ECE-471 Selected Topics in Machine Learning
Prof. Curro
Assignment 4

Evan Bubniak

October 10, 2019

1 Results

On CIFAR-10, the CNN specified below achieves 86% accuracy on the training set and 82% accuracy on the test set, for a top-5 categorical accuracy of 99.5% and 99.1%. On CIFAR-100, it respectively achieves 58% and 52%, for a top-5 categorical accuracy of 87.2% and 80.8%. Both CIFAR datasets are loaded using `tensorflow.keras.datasets` and preprocessed using Keras's `ImageDataGenerator` class, adding horizontal flips, rotations, crops, and ZCA whitening to improve regularization and expand the dataset size. The CIFAR-10 dataset was trained over 100 epochs, and the CIFAR-100 dataset was trained over 50.

For this assignment I defined *state of the art* to be at least 80% accuracy on the CIFAR-10 dataset; I aimed to avoid deep models with more than 20 layers to optimize for training time and computing resources, as well as to allow more rapid model prototyping. This necessarily meant looking past models which had performed with high ($> 95\%$) accuracy, but required well over a hundred epochs to train with relatively small batch sizes. I chose a batch size of 100 after initially trying a batch size of 1000, finding large batch sizes, even with batch normalization, caused drastic overfitting.

Several different techniques, including residual networks, a simple reuse of my

MNIST CNN (which used only a single convolutional layer), and a deeper version of the MNIST CNN which stacked multiple convolutional layers together, were tested. Even with normalization techniques, I found the residual network, which pooled at every layer, to drastically underperform on the training set (being no better than random guessing). Initially I suspected this to be a preprocessing mismatch between datasets, but the simpler model generalized better; the deep CNN came close to the target accuracy, achieving a top-5 categorical accuracy of 77.5% on the test set for CIFAR-100.

The final model I settled on was wide residual networks, which promised similar performance to ResNet-1001 with significantly fewer parameters and less train time.

Implementation bugs were a major headache, whereby the model not only underperformed on the training set, capping out at low accuracy even while using techniques from papers reporting high accuracy, but also achieved sub-guess accuracy on the validation set after 17 epochs. I ended up resolving this by ditching my previous code structure, which subclassed all the ResNet and convolutional blocks as Keras layers, and using a functional form (wrapping each block in a function rather than a class) instead.

