Deep Learning Assignment 4

Nikita Teplitskiy

October 1, 2020

The goal of this assignment was to create a pair of convolutional neural network designed to categorize images from the CIFAR-10 and CIFAR-100 data sets.

1 CIFAR-10 Model

The initial model used for the CIFAR-10 categorizer was based on the "toy ResNet" described in the TensorFlow/Keras documentation. This model worked moderately well achieving 76.9% top-1 accuracy and 98.3% top-5 accuracy. It consisted of 7 convolutional layers, 3 residual layers, and 2 dense layers resulting in a total of 223,242 trainable parameters. Convergence would occur within 15 epochs. This model suffers from significant overfit that was improved by adding regularization to some layers.

The second model attempted was from [2] which was a conventional convolutional network utilizing 4 convolutional layer, 2 dense layers, and max pooling where appropriate. The claimed benefit of this model was its relatively small size and memory footprint which would improve training speed. Unfortunately, the model presented in the paper does not appear to be the model they derived results from. The model as described and illustrated is so much smaller than the predicted model size and provides awful results.

The third design was a convolution only design from the paper [3] "Striving for Simplicity: The All Convolutional Net" that dispersed with any dense layers employing a purely convolutional design. This model performed adequately well achieving a top-1 and top-5 accuracy of 61.0% and 94.5% respectively. Curiously, when the author suggested dropout was added to the model, the top-5 accuracy started out very high and achieved 100% within a few epochs. Given the sub 10% true accuracy of the model, this erroneous result was due to misconfiguration.

2 CIFAR-100 Model

The first model used for the CIFAR-100 dataset was a modification of the third model mentioned previously. The only change made was alterting the output dimension to 100 categories and adding additional regularization. This model achieved mediocre results, reaching 68.9% top-5.

The second model was an enlarged version of the ResNet that was tested for the CIFAR-10 model. An additional convolutional block was added and the sizes of the dense layers were adjusted. This model achieved 79.8% accuracy but further improvement was curtailed by limited training time.

The third model used was lifted directly from a tutorial [4] as I have zero success with the previous two models. This model was fast and allowed me to diagnose some major issues the prevented the previous models from working as expected. This model achieves a subpar top-5 accuracy of 67.4% but made up for it in speed and simplicity.

3 Conclusions

Further exprimentation is needed in saving and restoring models. TensorBoard would be more convenient for monitoring results. Need to set up CUDA environment as CPU training times are getting excessive.

4 References

1. https://www.tensorflow.org/guide/keras/functional#a_toy_resnet_model
2. Calik, Demirci. Cifar-10 Image Classification with ConvolutionalNeural Networks For Embedded Systems
3. Springenberg, Dosovitskiy, Brox, Riedmiller. STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET
4. https://www.machinecurve.com/index.php/2020/02/09/how-to-build-a-convnet-for-cifar-10-and-cifar-100-

