Experiments with Normalization/Optimization

CGML Assignment 4

Camille Chow

For classifying the CIFAR100 dataset, I was able to adapt my convolutional neural net from Assignment 3 with very few alterations, the main difference being the use of batch normalization and the number of filters at each layer. Other hyperparameters were adjusted and tested, however, none of these experiments produced a significantly improved top-5 accuracy. The number of epochs was also increased to 10 and the activation function for the convolutional layers was changed to elu rather than relu.

For the CIFAR10 dataset, I initially attempted to use a similar model as the CIFAR100, but couldn't produce an accuracy better than 70%. Initial attempts at increasing the depth of the model resulted in decreased accuracy. The current architecture is modeled after the following tutorial: https://appliedmachinelearning.blog/2018/03/24/achieving-90-accuracy-in-object-recognition-task-on-cifar-10-dataset-with-keras-convolutional-neural-networks/, which uses six convolutional layers with increasing filter size. I attempted to implement data augmentation, however, this too resulted in lower accuracy rates (likely due to overfitting) and extra computation time, so I chose to omit it. I added an additional dense layer and dropout layer to improve accuracy. After tuning and testing with various optimizers, pooling and kernel sizes, filter numbers, dropout rates, activation functions, batch norm parameters, and learning rate schedules, I arrived to similar hyperparameters from the tutorial, as these were already close to optimal. From that point on, the only gains made in accuracy were achieved by increasing the number of epochs. I was able to achieve up to 80% accuracy with 10 epochs, 85% accuracy with 30 epochs, and 87% with 100 epochs. With more time and computational capability, an accuracy closer to state of the art can be achieved.

```
#Camille Chow
#ECE 471 Assignment 4
#Classifying CIFAR10 data
#Citation: Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.
import numpy as np
import tensorflow as tf

#dimensional constants
num_classes = 10
image_h = 32
image_w = 32
channels = 3

#get data
(x_train, y_train),(x_test, y_test) = tf.keras.datasets.cifar10.load_data()

#shape test data
x_test = x_test.reshape(x_test.shape[0], image_h, image_w, channels)
x_test = x_test.astype('float32')
x_test /= 255.0
# shape_data(x_test)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

#shape training data
x_train = x_train.reshape(x_train.shape[0], image_h, image_w, channels)
x_train = x_train.astype('float32')
x_train /= 255.0

# shape_data(x_train)
y_train = tf.keras.utils.to_categorical(y_train, num_classes)

#tunable hyperparams
batch_size = 100
epochs = 100
lam = .001

def lr_schedule(epoch):
    lrate = 0.001
    if epoch > 65:
        lrate = 0.0005
    elif epoch > 85:
        lrate = 0.0001
    return lrate

def add_conv_layer(model, num_filters):
        model.add(tf.keras.layers.Conv2D(num_filters, kernel_size=3,
                strides=(1, 1), activation='elu', padding='same'))

def add_pooling_layer(model):
        model.add(tf.keras.layers.MaxPooling2D(pool_size=3, strides=2))

def add_bn_layer(model):
        model.add(tf.keras.layers.BatchNormalization(momentum=0.99, epsilon=0.001))

#build cnn
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(32, kernel_size=3,
                strides=(1, 1), activation='elu', padding='same', input_shape=(image_h, i
mage_w, channels)))
add_bn_layer(model)
add_conv_layer(model, 32)
add_bn_layer(model)
add_pooling_layer(model)
model.add(tf.keras.layers.Dropout(.2))
add_conv_layer(model, 64)
add_bn_layer(model)
