```python
#!/bin/env python3.8
from datetime import datetime
import numpy as np
import requests, gzip, os, hashlib
import tensorflow as tf


from tqdm import trange

rng = np.random.default_rng(seed=42)

# Combination of:
# https://www.tensorflow.org/datasets/api_docs/python/tfds/core/DatasetBuilder
# https://github.com/tensorflow/datasets/blob/master/tensorflow_datasets/image_c
lassification/mnist.pyclass MNIST_builder(tfds.core.DatasetBuilder):
class MNSIT:
    rng = None
    data_dir = ""
    X = None
    Y = None
    X_test = None
    Y_test = None

    def __init__(self, data_dir="data/", rng=np.random.default_rng(seed=42)):
        self.data_dir = data_dir
        self.rng = rng

    def fetch(self, url):
        fp = os.path.join(self.data_dir, hashlib.md5(url.encode("utf-8")).hexdige
st())
        if os.path.isfile(fp):
            with open(fp, "rb") as f:
                data = f.read()
        else:
            with open(fp, "wb") as f:
                data = requests.get(url).content
                f.write(data)
        return np.frombuffer(gzip.decompress(data), dtype=np.uint8).copy()

    def download_and_prepare(self, val_percent=0.2):
        self.X = self.fetch(
            "http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz"
        )[0x10:].reshape((-1, 28, 28))
        self.Y = self.fetch(
            "http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz"
        )[8:]

        choices = np.array(
            self.rng.choice(len(self.X), size=int(len(self.X) * val_percent))
        )

        self.X_val = self.X[choices]
        self.Y_val = self.Y[choices]

        inverse_choices = range(len(self.X))
        inverse_choices = np.delete(inverse_choices, choices)

        self.X = self.X[inverse_choices]
        self.Y = self.Y[inverse_choices]

        self.X_test = self.fetch(
            "http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz"
```

```python
        )[0x10:].reshape((-1, 28, 28))
        self.Y_test = self.fetch(
            "http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz"
        )[8:]

        print("X:", self.X.shape)
        print("Y:", self.Y.shape)
        print("X_TEST:", self.X_test.shape)
        print("Y_TEST:", self.Y_test.shape)

    def as_dataset(self, split, shuffle_files=False):
        if split == "train":
            return self.X, self.Y
        elif split == "test":
            return self.X_test, self.Y_test
        elif split == "val":
            return self.X_val, self.Y_val
        else:
            raise ValueError("Invalid split")

    def info(self):
        return {
            "name": "mnist",
            "version": "0.0.1",
            "description": "The MNIST database of handwritten digits.",
        }

    def get_batch(self, split, batch_size):
        X, Y = self.as_dataset(split)
        choices = np.array(self.rng.choice(len(X), size=batch_size))
        return X[choices], Y[choices]


class tensorboar_handler:

    train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy("train_accuracy")
    val_accuracy = tf.keras.metrics.SparseCategoricalAccuracy("val_accuracy")

    def __init__(self, log_dir="logs/"):
        current_time = datetime.now().strftime("%Y%m%d-%H%M%S")
        self.log_dir = log_dir

        self.train_writer = tf.summary.create_file_writer(
            self.log_dir + "/" + current_time + "/train"
        )
        self.val_writer = tf.summary.create_file_writer(
            self.log_dir + "/" + current_time + "/val"
        )

    def log(self, step, loss, accuracy, split="train"):
        if split == "train":
            writer = self.train_writer
            self.train_accuracy.reset_states()

        elif split == "val":
            writer = self.val_writer
            self.val_accuracy.reset_states()

        else:
            raise ValueError("Invalid split")

        with writer.as_default():
            tf.summary.scalar("loss", loss, step=step)
            tf.summary.scalar("accuracy", accuracy, step=step)

    def close(self):
        self.train_writer.close()
        self.val_writer.close()


print("Num GPUs Available: ", tf.config.list_physical_devices("GPU"))
tf.debugging.set_log_device_placement(True)


model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Input(shape=(28, 28, 1)))
model.add(tf.keras.layers.Conv2D(32, 7, activation="relu"))
model.add(tf.keras.layers.Conv2D(32, 5, activation="relu"))
model.add(tf.keras.layers.Conv2D(32, 3, activation="relu"))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation="relu"))
model.add(tf.keras.layers.Dense(10, activation="softmax"))

mnist = MNSIT(rng=rng)
mnist.download_and_prepare()

loss = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

board = tensorboar_handler("logs/")

bar = trange(5000)
for epoch in bar:
    with tf.GradientTape() as tape:
        x, y = mnist.get_batch("train", 256)
        y_hat = model(x)
        loss_value = loss(y, y_hat)
        board.log(epoch, loss_value, board.train_accuracy(y, y_hat), split="train
")

    grads = tape.gradient(loss_value, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    if epoch % 50 == 0:
        x, y = mnist.as_dataset("val")
        y_hat = model(x)
        board.log(epoch, loss(y, y_hat), board.val_accuracy(y, y_hat), split="va
l")

    bar.set_description(f"Loss @ {epoch} => {loss_value.numpy():0.8f}")
    bar.refresh()
```