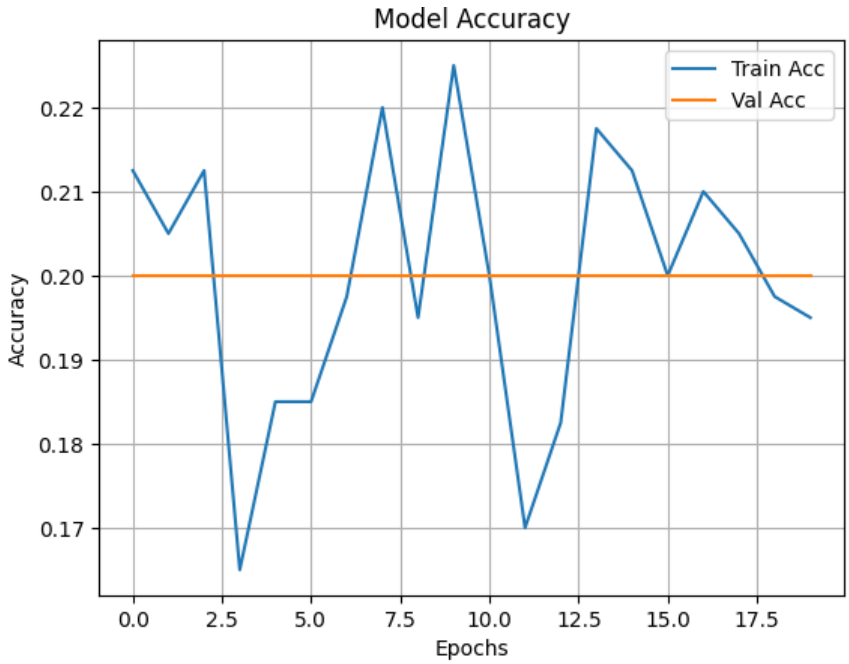
OUTPUT:

```
Classes: ['Colin Powell' 'Donald Rumsfeld' 'George W Bush' 'Gerhard Schroeder'
 'Tony Blair']
Shape: (1140, 128, 128, 3)
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usi
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

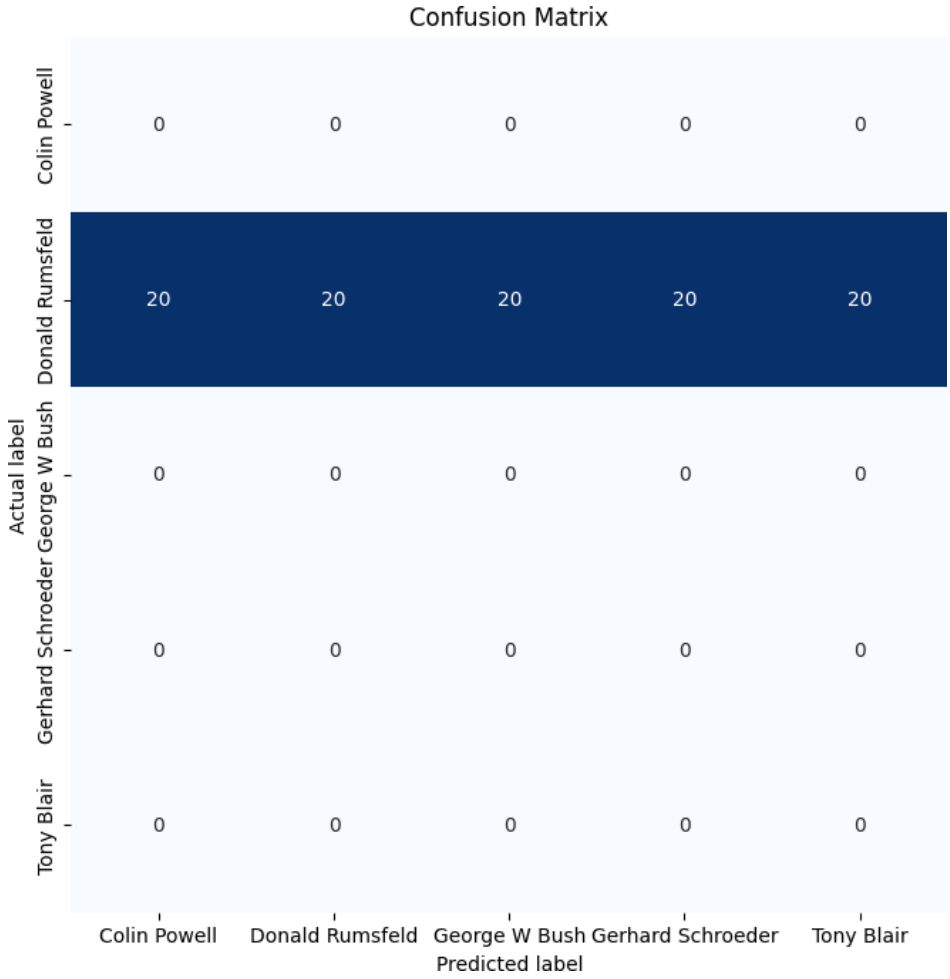
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 128)	1,605,760
dense_1 (Dense)	(None, 5)	645

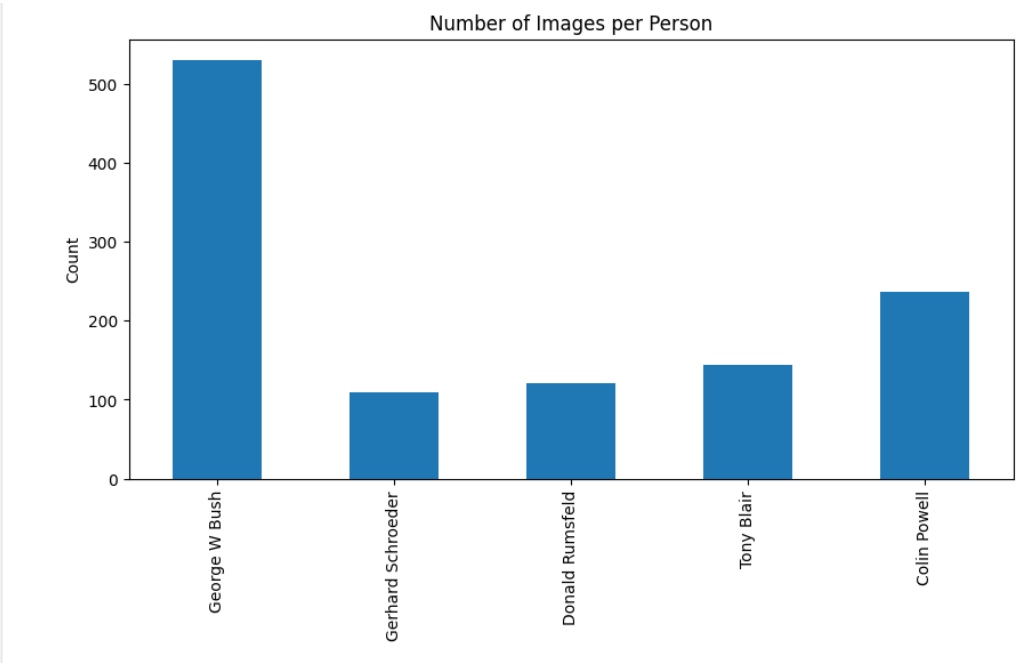
Total params: 1,662,725 (6.34 MB)
Trainable params: 1,662,725 (6.34 MB)
Non-trainable params: 0 (0.00 B)

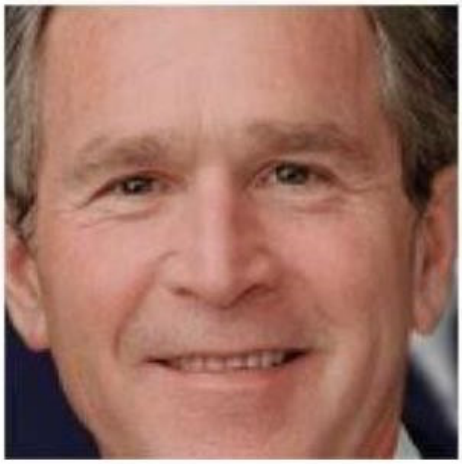

```
Epoch 1/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 13s 654ms/step - accuracy: 0.1778 - loss: 1.6119 - val_accuracy: 0.2000 - val_loss: 1.6095
Epoch 2/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 22s 747ms/step - accuracy: 0.1934 - loss: 1.6095 - val_accuracy: 0.2000 - val_loss: 1.6094
Epoch 3/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 21s 787ms/step - accuracy: 0.1642 - loss: 1.6095 - val_accuracy: 0.2000 - val_loss: 1.6094
Epoch 4/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 12s 748ms/step - accuracy: 0.1865 - loss: 1.6095 - val_accuracy: 0.2000 - val_loss: 1.6094
Epoch 5/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 11s 712ms/step - accuracy: 0.2253 - loss: 1.6093 - val_accuracy: 0.2000 - val_loss: 1.6094
Epoch 6/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 21s 750ms/step - accuracy: 0.2144 - loss: 1.6095 - val_accuracy: 0.2000 - val_loss: 1.6094
Epoch 7/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 20s 720ms/step - accuracy: 0.2066 - loss: 1.6095 - val_accuracy: 0.2000 - val_loss: 1.6094
Epoch 8/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 20s 671ms/step - accuracy: 0.1674 - loss: 1.6096 - val_accuracy: 0.2000 - val_loss: 1.6094
Epoch 9/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 22s 785ms/step - accuracy: 0.1752 - loss: 1.6095 - val_accuracy: 0.2000 - val_loss: 1.6094
Epoch 10/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 20s 758ms/step - accuracy: 0.1824 - loss: 1.6096 - val_accuracy: 0.2000 - val_loss: 1.6094
```




```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 548ms/step
```







```
1/1[==============================]-0s48ms/step
ColinPowell:0.20101844
Donald Rumsfeld: 0.20214622
George W Bush: 0.2216323
GerhardSchroeder:0.21147959
TonyBlair:0.16372345
```

RESULT:

PROGRAM:

