```python
#!/usr/bin/env python

# ECE472-Samuel Maltz
# Assignment 3: Classification of MNIST dataset using convolutional neural netwo
rk

import tensorflow as tf
import numpy as np

from mnist import MNIST
from absl import app
from absl import flags

FLAGS = flags.FLAGS
flags.DEFINE_string("data_dir", "data", "Name of directory with datasets")
flags.DEFINE_list("conv_filters", [3, 2], "Number of filters of convolutional layers")
flags.DEFINE_list("dense_widths", [], "Widths of dense layers")
flags.DEFINE_float("dropout", 0.2, "Dropout rate")
flags.DEFINE_float("learning_rate", 0.001, "Learning rate for Adam optimizer")
flags.DEFINE_integer("epochs", 12, "Number of training epochs")
flags.DEFINE_float("val_split", 0.2, "Validation fraction")
flags.DEFINE_float("kernel_reg", 0.01, "Regularizer coefficient")
flags.DEFINE_integer("random_seed", 12345, "Random seed")


class Data(object):
    def __init__(self, data_dir):
        mndata = MNIST(data_dir)

        self.training_images, self.training_labels = mndata.load_training()
        self.testing_images, self.testing_labels = mndata.load_testing()

        self.training_images = self.preprocess_images(self.training_images)
        self.training_labels = np.array(self.training_labels)
        self.testing_images = self.preprocess_images(self.testing_images)
        self.testing_labels = np.array(self.testing_labels)

    def preprocess_images(self, images):
        return np.reshape(np.array(images), (len(images), 28, 28, 1)).astype("flo
at32")


class Model(tf.keras.Model):
    def __init__(self, conv_filters, dense_widths, dropout, kernel_reg):
        super().__init__()
        self.regularizer = tf.keras.regularizers.L2(kernel_reg)

        self.conv = [
            tf.keras.layers.Conv2D(i, 3, activation="relu") for i in conv_filters
        ]
        self.maxpool = tf.keras.layers.MaxPool2D()
        self.flatten = tf.keras.layers.Flatten()
        self.dense = [
            tf.keras.layers.Dense(
                i, activation="relu", kernel_regularizer=self.regularizer
            )
            for i in dense_widths
        ]
        self.dropout = tf.keras.layers.Dropout(dropout)
        self.final_dense = tf.keras.layers.Dense(
            10, activation="softmax", kernel_regularizer=self.regularizer
        )
```



```python
    def call(self, x, training=False):
        for conv_layer in self.conv:
            x = conv_layer(x)
            x = self.maxpool(x)

        x = self.flatten(x)
        for dense_layer in self.dense:
            x = dense_layer(x)
            if training:
                x = self.dropout(x)

        return self.final_dense(x)


def main(a):
    tf.random.set_seed(FLAGS.random_seed)

    data = Data(FLAGS.data_dir)
    model = Model(
        FLAGS.conv_filters, FLAGS.dense_widths, FLAGS.dropout, FLAGS.kernel_reg
    )

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=FLAGS.learning_rate),
        loss=tf.keras.losses.SparseCategoricalCrossentropy(),
        metrics=tf.keras.metrics.SparseCategoricalAccuracy(),
    )

    model.fit(
        data.training_images,
        data.training_labels,
        epochs=FLAGS.epochs,
        verbose=2,
        validation_split=FLAGS.val_split,
    )

    model.summary()

    metrics = model.evaluate(data.testing_images, data.testing_labels, verbose=0
)

    print(model.metrics_names[1] + ": " + str(metrics[1]))


if __name__ == "__main__":
    app.run(main)
```