Nithilam Subbaian
10/2/2019

ECE Selected Topics in Machine Learning – Assignment 4

For CIFAR-10 we were told to achieve a performance similar to the state of the art, but I found papers that claimed accuracies of above 98%. I quickly came to realize this was not going to be possible for me for this assignment. After playing with various features, implementing datagen/ImageDataGenerator, normalization techniques as well as LearningRateScheduler I was able to achieve accuracies in the mid 80s range. I was able to reuse the basic structure of the mnist program which I started with, but I had to add a lot more capabilities in order to make it work for CIFAR with a reasonable accuracy. I was able to mostly use the same code for CIFAR-100 from CIFAR-10 with minor modifications, as only 80% was needed for top 5 accuracy.

The datagen/ImageDataGenerator helped perform data augmentation, which was helpful in increasing the accuracy of the model as the diversity of the data used while training the model was increased. Normalization and changing the learning rates were also new methods I learned improved the accuracy of the model. I did not play with the optimizer as Adam suited these purposes, but for further improvement, this is an area to look more into. I had trouble with this assignment with overfitting, as the validation accuracy repeatedly came out low; however, when I decreased the complexity of the model, this resulted in a decrease in test case accuracy, so I chose to leave the model with low validation accuracy. I did the shuffle the data to make sure it was not an ordering issue.

Here is the Program for CIFAR-10:

```
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split

LAMBDA = .0001
NUM_CLASSES = 10
BATCH_SIZE = 64

def lr_schedule(epoch):
	lrate = 0.001
	if epoch > 75:
	lrate = 0.0005
	elif epoch > 100:
	lrate = 0.00025
	elif epoch > 150:
	lrate = 0.000056
	return lrate

class Data(object):
	def __init__(self):

	(self.x_train, self.y_train), (self.x_test, self.y_test) = tf.keras.datasets.cifar10.load_data()

	rand_index = np.arange(len(self.x_train))
	np.random.shuffle(rand_index)
	rand_index2 = np.arange(len(self.x_test))
	np.random.shuffle(rand_index2)
	self.x_train = self.x_train[rand_index]
	self.y_train = self.y_train[rand_index]
	self.x_test = self.x_test[rand_index2]
	self.y_test = self.y_test[rand_index2]
```

```
        self.x_train, self.x_ver, self.y_train, self.y_ver = train_test_split(self.x_train, self.y_train, test_size=5000)

        self.x_train = self.x_train.reshape(self.x_train.shape[0], 32, 32, 3).astype('float32')/255.0
        self.x_ver = self.x_ver.reshape(self.x_ver.shape[0], 32, 32, 3).astype('float32')/255.0
        self.x_test = self.x_test.reshape(self.x_test.shape[0], 32, 32, 3).astype('float32')/255.0

        mean = np.mean(self.x_train,axis=(0,1,2,3))
        std = np.std(self.x_train,axis=(0,1,2,3))
        self.x_train = (self.x_train-mean)/(std+1e-7)
        self.x_test = (self.x_test-mean)/(std+1e-7)

        self.y_train = tf.keras.utils.to_categorical(self.y_train, NUM_CLASSES)
        self.y_test = tf.keras.utils.to_categorical(self.y_test, NUM_CLASSES)
        self.y_ver = tf.keras.utils.to_categorical(self.y_ver, NUM_CLASSES)

class Model(tf.Module):
        def __init__(self):
        self.model = tf.keras.Sequential()
        self.model.add(tf.keras.layers.Conv2D(32, (3,3), padding='same', kernel_regularizer=tf.keras.regularizers.l2(LAMBDA),
input_shape=data.x_train.shape[1:]))

        def activation(self):
        self.model.add(tf.keras.layers.Activation('elu'))

        def batch_norm(self):
        self.model.add(tf.keras.layers.BatchNormalization())

        def conv2D_layer(self):
        self.model.add(tf.keras.layers.Conv2D(128, (3,3), padding='same', kernel_regularizer=tf.keras.regularizers.l2(LAMBDA)))

        def maxPooling2D(self):
        self.model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))

if __name__ == "__main__":
        data = Data()
        model = Model()

        model.activation()
        model.batch_norm()

        model.conv2D_layer()
        model.activation()
        model.batch_norm()

        model.maxPooling2D()
        model.model.add(tf.keras.layers.Dropout(0.4))

        model.conv2D_layer()
        model.activation()
        model.batch_norm()

        model.conv2D_layer()
        model.activation()
        model.batch_norm()

        model.maxPooling2D()

        model.model.add(tf.keras.layers.Dropout(0.4))
        model.model.add(tf.keras.layers.Flatten())
        model.model.add(tf.keras.layers.Dense(NUM_CLASSES, activation='softmax'))
        model.model.summary()
```

```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
rotation_range=15,
width_shift_range=0.1,
height_shift_range=0.1,
horizontal_flip=True,
)

datagen.fit(data.x_train)

model.model.compile(loss='categorical_crossentropy',
optimizer=tf.keras.optimizers.Adam(),
metrics=['accuracy', 'top_k_categorical_accuracy'])

model.model.fit_generator(datagen.flow(data.x_train, data.y_train, batch_size=BATCH_SIZE),
        use_multiprocessing=True,
        steps_per_epoch=data.x_train.shape[0] // BATCH_SIZE, epochs=100,
        verbose=1,validation_data=(data.x_ver, data.y_ver),
        callbacks=[tf.keras.callbacks.LearningRateScheduler(lr_schedule)], workers = 4)

final_result = model.model.evaluate(data.x_test, data.y_test, batch_size=128, verbose=1)
print("Test Loss:", final_result[0])
print("Test Accuracy:", final_result[1])
print("Test Top-5 Accuracy:", final_result[2])
```