```
from __future___import unicode_literals, print_function, division
from io import open
import glob
import os
import unicodedata
import string
import random
import time
import math
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
# All valid characters (letters and some punctuation)
all_letters = string.ascii_letters + " .,;'-"
n_letters = len(all_letters) + 1  # Plus EOS marker
# Convert Unicode string to plain ASCII
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn' and c in all_letter )
# Load data
def findFiles(path): return glob.glob(path)
def readLines(filename):
    with open(filename, encoding='utf-8') as f:
        return [unicodeToAscii(line.strip()) for line in f]
category_lines = {}
all_categories = []
for filename in findFiles('data/names/*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
```

```
    all_categories.append(category)
    lines = readLines(filename)
    category_lines[category] = lines
n_categories = len(all_categories)
if n_categories == 0:
    raise RuntimeError('No data found — Download and extract
https://download.pytorch.org/tutorial/data.zip into the current directory.')
print("Found", n_categories, "categories:", all_categories[:5], "...")
# Define RNN
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.i2h = nn.Linear(n_categories + input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(n_categories + input_size + hidden_size, output_size)
        self.o2o = nn.Linear(hidden_size + output_size, output_size)
        self.dropout = nn.Dropout(0.1)
        self.softmax = nn.LogSoftmax(dim=1)
    def forward(self, category, input, hidden):
        input_combined = torch.cat((category, input, hidden), 1)
        hidden = self.i2h(input_combined)
        output = self.i2o(input_combined)
        output_combined = torch.cat((hidden, output), 1)
        output = self.o2o(output_combined)
        output = self.dropout(output)
        output = self.softmax(output)
        return output, hidden
    def initHidden(self):
        return torch.zeros(1, self.hidden_size)
# Encoding functions
def categoryTensor(category):
```

```
    tensor = torch.zeros(1, n_categories)
    tensor[0][all_categories.index(category)] = 1
    return tensor
def inputTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li in range(len(line)):
        letter = line[li]
        tensor[li][0][all_letters.find(letter)] = 1
    return tensor
def targetTensor(line):
    letter_indexes = [all_letters.find(line[li]) for li in range(1, len(line))]
    letter_indexes.append(n_letters - 1)  # EOS
    return torch.LongTensor(letter_indexes)
# FIXED: avoid lines with < 2 chars
def randomTrainingExample():
    while True:
        category = random.choice(all_categories)
        line = random.choice(category_lines[category])
        if len(line) < 2:
            continue  # Skip too-short lines
        category_tensor = categoryTensor(category)
        input_tensor = inputTensor(line)
        target_tensor = targetTensor(line)
        if target_tensor.size(0) == 0:
            continue  # Additional safety check
        return category, line, category_tensor, input_tensor, target_tensor
# Training
criterion = nn.NLLLoss()
learning_rate = 0.005
rnn = RNN(n_letters, 128, n_letters)
```

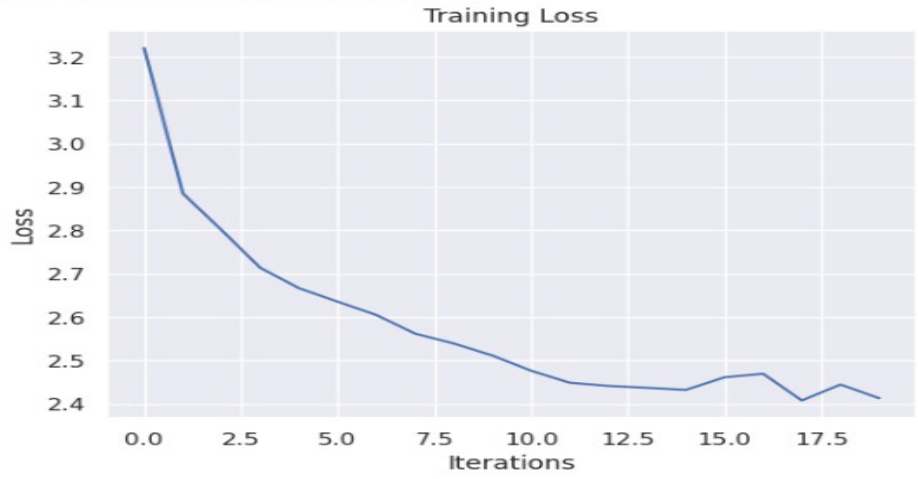
```
def train(category_tensor, input_tensor, target_tensor):
    target_tensor.unsqueeze_(-1)
    hidden = rnn.initHidden()
    rnn.zero_grad()
    loss = 0
for i in range(input_tensor.size(0)):
        output, hidden = rnn(category_tensor, input_tensor[i], hidden)
        l = criterion(output, target_tensor[i])
        loss   +=   l
 loss.backward()
for p in rnn.parameters():
        p.data.add_(p.grad.data, alpha=-learning_rate
return loss.item() / input_tensor.size(0)
def timeSince(since):
    now  =  time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
    return f'{m}m {int(s)}s'
start = time.time()
print_every = 5000
plot_every = 500
total_loss = 0
all_losses = []
for iter in range(1, n_iters + 1):
    category, line, category_tensor, input_tensor, target_tensor = randomTrainingExample()
    loss = train(category_tensor, input_tensor, target_tensor)
    total_loss += loss
if iter % print_every == 0:
        print(f'{timeSince(start)} ({iter} {iter / n_iters * 100:.2f}%) Loss: {loss:.4f}')
```

```
 if iter % plot_every == 0:
        all_losses.append(total_loss / plot_every)
        total_loss = 0
# Plot loss
plt.figure()
plt.plot(all_losses)
plt.title("Training Loss")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
# Sampling
max_length = 20
def sample(category, start_letter='A'):
    with torch.no_grad():
        category_tensor = categoryTensor(category)
        input = inputTensor(start_letter)
        hidden = rnn.initHidden()
        output_name = start_letter
 for i in range(max_length):
            output, hidden = rnn(category_tensor, input[0], hidden)
            topv, topi = output.topk(1)
            topi = topi[0][0]
            if topi == n_letters - 1:
                break
            else:
                letter = all_letters[topi]
                output_name += letter
                input = inputTensor(letter)
```

```
        return output_name
    def samples(category, start_letters='ABC'):
        print(f"\nSamples for {category}:")
        for start_letter in start_letters:
            print(f"{start_letter} ➜", sample(category, start_letter))
    # Show samples
    samples('Italian', 'MAR')
    samples('Japanese', 'TAK')
    samples('French', 'LUC')
    samples('Russian', 'AND')
```

OUTPUT:

```
Found 18 categories: ['Irish', 'Korean', 'Portuguese', 'Dutch', 'Vietnamese'] ...
0m 21s (5000 50.00%) Loss: 2.4227
0m 42s (10000 100.00%) Loss: 2.5359
```




```
Samples for Italian:
M ➜ Marin
A ➜ Allain
R ➜ Roni

Samples for Japanese:
T ➜ Tama
A ➜ Aramas
K ➜ Koura

Samples for French:
L ➜ Lerre
U ➜ Ueren
C ➜ Cherro

Samples for Russian:
A ➜ Arevev
N ➜ Nakin
D ➜ Devevevev
```

RESULT:

PROGRAM:

```
import re
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras import Sequential
from keras.layers import Embedding, Bidirectional, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
# Load dataset and reduce sample size for faster training
data = pd.read_csv('IMDB Dataset.csv').sample(n=2000, random_state=42)
# Clean HTML, URLs, and non-alphanumeric
def clean_text(text):
    text = re.sub('<.*?>', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub(r'[^a-zA-Z ]', ' ', text)
    return text.lower()
data['review'] = data['review'].apply(clean_text)
# NLTK Preprocessing
nltk.download('stopwords')
nltk.download('wordnet')
stop_words = set(stopwords.words('english'))
lemmatizer = nltk.stem.WordNetLemmatizer()
def preprocess(text):
    words = text.split()
```

```
    words = [lemmatizer.lemmatize(w) for w in words if w not in stop_words]
    return ' '.join(words)
data['review'] = data['review'].apply(preprocess)
# Prepare input and labels
reviews = data['review'].values
encoder = LabelEncoder()
labels = encoder.fit_transform(data['sentiment'].values) # Use numerical labels
# Tokenize and pad sequences
vocab_size = 5000
embedding_dim = 64
max_length = 100
tokenizer = Tokenizer(num_words=vocab_size, oov_token="<OOV>")
tokenizer.fit_on_texts(reviews)
sequences = tokenizer.texts_to_sequences(reviews)
padded = pad_sequences(sequences, maxlen=max_length, padding='post', truncating='post')
# Train/test split using padded sequences and numerical labels
X_train, X_test, y_train, y_test = train_test_split(padded, labels, stratify=labels,
test_size=0.2, random_state=42)
# Model
model = Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(LSTM(64, return_sequences=False)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
# Early stopping to avoid overfitting
es = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
# Train
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2, batch_size=64,
callbacks=[es], verbose=2)
```

