ASSIGNMENT – 2

You are designing a CRNN to recognize handwritten Tamil characters.

How would you process the input images and structure your CRNN model?

Introduction

- **Background**: Briefly introduce the concept of Optical Character Recognition (OCR) and its importance in recognizing handwritten text.
- **Objective**: Describe the purpose of this project, which is to build a model capable of recognizing handwritten Tamil characters using a Convolutional Recurrent Neural Network (CRNN).
- **Model Choice**: Explain why the CRNN model is suitable for this task, emphasizing its ability to handle sequential data and capture spatial dependencies.

---

2. Dataset Preparation

- **Data Collection**: Explain the source of your handwritten Tamil character images or if you're using a public dataset.
- **Dataset Structure**: Detail the folder and file structure. For example:
    - Organize images into folders named after the character they represent (e.g., tamil_character_1, tamil_character_2, etc.).

   - Use a CSV file to map each image file to its corresponding label.

- **Image Preprocessing**: Describe any preprocessing steps applied, such as resizing, grayscale conversion, or normalization. Mention any specific dimensions (e.g., 32x128 pixels).
- **Labeling**: Explain how you labeled the images, either through folder names or a CSV file, and ensure labels correspond to unique Tamil characters.

---

3. Model Architecture

- **Convolutional Layers**: Describe the convolutional layers' role in extracting spatial features from the image input, noting the layers used and any significant details like filter size and stride.
- **Recurrent Layers**: Explain the use of recurrent layers (e.g., LSTM or GRU) to capture sequential patterns in the data.
- **CTC (Connectionist Temporal Classification) Loss**: Describe how CTC loss is used in training the model to handle variable-length output sequences. This is critical for character recognition tasks where character sequences are not aligned to specific positions in the image.
- **Code Explanation**: Provide a high-level breakdown of the code in crnn_model.py. Mention the purpose of each main section, such as defining layers, compiling the model, and the function for CTC loss.

---

4. Training and Evaluation

- **Training Setup**: Outline the training parameters (e.g., batch size, learning rate, optimizer) and hardware used (e.g., GPU if applicable).

- **Model Compilation**: Describe how the model was compiled, specifying the loss function (CTC loss) and evaluation metrics.
- **Evaluation Metrics**: Explain the metrics used to evaluate the model, such as accuracy or word error rate (WER), and why these metrics are relevant for OCR.
- **Training Process**: Summarize the training process, including the number of epochs and any augmentation techniques used to improve model generalization.

---

5. Results and Observations

- **Model Performance**: Report the model's performance on the test set, highlighting accuracy, WER, or other relevant metrics.
- **Examples**: Include a few examples of the model's predictions versus ground truth for handwritten Tamil characters.
- **Challenges**: Mention any challenges encountered, such as overfitting, noisy data, or difficulties in character differentiation.
- **Observations**: Discuss any observations, such as patterns in characters that were easier or harder for the model to recognize.

CODING :

```
import cv2
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.models import Model
```

```
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Reshape, LSTM,
Bidirectional, Dense, Dropout, GlobalAveragePooling1D
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split  # Import for splitting dataset
# Function to preprocess images
def preprocess_image(image_path, target_height=32, target_width=128):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img_resized = cv2.resize(img, (target_width, target_height))
    img_normalized = img_resized / 255.0
    img_reshaped = np.expand_dims(img_normalized, axis=-1)  # Add the channel dimension
    return img_reshaped
# Load dataset function
def load_dataset(data_dir, label_file, image_size=(32, 128)):
images = []
labels = []
with open(label_file, 'r', encoding='utf-8') as f:
for line in f:
img_name, label = line.strip().split(',')
img_path = os.path.join(data_dir, img_name)
img=preprocess_image(img_path,
target_height=image_size[0], target_width=image_size[1])
images.append(img)
labels.append(label)  # Store character labels as-is
return np.array(images), labels
# CRNN model architecture
def CRNN_Model(input_shape, num_classes):
    input_img = Input(shape=input_shape)

    # Convolutional layers to extract features from images
```

```
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    # Reshape to 2D for LSTM layer
    x = Reshape(target_shape=(-1, 128))(x)
    # Bidirectional LSTM layer to capture temporal dependencies
    x = Bidirectional(LSTM(128, return_sequences=True))(x)
    x = Dropout(0.25)(x)
    # Use GlobalAveragePooling1D instead of GlobalAveragePooling2D
    x = GlobalAveragePooling1D()(x)  # Use GlobalAveragePooling1D after LSTM layer
    # Final dense layer to match the number of classes
    output = Dense(num_classes, activation='softmax')(x)
    # Create the model
    model = Model(inputs=input_img, outputs=output)
    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
# Load dataset
train_images, train_labels = load_dataset("data/images", "data/labels.csv")
# Encode labels
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
train_labels_categorical = to_categorical(train_labels_encoded)
# Ensure images are in the correct shape (add batch dimension)
train_images = np.array(train_images)
train_images = np.expand_dims(train_images, axis=-1)  # Adding channel dimension to
match (32, 128, 1)


# Split the dataset into training and testing sets
```

```
train_images, test_images, train_labels_categorical, test_labels_categorical =
train_test_split(
    train_images, train_labels_categorical, test_size=0.2, random_state=42
)
# Define model parameters
input_shape = (32, 128, 1)  # Image size 32x128 and 1 channel (grayscale)
num_classes = len(label_encoder.classes_)  # Number of unique characters
# Instantiate and compile the model
model = CRNN_Model(input_shape, num_classes)
# Train the model (with no validation split as the dataset is small)
model.fit(train_images, train_labels_categorical, epochs=5, batch_size=32)
# Save the model
model.save('crnn_model.h5')  # Save in .h5 format
# Load the saved model (using the same format)
model = tf.keras.models.load_model('crnn_model.h5')
# Evaluate on test data
test_loss, test_accuracy = model.evaluate(test_images, test_labels_categorical)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
# Function to predict and display the Tamil character
def predict_character(image_path, model, label_encoder):
    # Preprocess the input image
    img = preprocess_image(image_path)
    # Add batch dimension to the image (model expects batches)
    img = np.expand_dims(img, axis=0)  # Shape: (1, 32, 128, 1)
    # Predict the character index
    prediction = model.predict(img)
    # Get the predicted class index (the highest probability)
    predicted_class_idx = np.argmax(prediction, axis=-1)[0]

    # Map the predicted index back to the character label
```

```
 predicted_label = label_encoder.inverse_transform([predicted_class_idx])[0]
 # Display the predicted character
 print(f"Output: The predicted character is displayed: {predicted_label}")
# Example usage:
predict_character("data/images/image1.png", model, label_encoder)
```

OUTPUT:

```
Test Loss: 7.943552017211914
Test Accuracy: 0.0
```

```
Output: The predicted character is displayed: அ
```

Conclusion

- **Impact**: Briefly discuss the potential impact of this project, such as improving digital accessibility for Tamil speakers.
- **Limitations**: Note any limitations, like the need for a larger dataset or handling complex character structures.
- **Future Work**: Suggest improvements, such as using a larger dataset, fine-tuning hyperparameters, or integrating with other OCR technologies for more languages.