add_conv_layer(model, 64)
add_bn_layer(model)
add_pooling_layer(model)
model.add(tf.keras.layers.Dropout(.3))
```

```
add_conv_layer(model, 128)
add_bn_layer(model)
add_conv_layer(model, 128)
add_bn_layer(model)
add_pooling_layer(model)
model.add(tf.keras.layers.Dropout(.4))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='elu',kernel_regularizer=tf.keras.regular
izers.l2(lam)))
model.add(tf.keras.layers.Dropout(.5))
model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

#train model
model.compile(loss=tf.keras.losses.categorical_crossentropy, optimizer=tf.keras.optimizer
s.Adam(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_s
plit=.1, callbacks=[tf.keras.callbacks.LearningRateScheduler(lr_schedule)])

#test model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

[54]

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/100
45000/45000 [==============================] - 30s 677us/step - loss: 2.3743 - acc: 0.3741 - val_loss: 1.8629 - val_acc: 0.4856
Epoch 2/100
45000/45000 [==============================] - 24s 538us/step - loss: 1.5453 - acc: 0.5613 - val_loss: 1.2849 - val_acc: 0.6490
Epoch 3/100
45000/45000 [==============================] - 24s 533us/step - loss: 1.2298 - acc: 0.6472 - val_loss: 1.0070 - val_acc: 0.7192
Epoch 4/100
45000/45000 [==============================] - 24s 534us/step - loss: 1.0336 - acc: 0.7006 - val_loss: 1.0041 - val_acc: 0.7114
Epoch 5/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.9176 - acc: 0.7281 - val_loss: 0.8150 - val_acc: 0.7588
Epoch 6/100
45000/45000 [==============================] - 24s 533us/step - loss: 0.8274 - acc: 0.7555 - val_loss: 0.9696 - val_acc: 0.7198
Epoch 7/100
45000/45000 [==============================] - 24s 532us/step - loss: 0.7739 - acc: 0.7704 - val_loss: 0.7864 - val_acc: 0.7676
Epoch 8/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.7169 - acc: 0.7900 - val_loss: 0.6932 - val_acc: 0.8034
Epoch 9/100
45000/45000 [==============================] - 24s 533us/step - loss: 0.6732 - acc: 0.8024 - val_loss: 0.7178 - val_acc: 0.7830
Epoch 10/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.6438 - acc: 0.8140 - val_loss: 0.6609 - val_acc: 0.8078
Epoch 11/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.6246 - acc: 0.8230 - val_loss: 0.6568 - val_acc: 0.8198
Epoch 12/100
45000/45000 [==============================] - 24s 532us/step - loss: 0.5965 - acc: 0.8321 - val_loss: 0.6388 - val_acc: 0.8232
Epoch 13/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.5719 - acc: 0.8398 - val_loss: 0.6088 - val_acc: 0.8238
Epoch 14/100
45000/45000 [==============================] - 24s 532us/step - loss: 0.5529 - acc: 0.8474 - val_loss: 0.6939 - val_acc: 0.8080
Epoch 15/100
45000/45000 [==============================] - 24s 537us/step - loss: 0.5347 - acc: 0.8524 - val_loss: 0.6367 - val_acc: 0.8216
Epoch 16/100
45000/45000 [==============================] - 24s 536us/step - loss: 0.5166 - acc: 0.8600 - val_loss: 0.5784 - val_acc: 0.8370
Epoch 17/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.5038 - acc: 0.8642 - val_loss: 0.5977 - val_acc: 0.8416
Epoch 18/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.4837 - acc: 0.8722 - val_loss: 0.5962 - val_acc: 0.8430
Epoch 19/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.4794 - acc: 0.8732 - val_loss: 0.6371 - val_acc: 0.8266
Epoch 20/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.4686 - acc: 0.8777 - val_loss: 0.5739 - val_acc: 0.8492
Epoch 21/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.4517 - acc: 0.8826 - val_loss: 0.5712 - val_acc: 0.8528
Epoch 22/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.4476 - acc: 0.8847 - val_loss: 0.5920 - val_acc: 0.8482