```
# Evaluate
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob >= 0.5).astype(int).flatten()
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred, target_names=encoder.classes_))
# Test on new reviews
test_reviews = [
    "The movie was touching and wonderful!",
    "This was the worst movie I've ever watched.",
    "The acting was great but the plot was boring."]
test_seq = tokenizer.texts_to_sequences(test_reviews)
test_pad = pad_sequences(test_seq, maxlen=max_length, padding='post')
predictions = model.predict(test_pad)
for i, review in enumerate(test_reviews):
    sentiment = "Positive" if predictions[i] >= 0.5 else "Negative"
    print(f"\nReview: {review}")
    print("Predicted Sentiment:", sentiment)
```

OUTPUT:

review	sentiment
One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me. The first thing that stru	positive
A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the e	positive
I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dial	positive
Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time. This movie is slower than a soap opera... and suddenly,	negative
Petter Mattei's "Love in the Time of Money" is a visually stunning film to watch. Mr. Mattei offers us a vivid portrait about human relations. This is a movie that seems to be telling us what m	positive
Probably my all-time favorite movie, a story of selflessness, sacrifice and dedication to a noble cause, but it's not preachy or boring. It just never gets old, despite my having seen it some 15 o	positive
I sure would like to see a resurrection of a up dated Seahunt series with the tech they have today it would bring back the kid excitement in me.I grew up on black and white TV and Seahunt wi	positive
This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 years were brilliant, but things dropped off after that. By 1990, the show was not really funny	negative
Encouraged by the positive comments about this film on here I was looking forward to watching this film. Bad mistake. I've seen 950+ films and this is truly one of the worst of them - it's awf	negative
If you like original gut wrenching laughter you will like this movie. If you are young or old then you will love this movie, hell even my mom liked it. Great Camp!!!	positive
Phil the Alien is one of those quirky films where the humour is based around the oddness of everything rather than actual punchlines. At first it was very odd and pretty funny but a	negative
I saw this movie when I was about 12 when it came out. I recall the scariest scene was the big bird eating men dangling helplessly from parachutes right out of the air. The horror. The horror.	negative
So im not a big fan of Boll's work but then again not many are. I enjoyed his movie Postal (maybe im the only one). Boll apparently bought the rights to use Far Cry long ago even before the ga	negative
The cast played Shakespeare. Shakespeare lost. I appreciate that this is trying to bring Shakespeare to the masses, but why ruin something so good. Is it bec	negative
This a fantastic movie of three prisoners who become famous. One of the actors is george clooney and I'm not a fan but this roll is not bad. Another good thing about the movie is the soundt	positive
Kind of drawn in by the erotic scenes, only to realize this was one of the most amateurish and unbelievable bits of film I've ever seen. Sort of like a high school film project. What was Rosann	negative
Some films just simply should not be remade. This is one of them. In and of itself it is not a bad film. But it fails to capture the flavor and the terror of the 1963 film of the same title. Liam Nee	positive
This movie made it into one of my top 10 most awful movies. Horrible. There wasn't a continuous minute where there wasn't a fight with one monster or another. There was no	negative
I remember this film,it was the first film i had watched at the cinema the picture was dark in places i was very nervous it was back in 74/75 my Dad took me my brother & sister to Newbury ci	positive
An awful film! It must have been up against some real stinkers to be nominated for the Golden Globe. They've taken the story of the first famous female Renaissance painter and mangled it b	negative
After the success of Die Hard and it's sequels it's no surprise really that in the 1990s, a glut of 'Die Hard on a .....' movies cashed in on the wrong guy, wrong place, wrong time concept. That is	positive
I had the terrible misfortune of having to view this "b-movie" in it's entirety. All I have to say is--- save your time and money!!! This has got to be the worst b-movie of all time, it sl	negative
What an absolutely stunning movie, if you have 2.5 hrs to kill, watch it, you won't regret it, it's too much fun! Rajnikanth carries the movie on his shoulders and although there isn't anything m	positive
First of all, let's get a few things straight here: a) I AM an anime fan- always has been as a matter of fact (I used to watch Speed Racer all the time in Preschool). b) I DO like several B-Movies b	negative
This was the worst movie I saw at WorldFest and it also received the least amount of applause afterwards! I can only think it is receiving such recognition based on the amount of known acto	negative
The Karen Carpenter Story shows a little more about singer Karen Carpenter's complex life. Though it fails in giving accurate facts, and details. Cynthia Gibb (portrays Karen) was n	positive

```
Epoch 1/10
20/20 - 8s - 387ms/step - accuracy: 0.5234 - loss: 0.6914 - val_accuracy: 0.5156 - val_loss: 0.6917
Epoch 2/10
20/20 - 4s - 201ms/step - accuracy: 0.6852 - loss: 0.6305 - val_accuracy: 0.7031 - val_loss: 0.5613
Epoch 3/10
20/20 - 3s - 131ms/step - accuracy: 0.8562 - loss: 0.3760 - val_accuracy: 0.7250 - val_loss: 0.5346
Epoch 4/10
20/20 - 6s - 291ms/step - accuracy: 0.9086 - loss: 0.2276 - val_accuracy: 0.7625 - val_loss: 0.5668
Epoch 5/10
20/20 - 4s - 218ms/step - accuracy: 0.9727 - loss: 0.0952 - val_accuracy: 0.8156 - val_loss: 0.6896
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step

Accuracy: 0.6875
              precision    recall  f1-score   support

    negative       0.88      0.45      0.60       205
    positive       0.62      0.94      0.75       195

    accuracy                           0.69       400
   macro avg       0.75      0.69      0.67       400
weighted avg       0.75      0.69      0.67       400

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
200/200 - 114s - 568ms/step - accuracy: 0.7371 - loss: 0.4971 - val_accuracy: 0.8516 - val_loss: 0.3624
Epoch 2/20
200/200 - 144s - 720ms/step - accuracy: 0.9065 - loss: 0.2492 - val_accuracy: 0.8709 - val_loss: 0.3120
Epoch 3/20
200/200 - 144s - 721ms/step - accuracy: 0.9553 - loss: 0.1340 - val_accuracy: 0.8584 - val_loss: 0.3968
Epoch 4/20
200/200 - 135s - 676ms/step - accuracy: 0.9735 - loss: 0.0797 - val_accuracy: 0.8341 - val_loss: 0.4613
Epoch 5/20
200/200 - 104s - 518ms/step - accuracy: 0.9821 - loss: 0.0584 - val_accuracy: 0.8434 - val_loss: 0.6046

Test Accuracy: 85.83%
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

Review: The movie was touching and wonderful!
Predicted Sentiment: Positive

Review: This was the worst movie I've ever watched.
Predicted Sentiment: Negative

Review: The acting was great but the plot was boring.
Predicted Sentiment: Negative
```

RESULT:

PROGRAM:

```
import os
import json
import math
from collections import defaultdict, Counter
# Constants
UNK = "<unk>"
START = "<s>"
END = "</s>"
# Create output folder if not exists
os.makedirs("output", exist_ok=True)
def read_data(file_path):
    sentences = []
    sentence = []
    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            if line.strip():
                parts = line.strip().split('\t')
                if len(parts) == 3:
                    sentence.append((parts[1], parts[2]))
             else:
                if sentence:
                    sentences.append(sentence)
                    sentence = []
        if sentence:
            sentences.append(sentence)
    return sentences
# Task 1: Vocabulary Creation
def build_vocab(sentences, min_freq=3):
    word_freq = Counter(word for sent in sentences for word, _ in sent
```

```
 vocab = {word for word, freq in word_freq.items() if freq >= min_freq}
    with open("output/vocab_frequent.txt", "w", encoding="utf-8") as f:
        for word in sorted(vocab):
            f.write(word + "\n")
    return vocab
# Task 2: HMM Model Training
def train_hmm(sentences, vocab):
    transition_counts = defaultdict(Counter)
    emission_counts = defaultdict(Counter)
    pos_counts = Counter()
    tag_set = set()
for sent in sentences:
        tags = [START] + [tag for _, tag in sent] + [END]
        words = [UNK if word not in vocab else word for word, _ in sent]
 for i in range(len(words)):
            transition_counts[tags[i]][tags[i+1]] += 1
            emission_counts[tags[i+1]][words[i]] += 1
            pos_counts[tags[i+1]] += 1
            tag_set.add(tags[i+1])
# Save tag set
    with open("output/pos.txt", "w", encoding="utf-8") as f:
        for tag in sorted(tag_set):
            f.write(tag + "\n")
 # Convert to log-probabilities
    transition_probs = {
        prev: {cur: math.log(count / sum(tag_dict.values()))
               for cur, count in tag_dict.items()}
        for prev, tag_dict in transition_counts.items() }
    emission_probs = {
        tag: {word: math.log(count / pos_counts[tag])
```

```
            for word, count in word_dict.items()}
        for tag, word_dict in emission_counts.items() }
model = {
        "transition": transition_probs,
        "emission": emission_probs  }
with open("output/hmm.json", "w", encoding="utf-8") as f:
        json.dump(model, f)
  return transition_probs, emission_probs, tag_set
# Task 3: Greedy Decoding
def greedy_decode(sentence, transition, emission, tag_set, vocab):
    sentence = [UNK if word not in vocab else word for word in sentence]
    pred_tags = []
    prev_tag = START
 for word in sentence:
        best_tag = None
        best_score = float('-inf')
        for tag in tag_set:
            emit_score = emission.get(tag, {}).get(word, math.log(1e-10))
            trans_score = transition.get(prev_tag, {}).get(tag, math.log(1e-10))
            score = emit_score + trans_score
            if score > best_score:
                best_score = score
                best_tag = tag
        pred_tags.append(best_tag)
        prev_tag = best_tag
    return pred_tags
# Task 4: Viterbi Decoding
def viterbi_decode(sentence, transition, emission, tag_set, vocab):
    sentence = [UNK if word not in vocab else word for word in sentence]
    V = [{}]
```

```
    backpointer = [{}]
 for tag in tag_set:
        V[0][tag] = transition.get(START, {}).get(tag, math.log(1e-10)) + emission.get(tag,
{}).get(sentence[0], math.log(1e-10))
        backpointer[0][tag] = None
for t in range(1, len(sentence)):
        V.append({})
        backpointer.append({})
        for tag in tag_set:
            max_prob, best_prev = max(
                ((V[t-1][prev_tag] + transition.get(prev_tag, {}).get(tag, math.log(1e-10)),
prev_tag)
                 for prev_tag in tag_set),
                key=lambda x: x[0])
            V[t][tag] = max_prob + emission.get(tag, {}).get(sentence[t], math.log(1e-10))
            backpointer[t][tag] = best_prev
    last_tag = max(V[-1], key=V[-1].get)
    best_path = [last_tag]
       for t in range(len(sentence)-1, 0, -1):
        best_path.insert(0, backpointer[t][best_path[0]])
return best_path
# Accuracy evaluation
def evaluate(test_data, transition, emission, tag_set, vocab, decoder):
    correct = total = 0
    for sent in test_data:
        words, true_tags = zip(*sent)
        pred_tags = decoder(words, transition, emission, tag_set, vocab)
        for t, p in zip(true_tags, pred_tags):
            correct += int(t == p)
            total += 1
    return correct / total * 100
```

