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

# ECE472-Samuel Maltz
# Assignment 5: Classification of the AG News dataset using a convolutional
# neural network

import tensorflow as tf
import numpy as np
import csv

from absl import flags
from absl import app

FLAGS = flags.FLAGS
flags.DEFINE_string("data_dir", "data", "Directory of AG News dataset")
flags.DEFINE_integer("max_len", 90, "Max length of sequences")
flags.DEFINE_integer("embedding_size", 32, "Size of embedding vector")
flags.DEFINE_list("conv_filters", [32, 64], "Number of filters of convolutional layers")
flags.DEFINE_integer(
    "conv_per_pool", 1, "Number of convolutional layers per pooling layer"
)
flags.DEFINE_integer("pool_size", 2, "Window size of max pool")
flags.DEFINE_float("dropout", 0.2, "Dropout rate")
flags.DEFINE_float("learning_rate", 0.001, "Learning rate for Adam optimizer")
flags.DEFINE_integer("epochs", 1, "Number of training epochs")
flags.DEFINE_float("val_split", 0.1, "Validation fraction")
flags.DEFINE_float("kernel_reg", 0.001, "Regularizer coefficient")
flags.DEFINE_integer("random_seed", 12345, "Random seed")


class Data(object):
    def __init__(self, data_dir, max_len):
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(oov_token=True)

        self.train_sequences, self.train_labels = self.read_data(
            data_dir + "/train.csv", max_len
        )
        self.test_sequences, self.test_labels = self.read_data(
            data_dir + "/test.csv", max_len, test=True
        )

    def read_data(self, data_dir, max_len, test=False):
        texts = []
        labels = np.array([])
        with open(data_dir, "r") as data_csv:
            data_reader = csv.reader(data_csv)
            for row in data_reader:
                texts.append(row[1] + row[2])  # concatenates header and descrip
tion

                labels = np.append(labels, int(row[0]))

        labels = labels - 1  # labels are given as 1-4, converts to 0-3

        # Creates vocabulary only on training texts
        if not test:
            self.tokenizer.fit_on_texts(texts)

        # Pads or truncates sequences to max_len
        sequences = tf.keras.preprocessing.sequence.pad_sequences(
            self.tokenizer.texts_to_sequences(texts), maxlen=max_len, padding="p
ost"
```

```
        )

        return sequences, labels


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

        self.embedding = tf.keras.layers.Embedding(
            vocab_size + 1, embedding_size, mask_zero=True
        )

        self.conv = [
            {
                "conv": [
                    {
                        "conv": tf.keras.layers.Conv1D(
                            i,
                            3,
                            padding="same",
                            kernel_regularizer=self.regularizer,
                        ),
                        "batchnorm": tf.keras.layers.TimeDistributed(
                            tf.keras.layers.BatchNormalization()
                        ),
                        "relu": tf.keras.layers.ReLU(),
                        "dropout": tf.keras.layers.Dropout(dropout),
                    }
                    for j in range(conv_per_pool)
                ],
                "maxpool": tf.keras.layers.MaxPool1D(pool_size),
            }
            for i in conv_filters
        ]

        self.flatten = tf.keras.layers.Flatten()

        self.dense = tf.keras.layers.Dense(
            4, activation="softmax", kernel_regularizer=self.regularizer
        )

    def call(self, x, training=False):
        x = self.embedding(x)

        for block in self.conv:
            for layer in block["conv"]:
                x = layer["conv"](x)
                x = layer["batchnorm"](x)
                x = layer["relu"](x)
                if training:
                    x = layer["dropout"](x)
            x = block["maxpool"](x)

        x = self.flatten(x)

        return self.dense(x)


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

    FLAGS.conv_filters = list(map(int, FLAGS.conv_filters))

    data = Data(FLAGS.data_dir + "/ag_news_csv", FLAGS.max_len)
    model = Model(
        len(data.tokenizer.word_index),
        FLAGS.embedding_size,
        FLAGS.conv_filters,
        FLAGS.conv_per_pool,
        FLAGS.pool_size,
        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(),
    )

    callback = tf.keras.callbacks.EarlyStopping(
        monitor="val_sparse_categorical_accuracy",
        patience=3,
        restore_best_weights=True,
    )

    model.fit(
        data.train_sequences,
        data.train_labels,
        epochs=FLAGS.epochs,
        callbacks=[callback],
        verbose=2,
        validation_split=FLAGS.val_split,
    )

    model.summary()

    model.evaluate(data.test_sequences, data.test_labels, verbose=2)


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

```
3375/3375 - 57s - loss: 0.4577 - sparse_categorical_accuracy: 0.8579 - val_loss:
 0.3140 - val_sparse_categorical_accuracy: 0.9097
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        multiple                  4358432
_________________________________________________________________
time_distributed (TimeDistri multiple                  128
_________________________________________________________________
conv1d (Conv1D)              multiple                  3104
_________________________________________________________________
dropout (Dropout)            multiple                  0
_________________________________________________________________
re_lu (ReLU)                 multiple                  0
_________________________________________________________________
max_pooling1d (MaxPooling1D) multiple                  0
_________________________________________________________________
time_distributed_1 (TimeDist multiple                  256
_________________________________________________________________
conv1d_1 (Conv1D)            multiple                  6208
_________________________________________________________________
dropout_1 (Dropout)          multiple                  0
_________________________________________________________________
re_lu_1 (ReLU)               multiple                  0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 multiple                  0
_________________________________________________________________
flatten (Flatten)            multiple                  0
_________________________________________________________________
dense (Dense)                multiple                  5636
=================================================================
Total params: 4,373,764
Trainable params: 4,373,572
Non-trainable params: 192
_________________________________________________________________
238/238 - 1s - loss: 0.3072 - sparse_categorical_accuracy: 0.9120
```