2 Code

```
import sys
import numpy as np
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import tensorflow.keras.callbacks as callbacks
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Layer, Input, Add, Activation, Flatten, Dense, Conv2D,
        AveragePooling2D, BatchNormalization
from tensorflow.keras.regularizers import l2

L2_PENALTY = 0.0005
CHANNEL_AXIS = 1 if K.image_data_format() == "channels_first" else -1
RANDOM_SEED = 31415
BATCH_SIZE = 64
NUM_EPOCHS = 50

if len(sys.argv) > 1:
DATASET = sys.argv[1]
else:
DATASET = "cifar10"


if DATASET == "cifar100":
        print("---CIFAR-100---:")
        from tensorflow.keras.datasets import cifar100 as cifar
else:
        print("---CIFAR-10---:")
        from tensorflow.keras.datasets import cifar10 as cifar


def initial_convolution(input_layer, img_shape):
        x = Conv2D(16, (3, 3), padding='same', kernel_initializer='he_normal',
                kernel_regularizer=l2(L2_PENALTY),
                use_bias=False, input_shape = img_shape)(input_layer)
        x = BatchNormalization(axis=CHANNEL_AXIS,
                momentum=0.1, epsilon=1e-5,
                gamma_initializer='uniform')(x)
        x = Activation('relu')(x)

        return x

def expand_convolution(input_layer, base, k, strides = (1,1)):

        x = Conv2D(base * k, (3, 3), padding='same', strides = strides,
                kernel_initializer='he_normal',
                kernel_regularizer=l2(L2_PENALTY),
                use_bias=False)(input_layer)

        x = BatchNormalization(axis=CHANNEL_AXIS,
                momentum=0.1, epsilon=1e-5,
                gamma_initializer='uniform')(x)

        x = Activation('relu')(x)

        x = Conv2D(base * k, (3, 3), padding='same', kernel_initializer='he_normal',
                kernel_regularizer=l2(L2_PENALTY),
                use_bias=False)(x)

        skip = Conv2D(base * k, (1, 1), padding='same', strides = strides,
                kernel_initializer='he_normal',
                kernel_regularizer=l2(L2_PENALTY),
                use_bias=False)(input_layer)

        out = Add()([x, skip])

        return out

def convolutional_block(input_layer, num_filters, k=1):
        x = BatchNormalization(axis=CHANNEL_AXIS,
                momentum=0.1, epsilon=1e-5,
                gamma_initializer='uniform')(input_layer)
        x = Activation('relu')(x)
        x = Conv2D(num_filters * k, (3, 3), padding='same',
                kernel_initializer='he_normal',
                kernel_regularizer=l2(L2_PENALTY),
                use_bias=False)(x)
        x = Activation('relu')(x)
        x = Conv2D(num_filters * k, (3, 3), padding='same',
                kernel_initializer='he_normal',
                kernel_regularizer=l2(L2_PENALTY),
                use_bias=False)(x)
        out = Add()([x, input_layer])
        return out

def batchnorm_activate(input_layer):
        x = input_layer
        x = BatchNormalization(axis=CHANNEL_AXIS, momentum=0.1, epsilon=1e-5,
                gamma_initializer='uniform')(x)
        x = Activation('relu')(x)
        return x

class WideResNet:
        def __init__(self, img_shape, num_labels=10, N=2, k=1,):

                input_layer = Input(shape = img_shape)
                x = initial_convolution(input_layer, img_shape)
                x = expand_convolution(x, 16, k, strides = (1,1))
                x = convolutional_block(x, 16, k)
                x = batchnorm_activate(x)
                x = expand_convolution(x, 32, k, strides = (2,2))
                x = convolutional_block(x, 32, k)
                x = batchnorm_activate(x)
                x = expand_convolution(x, 64, k, strides = (2,2))
                x = convolutional_block(x, 64, k)
                x = batchnorm_activate(x)
                x = AveragePooling2D((8,8))(x)
                x = Flatten()(x)
                x = Dense(num_labels, kernel_regularizer=l2(L2_PENALTY),
                        activation='softmax')(x)
                self.model = Model(input_layer, x)

        def compile(self):

                self.model.compile(loss="categorical_crossentropy", optimizer="adam",
                        metrics=["acc", "top_k_categorical_accuracy"])

                self.model.summary()

        def fit_generator(self, generator, X_train, y_train, X_val, y_val):
                self.model_log = self.model.fit_generator(
                        generator.flow(X_train, y_train, batch_size=BATCH_SIZE),
                        steps_per_epoch=len(X_train) // BATCH_SIZE,
                        epochs=NUM_EPOCHS,
                        callbacks=[
                                callbacks.ModelCheckpoint("{}-weights.h5".format(DATASET),
                        monitor="val_acc",
                        save_best_only=True,
                        verbose=1)],
                validation_data=(X_val, y_val),
                validation_steps=X_val.shape[0] // BATCH_SIZE)

        return self.model_log


        def test(self, X_test, y_test):
                self.model.evaluate(X_test, y_test, verbose = 1)

        def load_weights(self):
                self.model.load_weights("{}-weights.h5".format(DATASET))

def get_dataset():
        (X_train, y_train), (X_test, y_test) = cifar.load_data()

        X_train = X_train.astype('float32')
```

```
        X_train = (X_train - X_train.mean(axis=0)) / (X_train.std(axis=0))

        X_test = X_test.astype('float32')
        X_test = (X_test - X_test.mean(axis=0)) / (X_test.std(axis=0))

        y_train = keras.utils.to_categorical(y_train)
        y_test = keras.utils.to_categorical(y_test)


        return X_train, y_train, X_test, y_test

def split_training_set(X_train, y_train, test_size=1/10):
        X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
                test_size=test_size, random_state=RANDOM_SEED)
        return X_train, y_train, X_val, y_val



generator = ImageDataGenerator(rotation_range=10,
        width_shift_range=5./32,
        height_shift_range=5./32,
        zca_whitening = True)


X_train, y_train, X_test, y_test = get_dataset()
#X_train, X_val, y_train, y_val = split_training_set(X_train, y_train)

img_shape = X_train.shape[1:]
num_labels = len(y_train[0])

model = WideResNet(img_shape, num_labels=num_labels, N=2, k=8)
model.compile()
model.load_weights()

model.fit_generator(generator, X_train, y_train, X_test, y_test)

model.test(X_test, y_test)
```