```
# Main function
if __name__ == "__main__":
    print("Reading training data...")
    train_data = read_data("train_dataset.txt")
    print("Reading dev data...")
    dev_data = read_data("dev_dataset.txt")
    print("Building vocabulary...")
    vocab = build_vocab(train_data)
    print("Training HMM...")
    transition, emission, tag_set = train_hmm(train_data, vocab)
    print("Evaluating Greedy Decoding...")
    greedy_acc = evaluate(dev_data, transition, emission, tag_set, vocab, greedy_decode)
    print(f"Greedy Accuracy: {greedy_acc:.2f}%")
    print("Evaluating Viterbi Decoding...")
    viterbi_acc = evaluate(dev_data, transition, emission, tag_set, vocab, viterbi_decode)
    print(f"Viterbi Accuracy: {viterbi_acc:.2f}%")
```

OUTPUT:

```
1       quickly ADV
2       they    PRON
3       the     DET
4       sees    VERB

1       tall    ADJ
2       happily ADV
3       an      DET
4       writes  VERB

1       they    PRON
2       tall    ADJ
3       dog     NOUN

1       an      DET
2       a       DET
3       book    NOUN

1       eats    VERB
2       a       DET
3       reads   VERB
4       happily ADV
5       she     PRON

1       a       DET
2       I       PRON
3       eats    VERB
4       sleeps  VERB

1       a       DET
2       drives  VERB
3       slowly  ADV

1       cat     NOUN
2       a       DET
3       big     ADJ
4       happy   ADJ

1       we      PRON
2       a       DET
3       book    NOUN
```


```
1       quickly ADV
2       they    PRON
3       the     DET
4       sees    VERB

1       tall    ADJ
2       happily ADV
3       an      DET
4       writes  VERB

1       they    PRON
2       tall    ADJ
3       dog     NOUN

1       an      DET
2       a       DET
3       book    NOUN

1       eats    VERB
2       a       DET
3       reads   VERB
4       happily ADV
5       she     PRON

1       a       DET
2       I       PRON
3       eats    VERB
4       sleeps  VERB

1       a       DET
2       drives  VERB
3       slowly  ADV

1       cat     NOUN
2       a       DET
3       big     ADJ
4       happy   ADJ

1       we      PRON
2       a       DET
3       book    NOUN
```

```
Max sequence length: 182
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Model: "functional_2"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 182)	0
embedding_2 (Embedding)	(None, 182, 128)	1,243,136
bidirectional_2 (Bidirectional)	(None, 182, 128)	98,816
time_distributed_2 (TimeDistributed)	(None, 182, 11)	1,419

```
 Total params: 1,343,371 (5.12 MB)
 Trainable params: 1,343,371 (5.12 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 19s 264ms/step - accuracy: 0.8122 - loss: 0.6945 - val_accuracy: 0.9163 - val_loss: 0.2196
Epoch 2/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 9s 225ms/step - accuracy: 0.9099 - loss: 0.2376 - val_accuracy: 0.9184 - val_loss: 0.2141
Epoch 3/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 10s 219ms/step - accuracy: 0.9152 - loss: 0.2277 - val_accuracy: 0.9182 - val_loss: 0.2127
Epoch 4/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 10s 222ms/step - accuracy: 0.9180 - loss: 0.2257 - val_accuracy: 0.9175 - val_loss: 0.2119
Epoch 5/5
42/42 ━━━━━━━━━━━━━━━━━━━━ 10s 224ms/step - accuracy: 0.9190 - loss: 0.2298 - val_accuracy: 0.9182 - val_loss: 0.2112
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.9083 - loss: 0.2366
Test Accuracy: 91.07%
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

Sample Prediction:
terroriston -> adp
ka -> pron
baap -> pron
hai -> det
, -> det
modi -> pron
. -> adj
bharat -> pron
ki -> det
shan -> pron
hai'
 'modi -> pron
. -> adj
jai -> noun
hind -> det
vandematram -> det
```

```
Reading training data...
Reading dev data...
Building vocabulary...
Training HMM...
Evaluating Greedy Decoding...
Greedy Accuracy: 100.00%
Evaluating Viterbi Decoding...
Viterbi Accuracy: 100.00%
```

RESULT:

PROGRAM:

```
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense
# Parameters
batch_size = 64
epochs = 10
latent_dim = 256
num_samples = 10000  # Number of sentence pairs used for training
# Load dataset
data_path = "fra.txt"  # Make sure this file is in the same directory
# Prepare data
input_texts = []
target_texts = []
input_chars = set()
target_chars = set()
with open(data_path, "r", encoding="utf-8") as f:
    lines = f.read().split("\n")
for line in lines[:min(num_samples, len(lines) - 1)]:
    input_text, target_text, _ = line.split("\t")
    target_text = "\t" + target_text + "\n"  # Start and End tokens
    input_texts.append(input_text)
    target_texts.append(target_text)
    input_chars.update(input_text)
    target_chars.update(target_text)
input_chars = sorted(list(input_chars))
target_chars = sorted(list(target_chars))
num_encoder_tokens = len(input_chars)
num_decoder_tokens = len(target_chars)
max_encoder_seq_length = max(len(txt) for txt in input_texts)
```

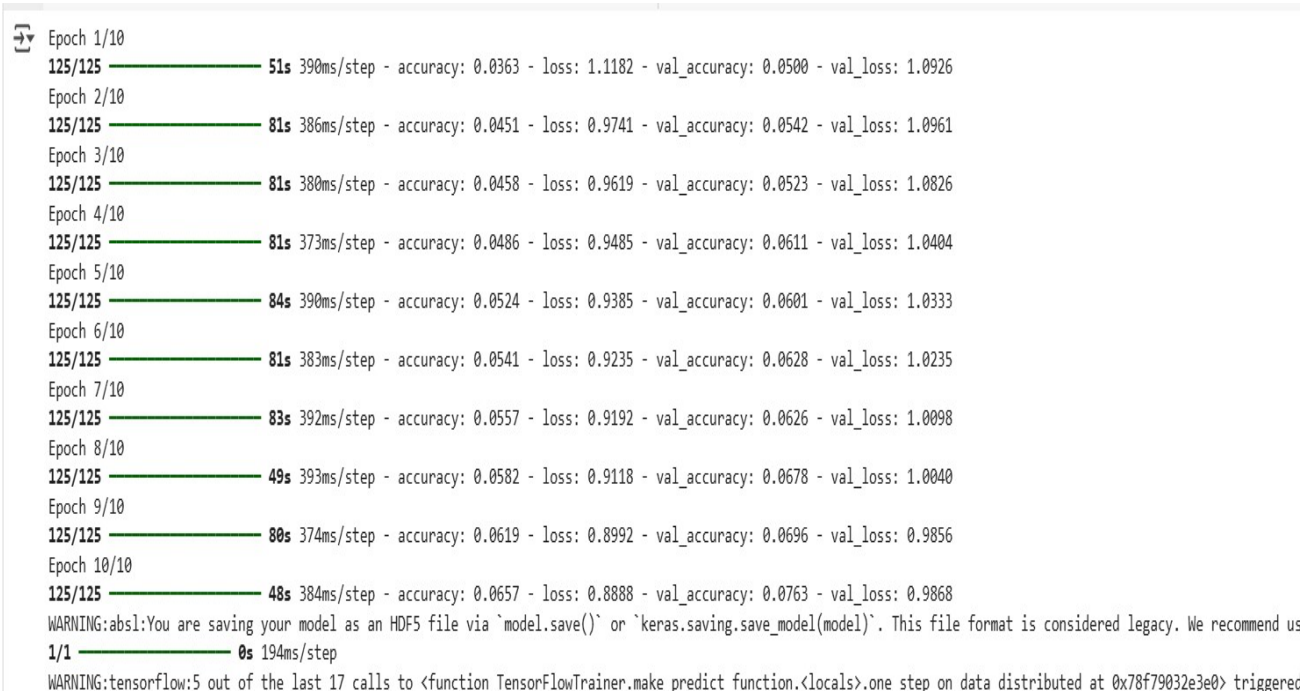
```
max_decoder_seq_length = max(len(txt) for txt in target_texts)
input_token_index = dict((char, i) for i, char in enumerate(input_chars))
target_token_index = dict((char, i) for i, char in enumerate(target_chars))
# Prepare encoder & decoder input/output data
encoder_input_data = np.zeros((len(input_texts), max_encoder_seq_length,
num_encoder_tokens), dtype="float32")
decoder_input_data = np.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype="float32")
decoder_target_data = np.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype="float32")
for i, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
    for t, char in enumerate(input_text):
        encoder_input_data[i, t, input_token_index[char]] = 1.0
    for t, char in enumerate(target_text):
        decoder_input_data[i, t, target_token_index[char]] = 1.0
        if t > 0:
            decoder_target_data[i, t - 1, target_token_index[char]] = 1.0
# Encoder
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
encoder_states = [state_h, state_c]
# Decoder
decoder_inputs = Input(shape=(None, num_decoder_tokens))
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation="softmax")
decoder_outputs = decoder_dense(decoder_outputs)
# Model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer="rmsprop", loss="categorical_crossentropy",
metrics=["accuracy"])
```

```
model.fit(
    [encoder_input_data, decoder_input_data],
    decoder_target_data,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,)
# Save model
model.save("eng2french_seq2seq.h5")
# Inference setup
reverse_input_char_index = dict((i, char) for char, i in input_token_index.items())
reverse_target_char_index = dict((i, char) for char, i in target_token_index.items())
encoder_model = Model(encoder_inputs, encoder_states)
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
decoder_outputs, state_h, state_c = decoder_lstm(
    decoder_inputs, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs] + decoder_states)
# Decode function
def decode_sequence(input_seq):
    states_value = encoder_model.predict(input_seq)
    target_seq = np.zeros((1, 1, num_decoder_tokens))
    target_seq[0, 0, target_token_index["\t"]] = 1.0
    stop_condition = False
    decoded_sentence = ""
```

```
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char

        if sampled_char == "\n" or len(decoded_sentence) > max_decoder_seq_length:
            stop_condition = True
        target_seq = np.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.0
        states_value = [h, c]
    return decoded_sentence
# Test on few samples
for seq_index in range(5):
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    decoded_sentence = decode_sequence(input_seq)
    print("Input:", input_texts[seq_index])
    print("Predicted translation:", decoded_sentence.strip())
```