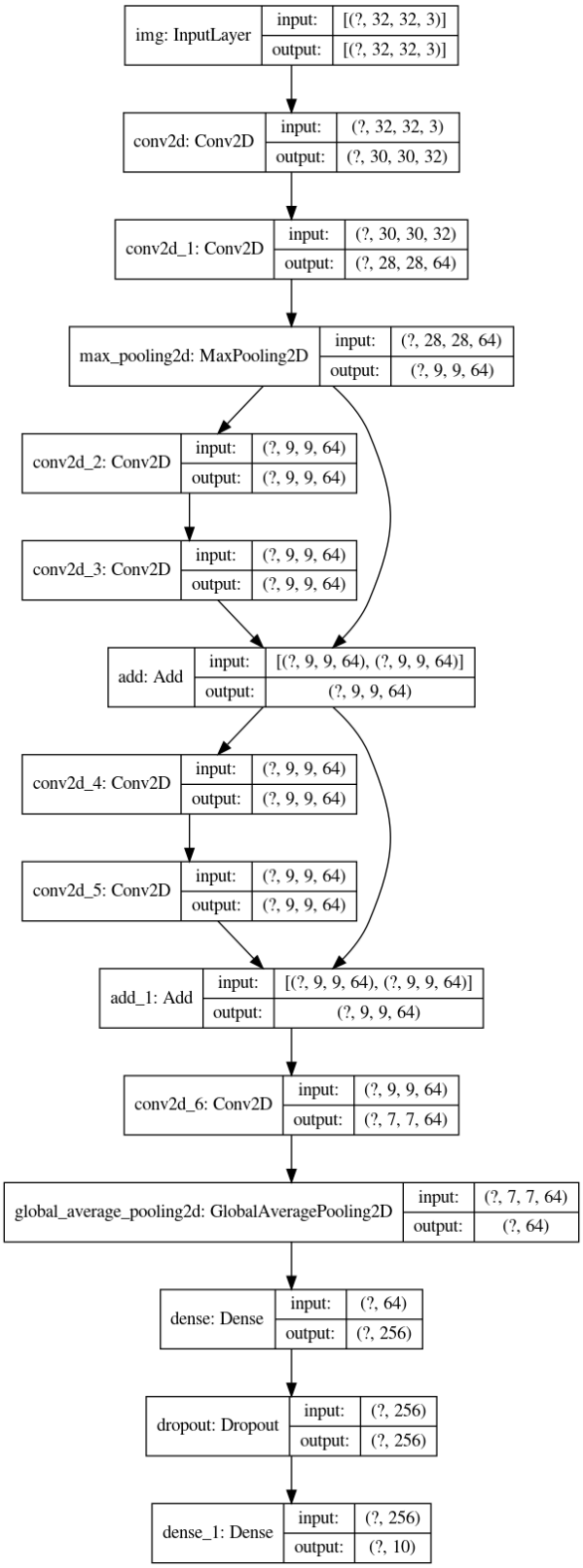

Figure 1: CIFAR-10 ResNet Structure

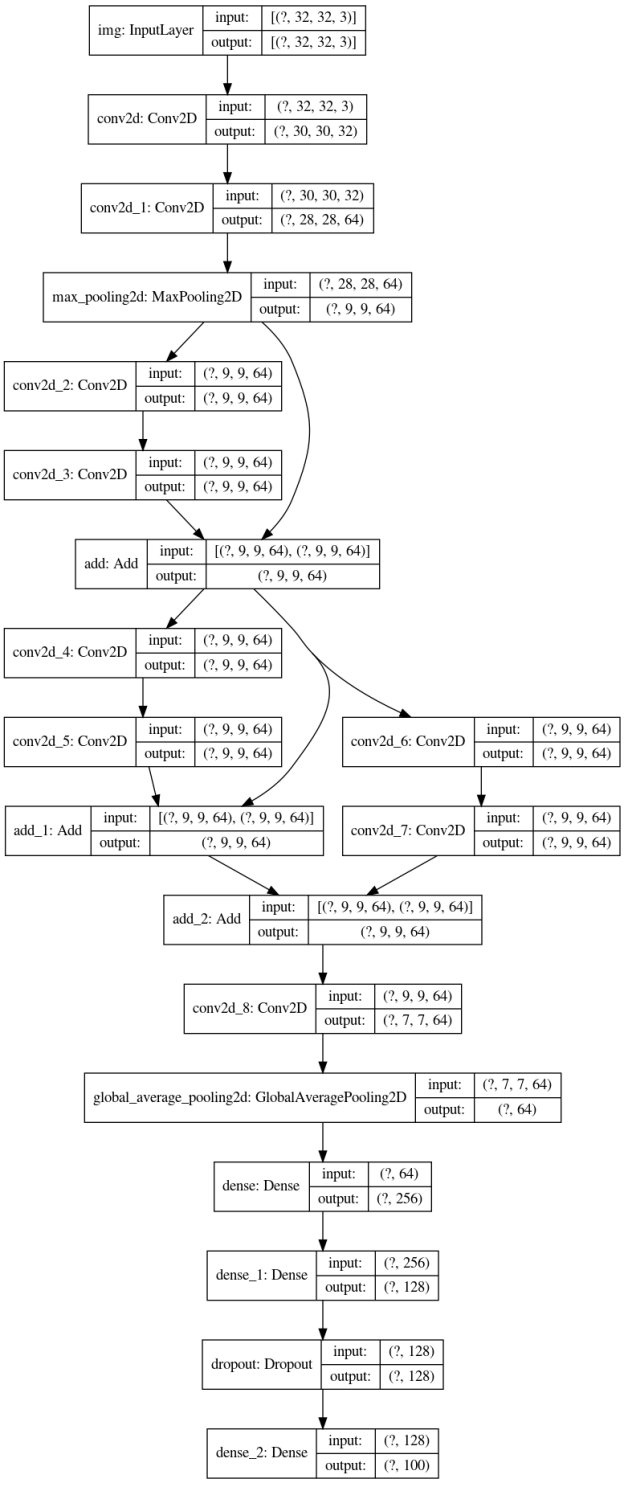

Figure 2: CIFAR-100 ResNet Structure

5 CIFAR-10 Model

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers
import matplotlib.pyplot as plt

# use functional model because Keras says so
# who am I to say otherwise

# Input constructs tensor
def model1():
    inputs = keras.Input(shape=(32, 32, 3), name="img")
    #x = layers.Dropout(0.2)(inputs)
    x = layers.Conv2D(32, 3, activation="elu", bias_regularizer=regularizers.
        l2(1e-3))(inputs)
    x = layers.Conv2D(64, 3, activation="elu")(x)
    block_1_output = layers.MaxPooling2D(3)(x)

    x = layers.Conv2D(64, 3, activation="elu", padding="same")(block_1_output)
    x = layers.Conv2D(64, 3, activation="elu", padding="same",
        bias_regularizer=regularizers.l2(1e-4))(x)
    block_2_output = layers.add([x, block_1_output])

    x = layers.Conv2D(64, 3, activation="elu", padding="same")(block_2_output)
    x = layers.Conv2D(64, 3, activation="elu", padding="same",
        bias_regularizer=regularizers.l2(1e-4))(x)
    block_3_output = layers.add([x, block_2_output])

    x = layers.Conv2D(64, 3, activation="elu", bias_regularizer=regularizers.
        l2(1e-4))(block_3_output)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(256, activation="relu")(x)
    #x = layers.Dense(128, activation="relu")(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(10)(x)
    return inputs, outputs

def model2():
    inputs = keras.Input(shape=(32, 32, 3), name="img")
    x = layers.Conv2D(3, 9, strides=1, activation="relu", padding='valid')(
        inputs)
    #there may be a pooling layer before the first conv
