TASK 3: Handwritten Character Recognition

Objective: Identify handwritten characters or alphabets.

Approach: Use image processing and deep learning.

Key Features:

● Dataset: MNIST (digits), EMNIST (characters).

● Model: Convolutional Neural Networks (CNN).

● Extendable to full word or sentence recognition with sequence modeling (like CRNN).

```
# Install tensorflow_datasets
!pip install tensorflow_datasets

# Import libraries
import tensorflow_datasets as tfds
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: tensorflow_datasets in /usr/local/lib/python3.11/dist-packages (4.9.9)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (1.4.0)
Requirement already satisfied: array_record>=0.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (0.7.2)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (0.1.9)
Requirement already satisfied: etils>=1.9.1 in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1;
Requirement already satisfied: immutabledict in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (4.2.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (2.0.2)
Requirement already satisfied: promise in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (2.3)
Requirement already satisfied: protobuf>=3.20 in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (5.29.5)
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (5.9.5)
Requirement already satisfied: pyarrow in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (18.1.0)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (2.32.3)
Requirement already satisfied: simple_parsing in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (0.1.7)
Requirement already satisfied: tensorflow-metadata in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (1.17.2)
Requirement already satisfied: termcolor in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (3.1.0)
Requirement already satisfied: toml in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (0.10.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (4.67.1)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from tensorflow_datasets) (1.17.2)
Requirement already satisfied: einops in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; pythor
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; pythor
Requirement already satisfied: importlib_resources in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=
Requirement already satisfied: typing_extensions in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1
Requirement already satisfied: zipp in /usr/local/lib/python3.11/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_v
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19.0->tensorflc
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19.0->tensorflow_datasets)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19.0->tensorflow_data
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.19.0->tensorflow_data
Requirement already satisfied: attrs>=18.2.0 in /usr/local/lib/python3.11/dist-packages (from dm-tree->tensorflow_datasets) (25.3.0
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from promise->tensorflow_datasets) (1.17.0)
Requirement already satisfied: docstring-parser<1.0,>=0.15 in /usr/local/lib/python3.11/dist-packages (from simple_parsing->tensorfl
Requirement already satisfied: googleapis-common-protos<2,>=1.56.4 in /usr/local/lib/python3.11/dist-packages (from tensorflow-metac
```

```
# Load EMNIST Letters Dataset
(ds_train, ds_test), ds_info = tfds.load(
    'emnist/letters',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True
)
```

```
# Normalize & reshape
def preprocess(image, label):
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.expand_dims(image, -1)  # Add channel dimension
    label -= 1  # Convert from 1–26 to 0–25
    return image, label

ds_train = ds_train.map(preprocess).batch(128).prefetch(tf.data.AUTOTUNE)
ds_test = ds_test.map(preprocess).batch(128).prefetch(tf.data.AUTOTUNE)
```

```
# Define CNN
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(26, activation='softmax')
])

# Compile
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train
hist = model.fit(ds_train, epochs=5, validation_data=ds_test) # Assign the history to 'hist'
```

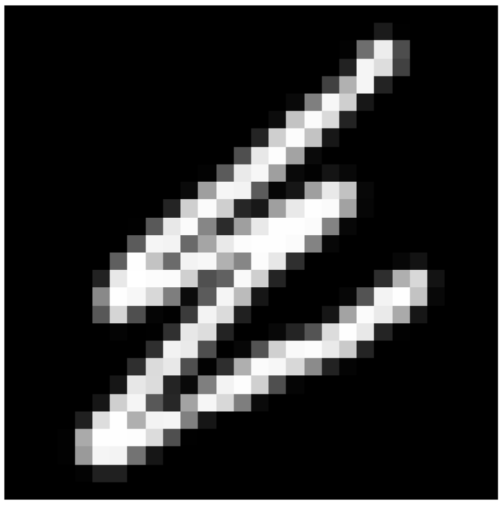
```
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
694/694 ━━━━━━━━━━━━━━━━━━━━ 8s 8ms/step - accuracy: 0.5526 - loss: 1.5050 - val_accuracy: 0.8756 - val_loss: 0.3944
Epoch 2/5
694/694 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.8324 - loss: 0.5256 - val_accuracy: 0.8947 - val_loss: 0.3154
Epoch 3/5
694/694 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8651 - loss: 0.4224 - val_accuracy: 0.9030 - val_loss: 0.2885
Epoch 4/5
694/694 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.8810 - loss: 0.3724 - val_accuracy: 0.9097 - val_loss: 0.2720
Epoch 5/5
694/694 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8931 - loss: 0.3352 - val_accuracy: 0.9106 - val_loss: 0.2651
```

```
# Show prediction
for images, labels in ds_test.take(1):
    image = images[0]
    true_label = labels[0].numpy()
    pred = model.predict(tf.expand_dims(image, 0))
    pred_label = np.argmax(pred)

    plt.imshow(tf.squeeze(image), cmap='gray')
    plt.title(f"Prediction: {chr(pred_label + 65)}, True: {chr(true_label + 65)}")
    plt.axis('off')
    plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 268ms/step
```