OUTPUT:



```
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 189ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
Input: Go.
Predicted translation: eae
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Input: Go.
Predicted translation: eae
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Input: Go.
Predicted translation: eae
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
Input: Hi.
Predicted translation: Aeree
```

RESULT:

PROGRAM:

```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.utils import load_img, img_to_array, to_categorical
from tensorflow.keras.applications import ResNet50V2 # Corrected import
from tensorflow.keras.applications.resnet_v2 import preprocess_input # Import
preprocess_input specifically
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout, Rescaling, RandomFlip,
RandomRotation, RandomTranslation, RandomZoom
from sklearn.metrics import confusion_matrix
# Load images from a given path and assign a label
def load_images_from_path(path, label):
    images = []
    labels = []
    for file in os.listdir(path):
        file_path = os.path.join(path, file)
        if os.path.isfile(file_path):  # Check if it's a file
            img = load_img(file_path, target_size=(224, 224))
            images.append(img_to_array(img))
            labels.append(label)
    return images, labels
# Show sample images
def show_images(images, title):
    fig, axes = plt.subplots(1, 8, figsize=(20, 20), subplot_kw={'xticks': [], 'yticks': []})
    fig.suptitle(title)
    for i, ax in enumerate(axes.flat):
        ax.imshow(images[i] / 255.)
# Data loading
```

```
x_train, y_train, x_test, y_test = [], [], [], []
# Train Data
img, lab = load_images_from_path('train/arctic_fox', 0)
x_train += img; y_train += lab
show_images(img, "Arctic Fox")
img, lab = load_images_from_path('train/polar_bear', 1)
x_train += img; y_train += lab
show_images(img, "Polar Bear")
img, lab = load_images_from_path('train/walrus', 2)
x_train += img; y_train += lab
show_images(img, "Walrus")
# Test Data
img, lab = load_images_from_path('test/arctic_fox', 0)
x_test += img; y_test += lab
img, lab = load_images_from_path('test/polar_bear', 1)
x_test += img; y_test += lab
img, lab = load_images_from_path('test/walrus', 2)
x_test += img; y_test += lab
# Convert and preprocess
x_train = preprocess_input(np.array(x_train))
x_test = preprocess_input(np.array(x_test))
y_train_encoded = to_categorical(y_train, num_classes=3)
y_test_encoded = to_categorical(y_test, num_classes=3)
# Load pretrained ResNet50V2 base
base_model = ResNet50V2(weights='imagenet', include_top=False, input_shape=(224, 224,
3))
base_model.trainable = False  # Freeze base
# Build the final model
model = Sequential([
    Rescaling(1./255),
    RandomFlip("horizontal"),
```

```
    RandomRotation(0.2),
    RandomTranslation(0.1, 0.1),
    RandomZoom(0.2),
    base_model,
    Flatten(),
    Dense(1024, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(
    x_train, y_train_encoded,
    validation_data=(x_test, y_test_encoded),
    epochs=10, batch_size=10)
# Plot accuracy graph
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training vs Validation Accuracy')
plt.show()
# Confusion matrix
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test_encoded, axis=1)
conf_matrix = confusion_matrix(y_true, y_pred_classes)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Arctic Fox', 'Polar Bear', 'Walrus'],
            yticklabels=['Arctic Fox', 'Polar Bear', 'Walrus'])
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Confusion Matrix")
plt.show()
# Predict Arctic Fox sample
sample_path = 'samples/arctic_fox/arctic_fox_140.jpeg'
img = load_img(sample_path, target_size=(224, 224))
plt.imshow(img); plt.axis('off'); plt.title('Prediction: Arctic Fox'); plt.show()
x = img_to_array(img)
x = preprocess_input(np.expand_dims(x, axis=0))
pred = model.predict(x)[0]
print("Sample Arctic Fox Prediction:")
for label, p in zip(['Arctic Fox', 'Polar Bear', 'Walrus'], pred):
    print(f"{label}: {p:.4f}")
# Predict Walrus sample
sample_path = 'samples/walrus/walrus_143.jpeg'
img = load_img(sample_path, target_size=(224, 224))
plt.imshow(img); plt.axis('off'); plt.title('Prediction: Walrus'); plt.show()
x = img_to_array(img)
x = preprocess_input(np.expand_dims(x, axis=0))
pred = model.predict(x)[0]
print("Sample Walrus Prediction:")
for label, p in zip(['Arctic Fox', 'Polar Bear', 'Walrus'], pred):
    print(f"{label}: {p:.4f}")
```

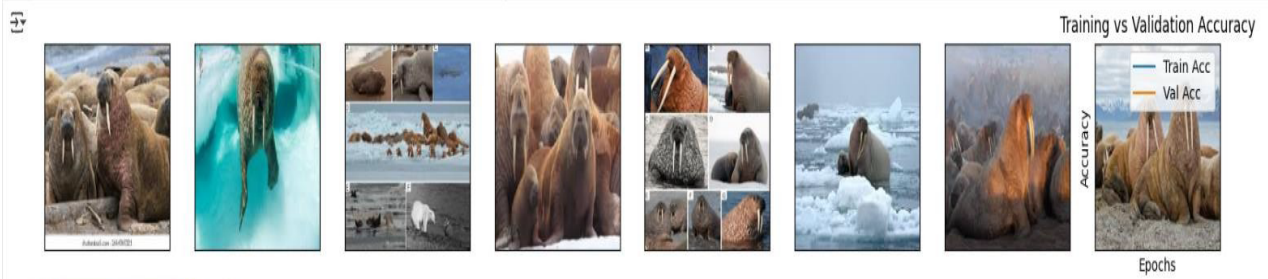
OUTPUT:

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 75s 7s/step - accuracy: 0.4389 - loss: 6.5543 - val_accuracy: 0.4474 - val_loss: 13.5735
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 39s 7s/step - accuracy: 0.4933 - loss: 11.1633 - val_accuracy: 0.3684 - val_loss: 8.0306
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 38s 6s/step - accuracy: 0.4622 - loss: 7.5413 - val_accuracy: 0.4474 - val_loss: 3.9444
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 42s 7s/step - accuracy: 0.5544 - loss: 3.1411 - val_accuracy: 0.3947 - val_loss: 3.5779
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 43s 7s/step - accuracy: 0.6500 - loss: 2.1115 - val_accuracy: 0.5263 - val_loss: 4.2944
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 36s 6s/step - accuracy: 0.4867 - loss: 3.1163 - val_accuracy: 0.6316 - val_loss: 1.9078
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 23s 6s/step - accuracy: 0.5056 - loss: 3.3300 - val_accuracy: 0.4737 - val_loss: 2.6970
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 44s 7s/step - accuracy: 0.5933 - loss: 1.5283 - val_accuracy: 0.6053 - val_loss: 3.6558
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 41s 7s/step - accuracy: 0.6256 - loss: 3.2951 - val_accuracy: 0.5263 - val_loss: 2.4695
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 25s 7s/step - accuracy: 0.5533 - loss: 2.7452 - val_accuracy: 0.6053 - val_loss: 1.5638
```

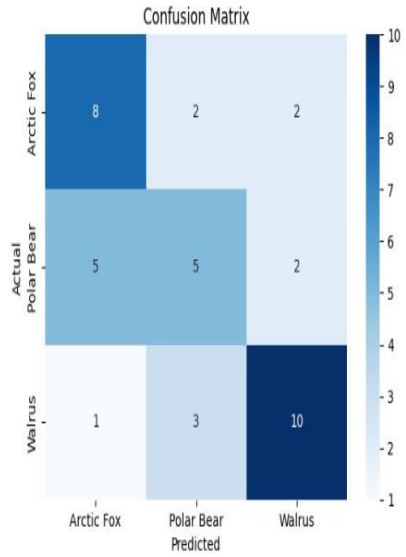
Arctic Fox

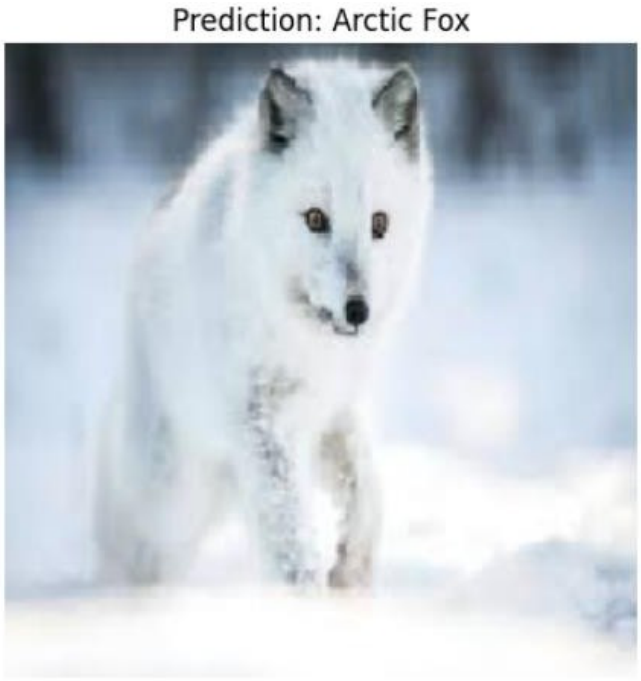
Polar Bear

Walrus




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step
```






```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step
Sample Arctic Fox Prediction:
Arctic Fox: 0.9932
Polar Bear: 0.0068
Walrus: 0.0000
```