    x = layers.MaxPooling2D((2,2), strides=2)(x)

    x = layers.Conv2D(3, 5, strides=1, activation="relu", padding='same')(x)
    x = layers.MaxPooling2D((2,2), strides=2)(x)

    x = layers.Conv2D(3, 5, strides=1, activation="relu", padding='same')(x)
    x = layers.MaxPooling2D((2,2), strides=2)(x)
```

```
    x = layers.Conv2D(3, 5, strides=1, activation="relu", padding='same')(x)
    x = layers.MaxPooling2D((2,2), strides=2)(x)

    x = layers.Dense(384, activation="relu")(x)
    x = layers.Dense(128, activation="relu")(x)
    outputs = layers.Dense(10)(x)
    return inputs, outputs

def model3():
    inputs = keras.Input(shape=(32, 32, 3), name="img")
    #add dropout here
    x = layers.Dropout(0.2)(inputs)
    x=inputs
    x = layers.Conv2D(96, 5, activation="relu")(x)
    x = layers.MaxPooling2D(3, strides=2)(x)
    x = layers.Dropout(0.5)(x)

    x = layers.Conv2D(192, 5, activation="relu")(x)
    x = layers.MaxPooling2D(3, strides=2)(x)
    x = layers.Dropout(0.5)(x)

    x = layers.Conv2D(192, 3, activation="relu")(x)
    x = layers.Conv2D(192, 1, activation="relu")(x)
    x = layers.Conv2D(10, 1, activation="relu")(x)
    x = layers.GlobalAveragePooling2D()(x)
    outputs = layers.Dense(10)(x)
    return inputs, outputs

model = keras.Model(*model1(), name="MyModel")
model.summary()

# this needs additional packages
keras.utils.plot_model(model, "cifar10_model.png", show_shapes=True)

#model.compile(optimizer='adam',
#              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=
    True),
#              metrics=['accuracy'])

#history = model.fit(train_images, train_labels, epochs=10,
#                    validation_data=(test_images, test_labels), batch_size
    =1024)