Epoch 23/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.4280 - acc: 0.8918 - val_loss: 0.6921 - val_acc: 0.8250
Epoch 24/100
45000/45000 [==============================] - 24s 532us/step - loss: 0.4321 - acc: 0.8894 - val_loss: 0.5785 - val_acc: 0.8466
```

```
[54] Epoch 85/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.1499 - acc: 0.9692 - val_loss: 0.5803 - val_acc: 0.8732
Epoch 86/100
45000/45000 [==============================] - 24s 533us/step - loss: 0.1513 - acc: 0.9687 - val_loss: 0.5901 - val_acc: 0.8748
Epoch 87/100
45000/45000 [==============================] - 24s 533us/step - loss: 0.1519 - acc: 0.9688 - val_loss: 0.5801 - val_acc: 0.8756
Epoch 88/100
45000/45000 [==============================] - 24s 536us/step - loss: 0.1489 - acc: 0.9699 - val_loss: 0.5815 - val_acc: 0.8742
Epoch 89/100
45000/45000 [==============================] - 24s 529us/step - loss: 0.1485 - acc: 0.9698 - val_loss: 0.6002 - val_acc: 0.8718
Epoch 90/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.1475 - acc: 0.9704 - val_loss: 0.5930 - val_acc: 0.8748
Epoch 91/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.1509 - acc: 0.9684 - val_loss: 0.5958 - val_acc: 0.8704
Epoch 92/100
45000/45000 [==============================] - 24s 533us/step - loss: 0.1513 - acc: 0.9685 - val_loss: 0.5827 - val_acc: 0.8720
Epoch 93/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.1472 - acc: 0.9702 - val_loss: 0.6062 - val_acc: 0.8748
Epoch 94/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.1467 - acc: 0.9706 - val_loss: 0.5768 - val_acc: 0.8778
Epoch 95/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.1516 - acc: 0.9690 - val_loss: 0.5836 - val_acc: 0.8774
Epoch 96/100
45000/45000 [==============================] - 24s 536us/step - loss: 0.1486 - acc: 0.9694 - val_loss: 0.6085 - val_acc: 0.8702
Epoch 97/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.1418 - acc: 0.9721 - val_loss: 0.6044 - val_acc: 0.8708
Epoch 98/100
45000/45000 [==============================] - 24s 537us/step - loss: 0.1432 - acc: 0.9718 - val_loss: 0.6061 - val_acc: 0.8762
Epoch 99/100
45000/45000 [==============================] - 24s 536us/step - loss: 0.1403 - acc: 0.9721 - val_loss: 0.6220 - val_acc: 0.8698
Epoch 100/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.1430 - acc: 0.9714 - val_loss: 0.6058 - val_acc: 0.8756
<tensorflow.python.keras.callbacks.History at 0x7fbebdb45f28>
```

```
Epoch 32/100
45000/45000 [==============================] - 24s 535us/step - loss: 0.3666 - acc: 0.9106 - val_loss: 0.5892 - val_acc: 0.8570
Epoch 33/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.3686 - acc: 0.9116 - val_loss: 0.5923 - val_acc: 0.8610
Epoch 34/100
45000/45000 [==============================] - 24s 529us/step - loss: 0.3610 - acc: 0.9143 - val_loss: 0.5727 - val_acc: 0.8610
Epoch 35/100
45000/45000 [==============================] - 24s 532us/step - loss: 0.3562 - acc: 0.9173 - val_loss: 0.5948 - val_acc: 0.8626
Epoch 36/100
45000/45000 [==============================] - 24s 532us/step - loss: 0.3501 - acc: 0.9170 - val_loss: 0.6146 - val_acc: 0.8586
Epoch 37/100
45000/45000 [==============================] - 24s 534us/step - loss: 0.3452 - acc: 0.9202 - val_loss: 0.6417 - val_acc: 0.8452
Epoch 38/100
45000/45000 [==============================] - 24s 532us/step - loss: 0.3433 - acc: 0.9201 - val_loss: 0.5926 - val_acc: 0.8610
Epoch 39/100
```

[58]
```
#test model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.6457389142274856
Test accuracy: 0.8678
```

```
#Camille Chow
#ECE 471 Assignment 4
#Classifying CIFAR100 data
#Citation: Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.