Prediction: Walrus

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 309ms/step
Sample Walrus Prediction:
Arctic Fox: 0.9168
Polar Bear: 0.0795
Walrus: 0.0037
```

RESULT:

EXP.NO: 09 DATE:	**Medicinal Plants Identification and Classification using VGG16**

ABSTRACT:

This project focuses on developing a deep learning-based system for the identification and classification of medicinal plants using image classification techniques. Leveraging the visual features of plant leaves—such as shape, color, and texture—a model was trained to recognize and differentiate between species. The model uses the VGG16 architecture with transfer learning to efficiently extract patterns from high-resolution plant images. The system achieved high classification accuracy and was robust across varied environmental conditions like lighting and background. This model can assist users, including researchers and herbalists, in identifying medicinal plants with ease. Moreover, the deployment on a mobile-friendly platform allows real-time identification, offering a practical and accessible tool for the general public. The application of this model can aid in conservation efforts, traditional medicine, and educational initiatives.

PROBLEM STATEMENT:

Accurate identification of medicinal plants plays a vital role in traditional healthcare, yet it is often challenged by the need for expert botanical knowledge and the visual similarity among species. This project addresses the issue by developing an automated image processing system that leverages machine learning to identify medicinal plants from photographs. Through a user-friendly interface that allows users to capture or upload images, the tool aims to provide instant identification and relevant medicinal information, making plant recognition more accessible and promoting the safe use of herbal resources.

OBJECTIVE:

The primary goals of this study are as follows:

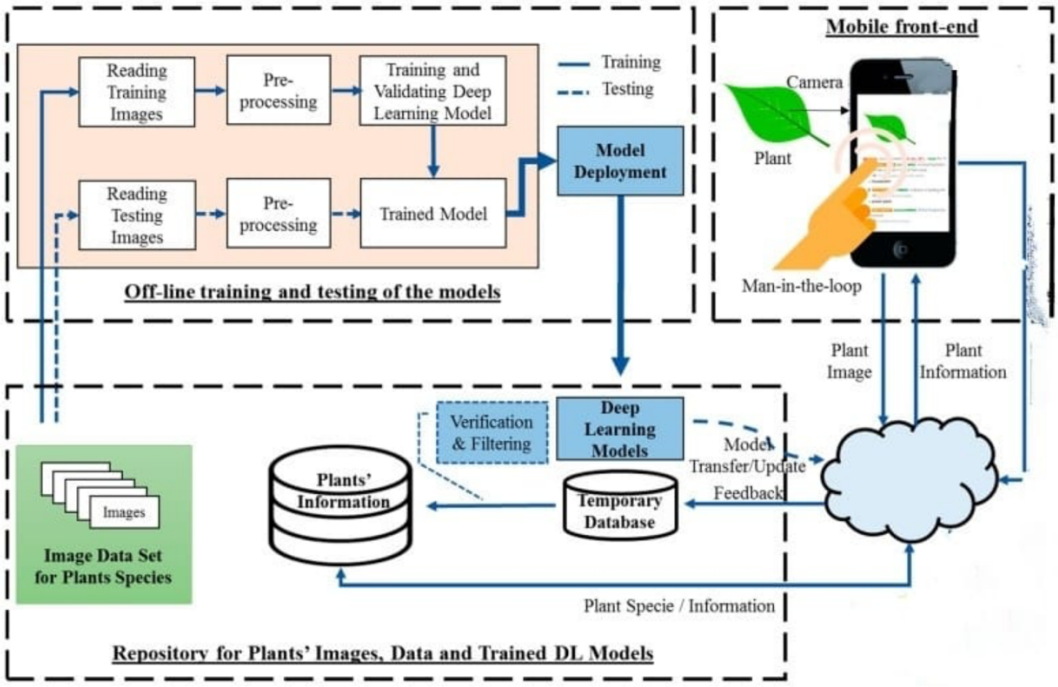
- To develop a machine learning-based image processing model capable of accurately identifying medicinal plants from the photographs.
- To analyze the effectiveness of different image classification algorithms for distinguishing between visually similar medicinal plant species.
- To evaluate the model's accuracy across varying environmental conditions, such as lighting, background, and image quality.
- To create a user-friendly interface that allows users to upload or capture images for plant identification, providing information on the medicinal properties of identified plants.
- To contribute to the accessibility of plant identification, supporting safer and more effective use of herbal remedies.
- To develop a machine learning-based image processing model.

PROPOSED SOLUTION:

To address the challenge of accurately identifying medicinal plants without expert botanical knowledge, this project proposes a deep learning-based image classification system using Convolutional Neural Networks (CNNs). Specifically, the solution uses the **VGG16 architecture with transfer learning**, enabling efficient training with limited labeled data. The model is trained on a well-structured dataset of medicinal plant images, focusing on features like leaf shape, color, and texture. Preprocessing techniques such as image resizing, normalization, and augmentation enhance model robustness. Once trained, the model is integrated into a **Streamlit-based web application** that allows users to upload or capture plant images for real-time identification. The app returns the predicted plant species along with detailed medicinal information. This system offers a scalable, accessible, and efficient tool for researchers, herbal practitioners, and the public to identify plants in diverse environments, improving safety and awareness in herbal medicine usage.

METHODOLOGY:

The methodology follows a structured pipeline: data collection, preprocessing, model training, evaluation, and deployment. A dataset with 30 medicinal plant species was sourced from Kaggle, split into training, validation, and test sets. Images were resized to 224x224 pixels to match the VGG16 input requirement. Data augmentation techniques such as rotation, flipping, and brightness adjustments were applied to improve generalization. Transfer learning using VGG16 allowed the use of pretrained weights from ImageNet, saving time and improving accuracy. The model was fine-tuned on the custom dataset. Metrics like accuracy, precision, recall, and F1-score were used to evaluate performance. A confusion matrix was generated to visualize misclassifications. After achieving satisfactory results, the model was integrated with a web interface using Streamlit for real-time plant identification.

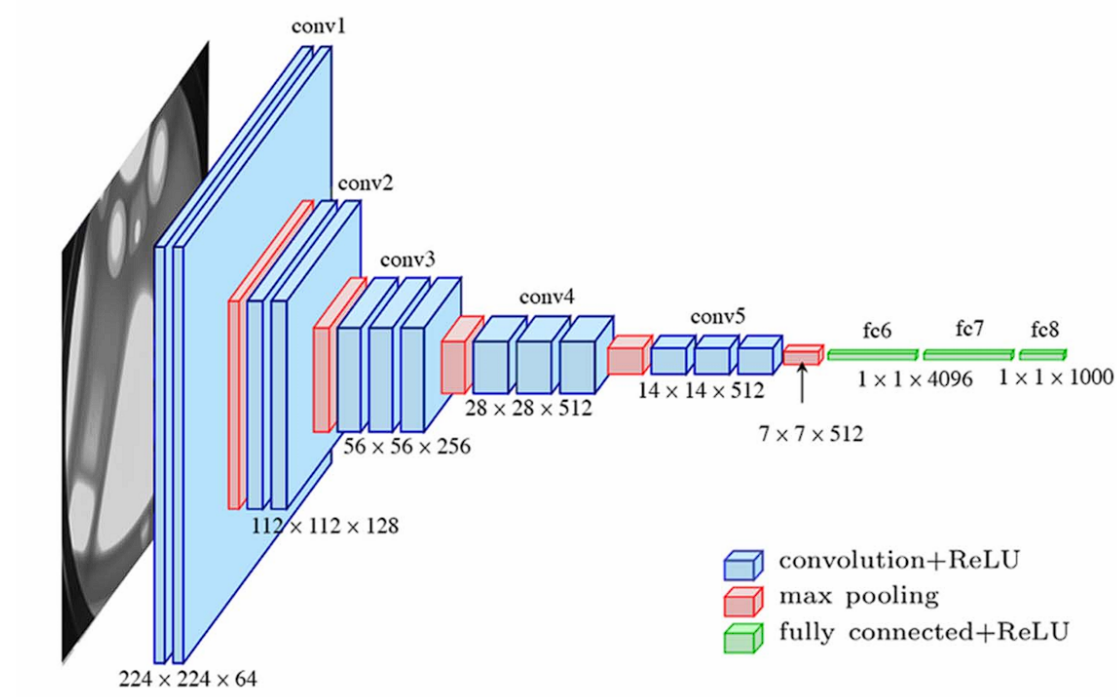


SYSTEM ARCHITECTURE:

The model uses the **VGG16 architecture**, a 16-layer Convolutional Neural Network known for its simplicity and accuracy in image classification tasks. It includes:

- 13 Convolutional layers with 3x3 filters to extract fine-grained features like leaf texture.
- 5 Max-Pooling layers for downsampling.
- 3 Fully Connected (Dense) layers for classification.
- A final Softmax layer for multiclass probability prediction.

Instead of training from scratch, **transfer learning** was applied using pretrained ImageNet weights. The top layers were replaced to fit the plant dataset, and the convolutional base was fine-tuned for better accuracy.



TECHNOLOGIES USED:

- **Python**: Core programming language used for preprocessing, training, and deployment.
- **TensorFlow/Keras**: Used to build and train the Convolutional Neural Network (CNN), specifically the VGG16 model.
- **OpenCV**: Employed for basic image processing and augmentation.
- **NumPy & Pandas**: For data manipulation and handling labels.
- **Matplotlib/Seaborn**: Used for plotting training curves, confusion matrices, and performance metrics.
- **Streamlit**: Framework used for deploying the model with a clean, interactive web interface.
- **Google Colab**: Utilized for model training due to its GPU support and ease of use.

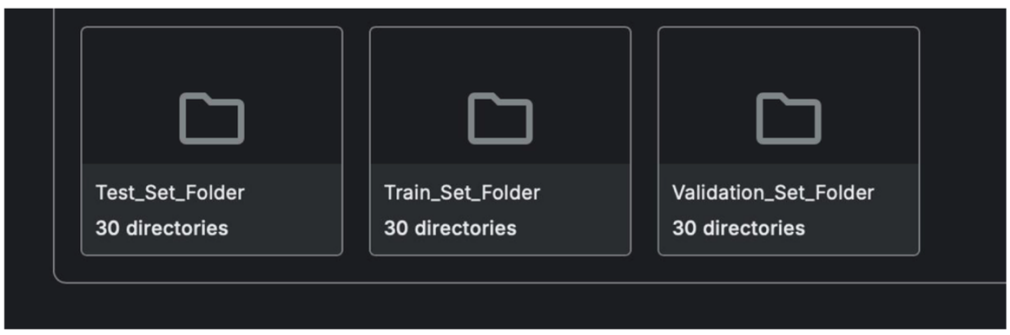
- **Cloud Storage/Database (Optional)**: For storing plant images and corresponding data in a scalable format.

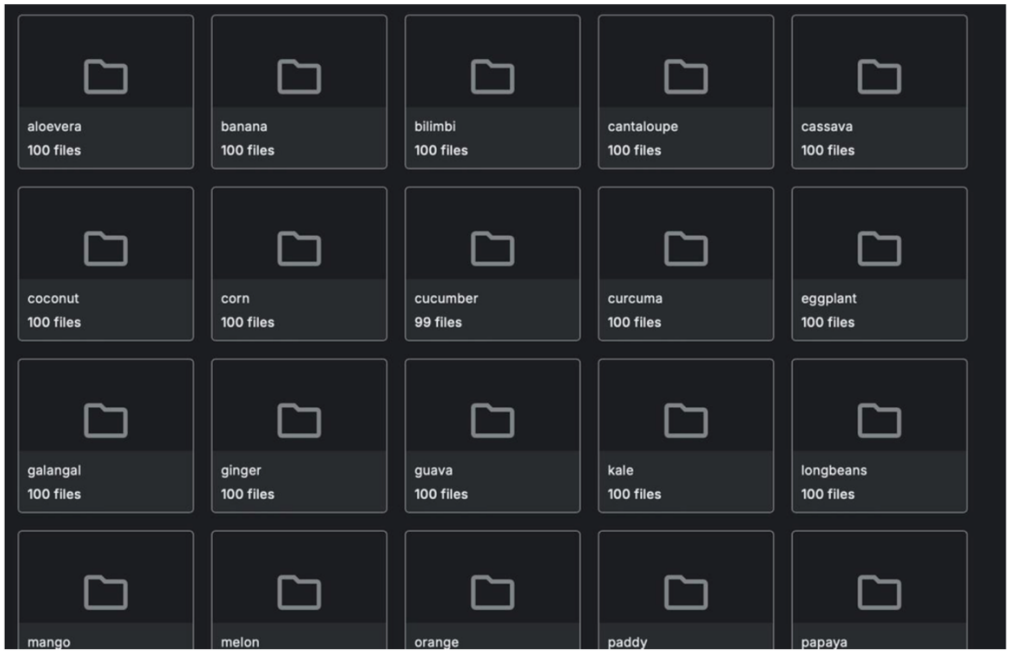
FEATURES IMPLEMENTED:

- **Automated Plant Identification**: Upload or capture an image to classify the plant.
- **Medicinal Information**: Provides common uses, botanical name, and medicinal properties.
- **Image Preprocessing**: Resize, normalize, and augment images before classification.
- **User Feedback Mechanism**: Option to confirm/correct predictions to improve future accuracy.
- **Cloud-Based Storage (Optional)**: Integration with cloud repositories for managing plant data and images.
- **Responsive UI**: Streamlit-based interface for web access on desktop and mobile.
- **Confusion Matrix Display**: Shows how well the model distinguishes between plant classes.
- **Live Demo**: Host the model and test predictions through a browser.

DATASET:

- **Source**: Kaggle (Medicinal Plants Dataset)
- **Size**: Images of 30 medicinal plant species
- **Structure**: Divided into train, test, and validation folders
- **Data Type**: High-resolution leaf, flower, and stem images
- **Preprocessing**:
  - Resizing to 224x224 pixels
  - Normalization (pixel scaling between 0–1)
  - Data augmentation (rotation, flip, brightness)
- **Labels**: Plant names corresponding to folders
- The dataset enabled robust training and testing across species with visually similar characteristics.





TRAINING AND VALIDATION:

The model was trained using 10 epochs on the preprocessed training dataset. The Adam optimizer and categorical cross-entropy loss were used. Training accuracy reached **99.61%**, indicating excellent learning. The validation dataset helped prevent overfitting by monitoring loss and accuracy at each epoch. Early stopping and checkpoint callbacks were added to retain the best-performing model. Data augmentation and dropout layers helped generalize the model to unseen images. Accuracy and loss curves were plotted to ensure a stable training process. The final model was tested using an independent test set to evaluate real-world performance.