Output for 150 epochs:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
activation (Activation)	(None, 32, 32, 32)	0
batch_normalization (BatchNo	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 128)	36992
activation_1 (Activation)	(None, 32, 32, 128)	0
batch_normalization_1 (Batch	(None, 32, 32, 128)	512
max_pooling2d (MaxPooling2D)	(None, 16, 16, 128)	0
dropout (Dropout)	(None, 16, 16, 128)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	147584
activation_2 (Activation)	(None, 16, 16, 128)	0
batch_normalization_2 (Batch	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
activation_3 (Activation)	(None, 16, 16, 128)	0
batch_normalization_3 (Batch	(None, 16, 16, 128)	512
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0

```
dense (Dense)            (None, 10)                81930
=================================================================
Total params: 416,650
Trainable params: 415,818
Non-trainable params: 832
_________________________________________________________________
Epoch 98/100
702/703 [============================>.] - ETA: 0s - loss: 0.4709 - acc: 0.8784 - top_k_categorical_accuracy: 0.9964
703/703 [==============================] - 11s 15ms/step - loss: 0.4708 - acc: 0.8784 - top_k_categorical_accuracy: 0.9964 - val_loss: 2.1340
- val_acc: 0.3979 - val_top_k_categorical_accuracy: 0.8145
Epoch 99/100
702/703 [============================>.] - ETA: 0s - loss: 0.4710 - acc: 0.8770 - top_k_categorical_accuracy: 0.9970
703/703 [==============================] - 11s 15ms/step - loss: 0.4711 - acc: 0.8770 - top_k_categorical_accuracy: 0.9970 - val_loss: 2.1725
- val_acc: 0.4290 - val_top_k_categorical_accuracy: 0.8064
Epoch 100/100
702/703 [============================>.] - ETA: 0s - loss: 0.4606 - acc: 0.8804 - top_k_categorical_accuracy: 0.9969
703/703 [==============================] - 11s 15ms/step - loss: 0.4606 - acc: 0.8804 - top_k_categorical_accuracy: 0.9970 - val_loss: 2.0650
- val_acc: 0.4316 - val_top_k_categorical_accuracy: 0.8200
10000/10000 [==============================] - 1s 71us/sample - loss: 0.5561 - acc: 0.8664 - top_k_categorical_accuracy: 0.9943
Test Loss: 0.5560953780651092
Test Accuracy: 0.8664
Test Top-5 Accuracy: 0.9943
```

For the CIFAR-100, we were told to achieve a performance of top-5 accuracy of 80%.