import numpy as np
import tensorflow as tf

#dimensional constants
num_classes = 100
image_h = 32
image_w = 32
channels = 3

#get data
(x_train, y_train),(x_test, y_test) = tf.keras.datasets.cifar100.load_data()

#shape test data
x_test = x_test.reshape(x_test.shape[0], image_h, image_w, channels)
x_test = x_test.astype('float32')
x_test /= 255.0
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

#shape training data
x_train = x_train.reshape(x_train.shape[0], image_h, image_w, channels)
x_train = x_train.astype('float32')
x_train /= 255.0
y_train = tf.keras.utils.to_categorical(y_train, num_classes)

#tunable hyperparams
batch_size = 100
epochs = 10
dropout = .5
dense_units = 1000
lam = .001

def add_conv_layer(model, num_filters):
        model.add(tf.keras.layers.Conv2D(num_filters, kernel_size=3,
                strides=(1, 1), activation='elu', padding='same'))

def add_pooling_layer(model):
        model.add(tf.keras.layers.MaxPooling2D(pool_size=3, strides=3))

def add_bn_layer(model):
        model.add(tf.keras.layers.BatchNormalization(momentum=0.99, epsilon=0.001))

#build cnn
model = tf.keras.Sequential()
add_conv_layer(model, 64)
add_bn_layer(model)
add_pooling_layer(model)
add_conv_layer(model, 32)
add_bn_layer(model)
add_pooling_layer(model)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(dense_units, activation='relu', kernel_regularizer=tf.ker
as.regularizers.l2(lam)))
model.add(tf.keras.layers.Dropout(dropout))
model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

#train model
model.compile(loss=tf.keras.losses.categorical_crossentropy, optimizer=tf.keras.optimizer
s.Adam(), metrics=['accuracy', 'top_k_categorical_accuracy'])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_s
plit=.1)

#test model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print('Test top-5 accuracy:', score[2])
```

```
-bash-4.2$ python cifar100.py
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169009152/169001437 [==============================] - 20s 0us/step
Train on 45000 samples, validate on 5000 samples
Epoch 1/10
2018-10-02 16:18:13.770909: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was
 not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
2018-10-02 16:18:13.771984: I tensorflow/core/common_runtime/process_util.cc:69] Creating new thread pool with default inter op setting: 2. Tun
e using inter_op_parallelism_threads for best performance.
45000/45000 [==============================] - 74s 2ms/step - loss: 4.0723 - acc: 0.1453 - top_k_categorical_accuracy: 0.3727 - val_loss: 4.149
0 - val_acc: 0.1442 - val_top_k_categorical_accuracy: 0.3724
Epoch 2/10
45000/45000 [==============================] - 74s 2ms/step - loss: 3.3010 - acc: 0.2621 - top_k_categorical_accuracy: 0.5537 - val_loss: 3.247
4 - val_acc: 0.2740 - val_top_k_categorical_accuracy: 0.5772
Epoch 3/10
45000/45000 [==============================] - 73s 2ms/step - loss: 2.9467 - acc: 0.3259 - top_k_categorical_accuracy: 0.6344 - val_loss: 3.081
5 - val_acc: 0.3026 - val_top_k_categorical_accuracy: 0.6100
Epoch 4/10
45000/45000 [==============================] - 73s 2ms/step - loss: 2.7395 - acc: 0.3633 - top_k_categorical_accuracy: 0.6808 - val_loss: 3.353
1 - val_acc: 0.2854 - val_top_k_categorical_accuracy: 0.5634
Epoch 5/10
45000/45000 [==============================] - 73s 2ms/step - loss: 2.5904 - acc: 0.4013 - top_k_categorical_accuracy: 0.7146 - val_loss: 2.734
9 - val_acc: 0.3710 - val_top_k_categorical_accuracy: 0.6870
Epoch 6/10
45000/45000 [==============================] - 73s 2ms/step - loss: 2.4909 - acc: 0.4209 - top_k_categorical_accuracy: 0.7337 - val_loss: 2.611
8 - val_acc: 0.4102 - val_top_k_categorical_accuracy: 0.7160
Epoch 7/10
45000/45000 [==============================] - 73s 2ms/step - loss: 2.4131 - acc: 0.4422 - top_k_categorical_accuracy: 0.7569 - val_loss: 2.718
7 - val_acc: 0.3910 - val_top_k_categorical_accuracy: 0.6956
Epoch 8/10
45000/45000 [==============================] - 73s 2ms/step - loss: 2.3493 - acc: 0.4620 - top_k_categorical_accuracy: 0.7702 - val_loss: 2.637
0 - val_acc: 0.4086 - val_top_k_categorical_accuracy: 0.7202
Epoch 9/10
45000/45000 [==============================] - 73s 2ms/step - loss: 2.2930 - acc: 0.4750 - top_k_categorical_accuracy: 0.7857 - val_loss: 2.587
0 - val_acc: 0.4232 - val_top_k_categorical_accuracy: 0.7308
Epoch 10/10
45000/45000 [==============================] - 73s 2ms/step - loss: 2.2420 - acc: 0.4911 - top_k_categorical_accuracy: 0.7979 - val_loss: 2.583
2 - val_acc: 0.4266 - val_top_k_categorical_accuracy: 0.7286
Test loss: 2.5375137771606444
Test accuracy: 0.4367
Test top-5 accuracy: 0.7359
-bash-4.2$
```