```
234/234 ━━━━━━━━━━━━━━━━━━━━ 434s 2s/step - accuracy: 0.7420 - loss: 0.7578 - val_accuracy: 0.4762 - val_loss: 1.6178
Epoch 2/10
234/234 ━━━━━━━━━━━━━━━━━━━━ 396s 2s/step - accuracy: 0.9697 - loss: 0.0931 - val_accuracy: 0.7619 - val_loss: 0.8324
Epoch 3/10
234/234 ━━━━━━━━━━━━━━━━━━━━ 429s 2s/step - accuracy: 0.9854 - loss: 0.0419 - val_accuracy: 0.7143 - val_loss: 0.9928
Epoch 4/10
234/234 ━━━━━━━━━━━━━━━━━━━━ 430s 2s/step - accuracy: 0.9904 - loss: 0.0264 - val_accuracy: 0.6667 - val_loss: 1.6227
Epoch 5/10
234/234 ━━━━━━━━━━━━━━━━━━━━ 1077s 5s/step - accuracy: 0.9920 - loss: 0.0274 - val_accuracy: 0.7619 - val_loss: 1.2909
Epoch 6/10
234/234 ━━━━━━━━━━━━━━━━━━━━ 370s 2s/step - accuracy: 0.9944 - loss: 0.0209 - val_accuracy: 0.7143 - val_loss: 1.8922
Epoch 7/10
234/234 ━━━━━━━━━━━━━━━━━━━━ 388s 2s/step - accuracy: 0.9912 - loss: 0.0288 - val_accuracy: 0.7619 - val_loss: 0.9814
Epoch 8/10
234/234 ━━━━━━━━━━━━━━━━━━━━ 385s 2s/step - accuracy: 0.9950 - loss: 0.0129 - val_accuracy: 0.7619 - val_loss: 1.3934
Epoch 9/10
234/234 ━━━━━━━━━━━━━━━━━━━━ 410s 2s/step - accuracy: 0.9946 - loss: 0.0152 - val_accuracy: 0.8095 - val_loss: 0.8091
Epoch 10/10
234/234 ━━━━━━━━━━━━━━━━━━━━ 417s 2s/step - accuracy: 0.9955 - loss: 0.0144 - val_accuracy: 0.8571 - val_loss: 0.8159
```

RESULT AND DISCUSSION:

The model achieved:

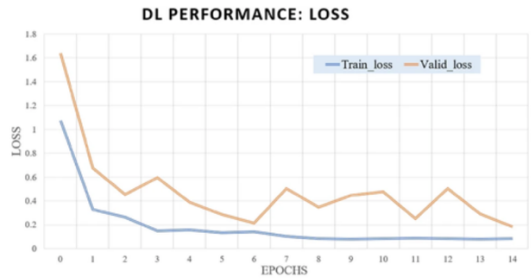
- **Training Accuracy**: 99.61%
- **Recall on Test Set**: 85.71%
- **Precision & F1-Score**: High for most classes, showing reliable classification
- **Confusion Matrix**: Most species were correctly classified; a few visually similar species showed occasional misclassification
- **Robustness**: Performed well under varied lighting and complex backgrounds

These results show that deep learning, especially with transfer learning (VGG16), can effectively classify medicinal plants with high accuracy. The gap between training and test accuracy indicates minor overfitting, which can be addressed with more data or fine-tuning. The model is well-suited for real-world deployment in mobile and web apps.

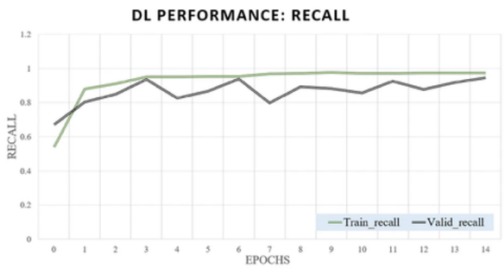
```
print("Train Accuracy  : {:.2f} %".format(history.history['accuracy'][-1]*100))
print("Test Accuracy   : {:.2f} %".format(accuracy_score(labels, predictions) * 100))
print("Precision Score : {:.2f} %".format(precision_score(labels, predictions, average='micro') * 100))
print("Recall Score    : {:.2f} %".format(recall_score(labels, predictions, average='micro') * 100))
```