Here is the program for CIFAR-100:

```
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split

LAMBDA = .0001
NUM_CLASSES = 100
BATCH_SIZE = 64

class Data(object):
        def __init__(self):

        (self.x_train, self.y_train), (self.x_test, self.y_test) = tf.keras.datasets.cifar100.load_data()

        rand_index = np.arange(len(self.x_train))
        np.random.shuffle(rand_index)
        rand_index2 = np.arange(len(self.x_test))
        np.random.shuffle(rand_index2)
        self.x_train = self.x_train[rand_index]
        self.y_train = self.y_train[rand_index]
        self.x_test = self.x_test[rand_index2]
        self.y_test = self.y_test[rand_index2]

        self.x_train, self.x_ver, self.y_train, self.y_ver = train_test_split(self.x_train, self.y_train, test_size=5000)

        self.x_train = self.x_train.reshape(self.x_train.shape[0], 32, 32, 3).astype('float32')/255.0
        self.x_ver = self.x_ver.reshape(self.x_ver.shape[0], 32, 32, 3).astype('float32')/255.0
        self.x_test = self.x_test.reshape(self.x_test.shape[0], 32, 32, 3).astype('float32')/255.0

        mean = np.mean(self.x_train,axis=(0,1,2,3))
        std = np.std(self.x_train,axis=(0,1,2,3))
        self.x_train = (self.x_train-mean)/(std+1e-7)
        self.x_test = (self.x_test-mean)/(std+1e-7)

        self.y_train = tf.keras.utils.to_categorical(self.y_train, NUM_CLASSES)
        self.y_test = tf.keras.utils.to_categorical(self.y_test, NUM_CLASSES)
        self.y_ver = tf.keras.utils.to_categorical(self.y_ver, NUM_CLASSES)
```

```
class Model(tf.Module):
	def __init__(self):
	self.model = tf.keras.Sequential()
	self.model.add(tf.keras.layers.Conv2D(32, (3,3), padding='same', kernel_regularizer=tf.keras.regularizers.l2(LAMBDA),
input_shape=data.x_train.shape[1:]))

	def activation(self):
	self.model.add(tf.keras.layers.Activation('elu'))

	def batch_norm(self):
	self.model.add(tf.keras.layers.BatchNormalization())

	def conv2D_layer(self):
	self.model.add(tf.keras.layers.Conv2D(128, (3,3), padding='same', kernel_regularizer=tf.keras.regularizers.l2(LAMBDA)))

	def maxPooling2D(self):
	self.model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))


if __name__ == "__main__":
	data = Data()
	model = Model()

	model.activation()
	model.batch_norm()

	model.conv2D_layer()
	model.activation()
	model.batch_norm()

	model.maxPooling2D()
	model.model.add(tf.keras.layers.Dropout(0.4))

	model.conv2D_layer()
	model.activation()
	model.batch_norm()

	model.conv2D_layer()
	model.activation()
	model.batch_norm()

	model.maxPooling2D()

	model.model.add(tf.keras.layers.Dropout(0.4))
	model.model.add(tf.keras.layers.Flatten())
	model.model.add(tf.keras.layers.Dense(NUM_CLASSES, activation='softmax'))
	model.model.summary()

	datagen = tf.keras.preprocessing.image.ImageDataGenerator(
	rotation_range=15,
	width_shift_range=0.1,
	height_shift_range=0.1,
	horizontal_flip=True,
	)

	datagen.fit(data.x_train)

	model.model.compile(loss='categorical_crossentropy',
	optimizer=tf.keras.optimizers.Adam(),
	metrics=['accuracy', 'top_k_categorical_accuracy'])
```

```
model.model.fit_generator(datagen.flow(data.x_train, data.y_train, batch_size=BATCH_SIZE),\
        steps_per_epoch=data.x_train.shape[0] // BATCH_SIZE, epochs=50,\
        verbose=1,validation_data=(data.x_ver, data.y_ver))

final_result = model.model.evaluate(data.x_test, data.y_test, batch_size=128, verbose=1)
print("Test Loss:", final_result[0])
print("Test Accuracy:", final_result[1])
print("Test Top-5 Accuracy:", final_result[2])
```

Output for 50 epochs:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 128)       36992
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 128)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 128)       0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 100)               819300
=================================================================
Total params: 1,154,020
Trainable params: 1,153,188
Non-trainable params: 832
_________________________________________________________________
Epoch 45/50
703/703 [==============================] - 22s 31ms/step - loss: 1.4527 - acc: 0.6435 - top_k_categorical_accuracy: 0.8979 - val_loss: 4.5644
- val_acc: 0.1474 - val_top_k_categorical_accuracy: 0.3726
Epoch 46/50
703/703 [==============================] - 22s 31ms/step - loss: 1.4448 - acc: 0.6410 - top_k_categorical_accuracy: 0.8982 - val_loss: 4.6112
- val_acc: 0.1428 - val_top_k_categorical_accuracy: 0.3560
Epoch 47/50
703/703 [==============================] - 22s 31ms/step - loss: 1.4377 - acc: 0.6486 - top_k_categorical_accuracy: 0.8980 - val_loss: 4.7397
- val_acc: 0.1446 - val_top_k_categorical_accuracy: 0.3434
Epoch 48/50
703/703 [==============================] - 22s 31ms/step - loss: 1.4294 - acc: 0.6478 - top_k_categorical_accuracy: 0.9005 - val_loss: 4.3527
- val_acc: 0.1674 - val_top_k_categorical_accuracy: 0.3814
Epoch 49/50
```

```
703/703 [==============================] - 22s 31ms/step - loss: 1.4373 - acc: 0.6490 - top_k_categorical_accuracy: 0.8974 - val_loss: 4.2862
- val_acc: 0.1546 - val_top_k_categorical_accuracy: 0.3858
Epoch 50/50
703/703 [==============================] - 22s 31ms/step - loss: 1.4249 - acc: 0.6511 - top_k_categorical_accuracy: 0.8991 - val_loss: 4.8304
- val_acc: 0.1320 - val_top_k_categorical_accuracy: 0.3510
10000/10000 [==============================] - 1s 74us/sample - loss: 1.8795 - acc: 0.5669 - top_k_categorical_accuracy: 0.8398
Test Loss: 1.8794701976776123
Test Accuracy: 0.5669
Test Top-5 Accuracy: 0.8398
```

Sources that helped me! (along with TensorFlow Docs)

https://www.cs.toronto.edu/~kriz/cifar.html
https://www.codecademy.com/articles/normalization
https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c
https://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/
https://appliedmachinelearning.blog/author/abhijeetchar/