```python
import struct as st
import numpy as np
import tensorflow as tf
from tensorflow.keras import models, regularizers
from tensorflow.keras.layers import (
    Dense,
    Conv2D,
    Dropout,
    Flatten,
    MaxPooling2D,
    BatchNormalization,
)

SHUFFLE_SIZE = 1000
BATCH_SIZE = 32
TRAIN_SPLIT = 0.8
EPOCHS = 5

dataset = {
    "train-images": "../datasets/MNIST_DATASET/train-images-idx3-ubyte",
    "train-labels": "../datasets/MNIST_DATASET/train-labels-idx1-ubyte",
    "test-images": "../datasets/MNIST_DATASET/t10k-images-idx3-ubyte",
    "test-labels": "../datasets/MNIST_DATASET/t10k-labels-idx1-ubyte",
}


def parseDataset():
    images_set = []
    labels_set = []
    for k, v in dataset.items():
        if "images" in k:
            with open(v, "rb") as f:
                magic, num_imgs, num_rows, num_cols = st.unpack(">IIII", f.read(1
6))
                images_set.append(
                    np.fromfile(f, dtype=np.dtype(np.ubyte))
                    .newbyteorder(">")
                    .reshape(num_imgs, num_rows, num_cols, 1)
                )
        elif "labels" in k:
            with open(v, "rb") as f:
                magic, num_items = st.unpack(">II", f.read(8))
                labels_set.append(
                    np.fromfile(f, dtype=np.dtype(np.uint8)).newbyteorder(">")
                )

    return ((images_set[0], labels_set[0]), (images_set[1], labels_set[1]))


def main():

    # load data:
    # [0] => IMAGES
    # [1] => LABELS
    train, test = parseDataset()
    normalized_train, normalized_test = (train[0] / 255.0, train[1]), (
        test[0] / 255.0,
        test[1],
    )

    model = models.Sequential()
    model.add(
```

```python
        Conv2D(
            64,
            kernel_size=(3, 3),
            activation=tf.nn.elu,
            input_shape=(train[0][0].shape[0], train[0][0].shape[1], 1),
            kernel_regularizer=regularizers.l2(l2=0.01),
        )
    )
    model.add(MaxPooling2D())
    model.add(
        Conv2D(
            64,
            kernel_size=(3, 3),
            activation=tf.nn.elu,
            kernel_regularizer=regularizers.l2(l2=0.2),
        )
    )
    model.add(MaxPooling2D())
    model.add(Flatten())  # Flattening the 2D arrays for fully connected layers
    model.add(
        Dense(
            10, activation=tf.nn.leaky_relu, kernel_regularizer=regularizers.l2(
l2=0.05)
        )
    )
    model.add(Dropout(0.05))
    model.add(
        Dense(
            20, activation=tf.nn.leaky_relu, kernel_regularizer=regularizers.l2(
l2=0.3)
        )
    )
    model.add(BatchNormalization())
    model.add(Dropout(0.2))
    model.add(
        Dense(
            50, activation=tf.nn.leaky_relu, kernel_regularizer=regularizers.l2(
l2=0.3)
        )
    )
    model.add(BatchNormalization())
    model.add(Dense(10, activation="softmax", kernel_regularizer=regularizers.l2(
)))
    model.summary()

    model.compile(
        optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"]
    )
    model.fit(
        x=normalized_train[0],
        y=normalized_train[1],
        batch_size=BATCH_SIZE,
        epochs=EPOCHS,
        steps_per_epoch=np.math.ceil(0.8 * train[0].shape[0] / BATCH_SIZE),
        validation_split=0.2,
    )
    model.evaluate(x=normalized_test[0], y=normalized_test[1])


if __name__ == "__main__":
    main()
```