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

# ECE472-Samuel Maltz
# Assignment 4: Classification of CIFAR10 and CIFAR100 datasets using
# convolutional neural networks

# As a first attempt at classifying the CIFAR10 dataset, the model used to
# classify the MNIST data was reused with CNN layers with 32, 64, 128 and 256
# filters followed by dense layers with widths of 1024, 512, 256, 128 and 10.
# The learning rate and L2 kernel regularization coefficient were 0.001 and
# dropout between dense layers was 20%. This initial attempt achieved an
# accuracy of 72%. Next, by running through different values for the learning
# rate, kernel regularizer coefficient and dropout it was determined that the
# best values were 0.001, 0.0005 and 0.3 respectively. This raised the
# accuracy to 76%. After this batch normalization was experimented between
# layers and it was determined it was best for only the CNN layers.
# Additionally, dropout was experimented on the CNN layers and was found to
# improve performance as well. These changes raised the accuracy to 81%.
# Finally the amount of convolutional filters and dense widths were
# experimented on and it was determined that doubling the filters in all
# convolutional layers to 64, 128, 256 and 512 and actually removing all dense
# layers besides for the last layer produced the best results. These results
# can be found in the results10.txt file and it can be seen that the model
# achieves an accuracy of 87.35% on the test dataset.
#
# With regards to the CIFAR100 dataset, the same model used on the CIFAR10
# dataset was attempted first. Afterwards, different parameters were varied as
# in the CIFAR10 dataset; however, it turned out that most of the settings
# were optimal for this model structure except that an additional dense layer
# with width 1024 is added. The results can be found in results100.txt which
# show that this model achieves an accuracy of 86.16% on the test dataset,
# unfortunately lower than the 90% goal.
#
# The dataset and unpickle function comes from:
# Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

import tensorflow as tf
import numpy as np
import pickle

from absl import app
from absl import flags

FLAGS = flags.FLAGS
flags.DEFINE_bool(
    "cifar100",
    False,
    "Whether to use the CIFAR-100 dataset instead of the CIFAR-10 dataset",
)
flags.DEFINE_string(
    "cifar10_dir", "cifar-10-batches-py", "Name of directory with CIFAR10 dataset"
)
flags.DEFINE_string(
    "cifar100_dir", "cifar-100-python", "Name of directory with CIFAR100 dataset"
)
flags.DEFINE_list(
    "conv_filters", [64, 128, 256, 512], "Number of filters of convolutional layers"
)
flags.DEFINE_integer(
    "conv_per_pool", 2, "Number of convolutional layers per pooling layer"
)
flags.DEFINE_integer("pool_size", 2, "Window size of max pool")
```



```python
flags.DEFINE_list("dense_widths", [], "Widths of dense layers")
flags.DEFINE_float("dropout", 0.3, "Dropout rate")
flags.DEFINE_float("learning_rate", 0.0005, "Learning rate for Adam optimizer")
flags.DEFINE_integer("epochs", 50, "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, cifar_dir, cifar100):
        if cifar100:
            data = self.unpickle(cifar_dir + "/train")
            self.train_images = self.preprocess_images(data[b"data"])
            self.train_labels = self.preprocess_labels(data[b"fine_labels"])

            data = self.unpickle(cifar_dir + "/test")
            self.test_images = self.preprocess_images(data[b"data"])
            self.test_labels = self.preprocess_labels(data[b"fine_labels"])
        else:
            self.train_images = np.array([]).reshape(0, 32, 32, 3)
            self.train_labels = np.array([])
            for i in range(1, 6):
                data = self.unpickle(cifar_dir + "/data_batch_" + str(i))
                self.train_images = np.concatenate(
                    (self.train_images, self.preprocess_images(data[b"data"]))
                )
                self.train_labels = np.concatenate(
                    (self.train_labels, self.preprocess_labels(data[b"labels"]))
                )

            data = self.unpickle(cifar_dir + "/test_batch")
            self.test_images = self.preprocess_images(data[b"data"])
            self.test_labels = self.preprocess_labels(data[b"labels"])

    def unpickle(self, file):
        with open(file, "rb") as fo:
            dict = pickle.load(fo, encoding="bytes")
        return dict

    def preprocess_images(self, images):
        return np.transpose(np.reshape(images, (-1, 3, 32, 32)), (0, 2, 3, 1)).a
stype(
            "float32"
        )

    def preprocess_labels(self, labels):
        return np.array(labels).astype("float32")


class Model(tf.keras.Model):
    def __init__(
        self,
        conv_filters,
        conv_per_pool,
        pool_size,
        dense_widths,
        dropout,
        kernel_reg,
        num_categories,
    ):
        super().__init__()
```

```
        self.regularizer = tf.keras.regularizers.L2(kernel_reg)

        # Convolution block
        self.conv = [
            {
                "conv": [
                    {
                        "conv": tf.keras.layers.Conv2D(i, 3, padding="same"),
                        "batchnorm": tf.keras.layers.BatchNormalization(),
                        "relu": tf.keras.layers.ReLU(),
                    }
                    for j in range(conv_per_pool)
                ],
                "maxpool": tf.keras.layers.MaxPool2D(pool_size),
                "dropout": tf.keras.layers.Dropout(dropout),
            }
            for i in conv_filters
        ]
        self.flatten = tf.keras.layers.Flatten()

        # Dense block
        self.dense = [
            {
                "dense": tf.keras.layers.Dense(i, kernel_regularizer=self.regular
izer),
                "relu": tf.keras.layers.ReLU(),
                "dropout": tf.keras.layers.Dropout(dropout),
            }
            for i in dense_widths
        ]
        self.final_dense = tf.keras.layers.Dense(
            num_categories, activation="softmax", kernel_regularizer=self.regular
izer
        )

    def call(self, x, training=False):
        for conv_block in self.conv:
            for conv_layer in conv_block["conv"]:
                x = conv_layer["conv"](x)
                x = conv_layer["batchnorm"](x)
                x = conv_layer["relu"](x)

            x = conv_block["maxpool"](x)
            x = conv_block["dropout"](x)

        x = self.flatten(x)
        for dense_layer in self.dense:
            x = dense_layer["dense"](x)
            x = dense_layer["relu"](x)
            if training:
                x = dense_layer["dropout"](x)

        return self.final_dense(x)


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

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

    if FLAGS.cifar100:
```



```
        num_categories = 100
        cifar_dir = FLAGS.cifar100_dir
        k = 5  # for top k accuracy
    else:
        num_categories = 10
        cifar_dir = FLAGS.cifar10_dir
        k = 1

    data = Data(cifar_dir, FLAGS.cifar100)
    model = Model(
        FLAGS.conv_filters,
        FLAGS.conv_per_pool,
        FLAGS.pool_size,
        FLAGS.dense_widths,
        FLAGS.dropout,
        FLAGS.kernel_reg,
        num_categories,
    )

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

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

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

    model.summary()

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


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