#(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
import LoadCIFAR
(x_train, y_train), (x_test, y_test), meta = LoadCIFAR.CIFAR_2D(*LoadCIFAR.
    load_CIFAR10())
```

```
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

top5 = tf.keras.metrics.SparseTopKCategoricalAccuracy(5, name='top5_acc')
#accuracy = tf.keras.metrics.SparseCategoricalCrossentropy(name='acc')

opt = keras.optimizers.Adam()
#0.01

# metrics are not the same as loss functions though loss functions may be
# used as metrics
model.compile(
    #optimizer=keras.optimizers.RMSprop(1e-3),
    optimizer=opt,
    loss=keras.losses.SparseCategoricalCrossentropy(),
    metrics=[top5, 'acc'],
)

history = model.fit(x_train, y_train, batch_size=256, epochs=30,
    validation_split=0.2)
#history = model.fit(x_train, y_train, epochs=20, validation_split=0.2)

model.save('cifar10_model')

model.evaluate(x_test, y_test, verbose=2)

#print(history.history.keys())

plt.clf()
plt.plot(history.history['acc'], label='Training Acc')
plt.plot(history.history['val_acc'], label='Validation Acc')
plt.plot(history.history['top5_acc'], label='Training Acc 5')
plt.plot(history.history['val_top5_acc'], label='Validation Acc 5')
plt.legend()
plt.title("Accuracy over time")
plt.xlabel('Epochs'); plt.ylabel('Accuracy')
plt.savefig('cifar10_history.pdf', format='pdf')
plt.show()
```

6 CIFAR-100 Model

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers
import matplotlib.pyplot as plt


def model1():
    inputs = keras.Input(shape=(32, 32, 3), name="img")
    #add dropout here
    x = layers.Dropout(0.2)(inputs)
    #x=inputs
    x = layers.Conv2D(96, 5, activation="relu")(x)
    x = layers.MaxPooling2D(3, strides=2)(x)
    #x = layers.Dropout(0.5)(x)

    x = layers.Conv2D(192, 5, activation="relu")(x)
    x = layers.MaxPooling2D(3, strides=2)(x)
    #x = layers.Dropout(0.5)(x)

    x = layers.Conv2D(192, 3, activation="relu")(x)
    x = layers.Conv2D(192, 1, activation="relu")(x)
    x = layers.Conv2D(100, 1, activation="relu")(x)
    x = layers.GlobalAveragePooling2D()(x)
    # Do I really need softmax here?
    outputs = layers.Dense(100, activation="softmax")(x)
    return inputs, outputs

def model2():
    inputs = keras.Input(shape=(32, 32, 3), name="img")
    x = layers.Conv2D(32, 3, activation="relu")(inputs)
    x = layers.Conv2D(64, 3, activation="relu")(x)
    block_1_output = layers.MaxPooling2D(3)(x)

    x = layers.Conv2D(64, 3, activation="relu", padding="same")(block_1_output
        )
    x = layers.Conv2D(64, 3, activation="relu", padding="same",
        activity_regularizer=regularizers.l2(1e-4))(x)
    block_2_output = layers.add([x, block_1_output])

    x = layers.Conv2D(64, 3, activation="relu", padding="same")(block_2_output
        )
    x = layers.Conv2D(64, 3, activation="relu", padding="same",
        activity_regularizer=regularizers.l2(1e-4))(x)
    block_3_output = layers.add([x, block_2_output])

    x = layers.Conv2D(64, 3, activation="relu", padding="same")(block_2_output
        )
    x = layers.Conv2D(64, 3, activation="relu", padding="same",
```

```
        activity_regularizer=regularizers.l2(1e-4))(x)
    block_4_output = layers.add([x, block_3_output])

    x = layers.Conv2D(64, 3, activation="relu")(block_4_output)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(256, activation="relu")(x)
    x = layers.Dense(128, activation="relu")(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(100, activation='softmax')(x)
    return inputs, outputs

def model3():
    inputs = keras.Input(shape=(32, 32, 3), name="img")
    x = layers.Conv2D(32, kernel_size=(3, 3), activation='relu')(inputs)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(64, kernel_size=(3, 3), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(128, kernel_size=(3, 3), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Flatten()(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(100, activation='softmax')(x)
    return inputs, outputs

model = keras.Model(*model2(), name="MyModel")
model.summary()
#model.compile()

# this needs additional packages
keras.utils.plot_model(model, "cifar100_model.png", show_shapes=True)

(x_train, y_train), (x_test, y_test) = keras.datasets.cifar100.load_data()
#import LoadCIFAR
#(x_train, y_train), (x_test, y_test), labels = LoadCIFAR.CIFAR_2D(*LoadCIFAR.
    load_CIFAR100())

x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

top5 = tf.keras.metrics.SparseTopKCategoricalAccuracy(5, name='top5_acc')
# just use default loss function instead
#accuracy = tf.keras.metrics.SparseCategoricalCrossentropy(name='acc')

opt = keras.optimizers.Adam()

# metrics are not the same as loss functions though loss functions may be
# used as metrics
model.compile(
    #optimizer=keras.optimizers.RMSprop(1e-3),
```

```
    optimizer=opt,
    loss=keras.losses.SparseCategoricalCrossentropy(),
    metrics=[top5, 'acc'],
)

history = model.fit(x_train, y_train, batch_size=256, epochs=30,
    validation_split=0.2)
#history = model.fit(x_train, y_train, epochs=20, validation_split=0.2)

model.save('cifar100_model')

model.evaluate(x_test, y_test, verbose=2)

plt.clf()
plt.plot(history.history['acc'], label='Training Acc')
plt.plot(history.history['val_acc'], label='Validation Acc')
plt.plot(history.history['top5_acc'], label='Training Acc 5')
plt.plot(history.history['val_top5_acc'], label='Validation Acc 5')
plt.legend()
plt.title("Accuracy over time")
plt.xlabel('Epochs'); plt.ylabel('Accuracy')
plt.savefig('cifar100_history.pdf', format='pdf')
plt.show()
```