Prediction: M, True: M



```
# Save the Model
model.save("emnist_letters_model_tfds.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is
```

Extendable to full word or sentence recognition with sequence modeling (like CRNN).

Import Libraries

```
import tensorflow as tf
from tensorflow.keras import layers
import os
```

Define Character Set

```
characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
char_to_num = layers.StringLookup(vocabulary=list(characters), oov_token="*")
num_to_char = layers.StringLookup(vocabulary=char_to_num.get_vocabulary(), invert=True)
```

Prepare Dataset Function

```
def process_image(image_path, label):
    image = tf.io.read_file(image_path)
    image = tf.io.decode_png(image, channels=1)
    image = tf.image.resize(image, [32, 128])
    image = tf.cast(image, tf.float32) / 255.0
    return image, label

def prepare_dataset(images_dir, labels_file):
    lines = tf.io.read_file(labels_file).numpy().decode().split("\n")
    records = [line.split() for line in lines if len(line.strip().split()) >= 2]
    paths = [os.path.join(images_dir, r[0]) for r in records]
    texts = [r[1] for r in records]

    labels = [char_to_num(tf.strings.unicode_split(t, input_encoding="UTF-8")) for t in texts]

    path_ds = tf.data.Dataset.from_tensor_slices(paths)
    label_ds = tf.data.Dataset.from_tensor_slices(labels)
    ds = tf.data.Dataset.zip((path_ds, label_ds))
    ds = ds.map(process_image, num_parallel_calls=tf.data.AUTOTUNE)
    return ds.batch(32).prefetch(tf.data.AUTOTUNE)
```

CTC Loss Function

```
def ctc_loss_fn(y_true, y_pred):
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

    return tf.keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)
```

Build CRNN Model

```
inputs = tf.keras.Input(shape=(32, 128, 1))

x = layers.Conv2D(64, (3,3), activation='relu', padding='same')(inputs)
x = layers.MaxPooling2D((2,2))(x)
x = layers.Conv2D(128, (3,3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2,2))(x)
x = layers.Conv2D(256, (3,3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=(2,1))(x)

new_shape = x.shape
x = layers.Reshape((new_shape[1], new_shape[2] * new_shape[3]))(x)

x = layers.Bidirectional(layers.LSTM(128, return_sequences=True))(x)
x = layers.Bidirectional(layers.LSTM(128, return_sequences=True))(x)

vocab_size = len(char_to_num.get_vocabulary()) + 1  # CTC blank token
outputs = layers.Dense(vocab_size, activation='softmax')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer="adam", loss=ctc_loss_fn)

model.summary()
```

```
Model: "functional_4"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer_4 (InputLayer)      │ (None, 32, 128, 1)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_10 (Conv2D)              │ (None, 32, 128, 64)    │           640 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_10 (MaxPooling2D) │ (None, 16, 64, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_11 (Conv2D)              │ (None, 16, 64, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_11 (MaxPooling2D) │ (None, 8, 32, 128)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_12 (Conv2D)              │ (None, 8, 32, 256)     │       295,168 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_2           │ (None, 8, 32, 256)     │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_12 (MaxPooling2D) │ (None, 4, 32, 256)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ reshape_2 (Reshape)             │ (None, 4, 8192)        │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ bidirectional_4 (Bidirectional) │ (None, 4, 256)         │     8,520,704 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ bidirectional_5 (Bidirectional) │ (None, 4, 256)         │       394,240 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_6 (Dense)                 │ (None, 4, 64)          │        16,448 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 9,302,080 (35.48 MB)
 Trainable params: 9,301,568 (35.48 MB)
 Non-trainable params: 512 (2.00 KB)
```

Summary:

Part 1 is suitable for recognizing isolated characters using CNN.

Part 2 extends to sequence prediction like full words using CRNN + CTC Loss.