```
Train Accuracy  : 99.61 %
Test Accuracy   : 85.71 %
Precision Score : 85.71 %
Recall Score    : 85.71 %
```

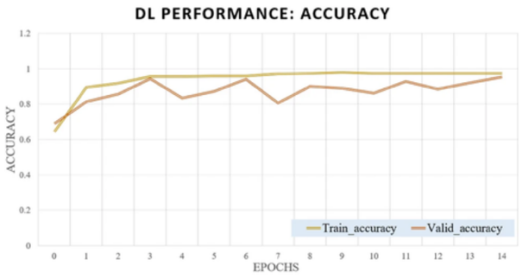
Performance Graphs:



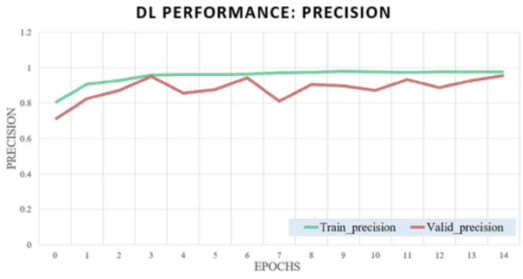
a)



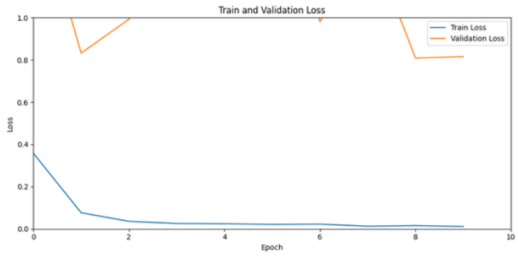
b)

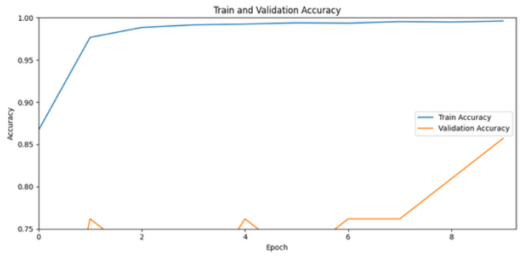


c)



d)





DEPLOYMENT (STREAMLIT WEB APP):

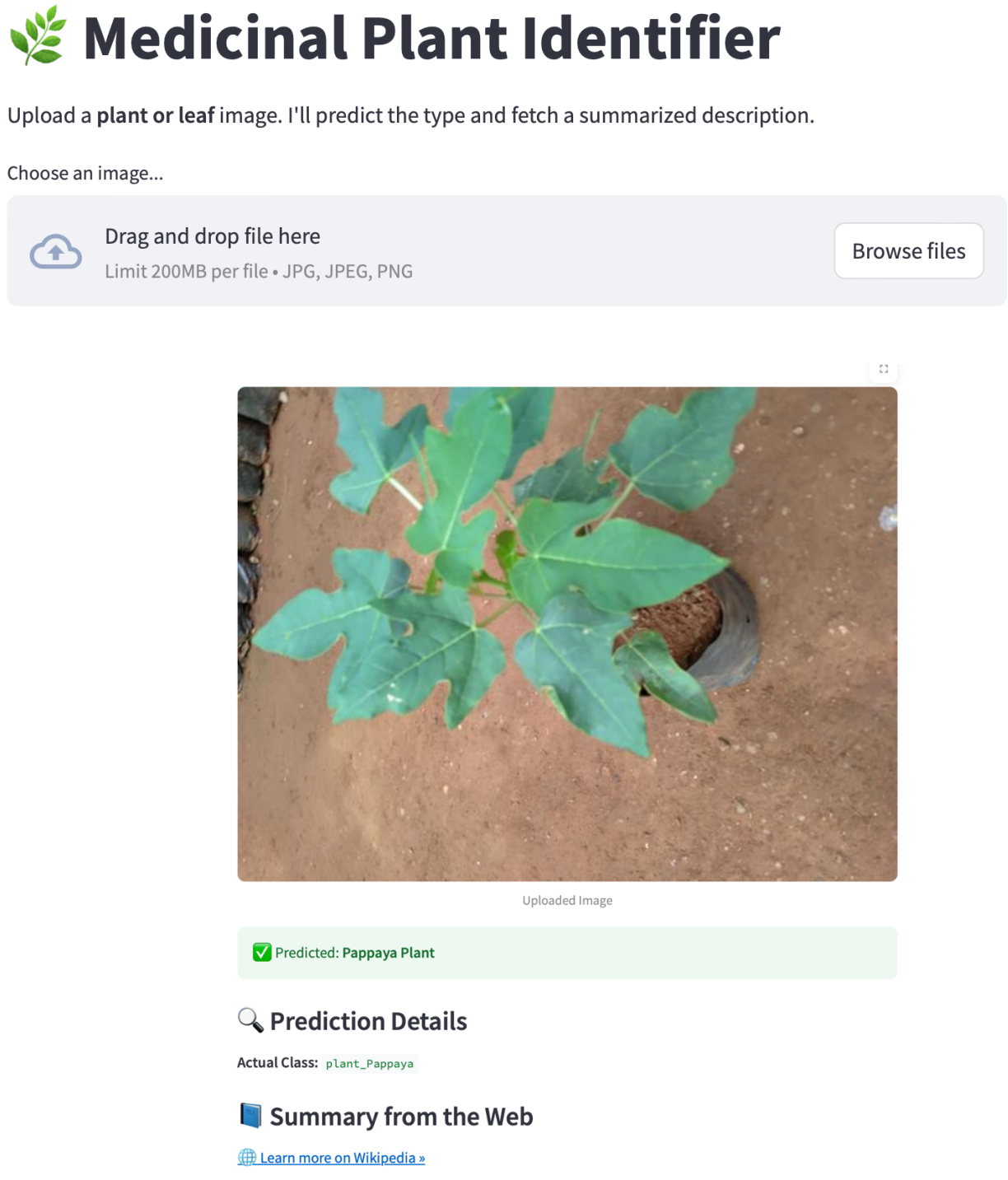
The trained model was deployed using **Streamlit**, a Python-based tool that allows the creation of interactive web applications with minimal code. Users can upload or capture images of medicinal plants via the web interface. The app processes the image and displays the predicted species along with its medicinal uses.

Key deployment features include:

- Instant predictions with confidence scores
- Clean UI with sidebar for image upload
- Output visualization including plant name and description
- Lightweight and can run on any browser
- Hosted publicly on Streamlit Cloud for easy access

Link (example): https://your-streamlit-app.streamlit.app

This makes the tool practical for researchers, students, and the general public.



CONCLUSION:

The project demonstrates that deep learning, specifically the VGG16 architecture with transfer learning, can effectively classify medicinal plants based on leaf images. The model achieved high training and test accuracy, proving its ability to distinguish between species with similar visual features. Its deployment via Streamlit enhances accessibility, allowing real-time plant identification. This application is particularly useful for botanists, herbalists, educators, and conservationists. Although the model shows great promise, there is scope for further refinement—particularly in improving test recall, expanding the dataset, and enhancing real-world usability under diverse environmental conditions.

FUTURE SCOPE:

1. **Offline Capability**: Optimize model for mobile offline use.
2. **Real-Time Camera Feed Classification**: Live video-based identification.
3. **Disease Detection**: Detect plant health issues alongside classification.
4. **AR Integration**: Augmented reality overlays showing medicinal information.
5. **Multi-language Support**: Regional language descriptions for inclusivity.
6. **Community Feedback & Crowdsourcing**: Continuous learning from user corrections.
7. **IoT Integration**: Track environmental data for plant conservation.
8. **Expanded Dataset**: Include more species and environmental conditions.

REFERENCES :

Plant Dataset : https://www.kaggle.com/datasets/yudhaislamisulistya/plants- type-datasets

[1]. Chanyal H, Yadav RK, Saini DKJ. Classifcation of medicinal plants leaves using deep learning technique: a review. Int J Intell Syst Appl Eng. 2022;10(4):78–87.

[2]. Javid A, Haghirosadat BF. A review of medicinal plants effective in the treatment or apoptosis of cancer cells. Cancer Press J. 2017;3(1):22–6.

[3]. Barimah KB, Akotia CS. The promotion of traditional medicine as enactment of community psychology in Ghana. J Community Psychol. 2015;43(1):99–106.

[4]. Rao RU, Lahari MS, Sri KP, Srujana KY, Yaswanth D. Identifcation of medicinal plants using deep learning. Int J Res Appl Sci Eng Technol. 2022;10:306–22.

[5]. Singh V, Misra AK. Detection of plant leaf diseases using image segmentation and soft computing techniques. Inf Process Agric. 2017;4(1):41–9.

[6]. Malik OA, Ismail N, Hussein BR, Yahya U. Automated real time identifcation of medicinal plants species in natural environment using deep learning models—a case study from Borneo Region. Plants. 2022;11(15):1952.

[7]. Valdez DB, Aliac CJG, Feliscuzo LS. Medicinal plant classifcation using convolutional neural network and transfer learning. In: 2022 IEEE International Conference on Artifcial Intelligence in Engineering and Technology (IICAIET). IEEE; 2022.p. 1–6.

[8]. Rajani s, Veena “Study on Identification and Classification of Medicinal Plants” International Journal of Advances in Science Engineering and Technology.

[9]. Nayana G. Gavhale, Dr. A. P. Thakare “Medicinal Plant Identification Using Image Processing Technique” Sipna College of Engineering and Technology Amravati

[10]. Shweta Srivastava, “Weka: A Tool for Data pre-processing, Classification, Ensemble, Clustering and Association Rule Mining”, International Journal of Computer Applications (0975 - 8887), vol. 88 no.10, Feb. 2014

[11]. Pradhan, A., Joshi, R., Sharma, S., & Rai, R. (2024). *Evaluating deep CNN models for medicinal plant leaf classification*. *Heliyon*, 10(1), e16322.

[12]. Yusof, S. M., Jamil, Z., & Rauf, H. A. (2023). *Ensemble learning model for medicinal plant classification using VGG19 and DenseNet201*. *Information*, 14(11), 618.

[13]. Sharma, A., Mishra, R., & Kale, P. (2025). *Herbify: A framework for regional medicinal plant identification using hybrid vision transformers*. *Plant Methods*, 21(1), 55.

[14]. Ghimire, D., Basnet, D., & Shrestha, P. (2025). *Identification of Nepalese medicinal herbs using deep convolutional neural networks*. *arXiv preprint*.

[15]. Vaidyanathan, R., & Narayanan, A. (2024). *CNN-X for large-scale medicinal plant classification on VNPlant-200 dataset*. *Journal of Engineering Science*, 20(2), 115–126.