7 CIFAR Dataset Loader

```
import numpy as np

def unpickle(fname):
    import pickle
    with open(fname, 'rb') as f:
        return pickle.load(f, encoding='bytes')

def load_CIFAR10():
    path = 'cifar-10-batches-py/data_batch_'
    data = np.array([]).reshape(0,3072)
    labels = np.array([], dtype='int32')
    for i in range(1,6):
        batch = unpickle(path + str(i))
        data = np.concatenate((data, batch[b'data']), axis=0)
        labels = np.concatenate((labels, batch[b'labels']), axis=0)

    test = unpickle('cifar-10-batches-py/test_batch')
    meta = unpickle('cifar-10-batches-py/batches.meta')
    # for whatever reason test labels are regular array, need to be wrapped
    te_data = test[b'data']
    te_labels = np.array(test[b'labels']).astype('int32')
    return (data, labels), (te_data, te_labels), meta[b'label_names']


def load_CIFAR100():
    test  = unpickle('cifar-100-python/test')
    train = unpickle('cifar-100-python/train')
    meta  = unpickle('cifar-100-python/meta')

    test_d = test[b'data'], np.array(test[b'fine_labels']).astype('int32')
    train_d = train[b'data'], np.array(train[b'fine_labels']).astype('int32')
    meta_d = meta[b'fine_label_names']

    return train_d, test_d, meta_d

def CIFAR_2D(train,test,meta):
    (train_x,train_y) = train
    (test_x,test_y) = test
    train_x = train_x.reshape(-1,3,1024).transpose([0,2,1]).reshape
        (-1,32,32,3)
    test_x = test_x.reshape(-1,3,1024).transpose([0,2,1]).reshape(-1,32,32,3)
    return (train_x, train_y), (test_x, test_y), meta

if __name__ == '__main__':
    import matplotlib.pyplot as plt
    _, (train_x,train_y), labels = CIFAR_2D(*load_CIFAR100())
    plt.figure(figsize=(10,10))
    for i in range(25):
        plt.subplot(5,5,i+1)
```

```
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_x[i])
    plt.xlabel(labels[train_y[i]].decode('utf-8'))
plt.suptitle('CIFAR-100 Train Sample')
plt.show()
_, (train_x,train_y), labels = CIFAR_2D(*load_CIFAR10())
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_x[i])
    plt.xlabel(labels[train_y[i]].decode('utf-8'))
plt.suptitle('CIFAR-10 Train Sample')
plt.show()
```