6.2 Assignment

October 11, 2020

0.0.1 Week6: Computer Vision

0.0.2 6.2 Assignment

a.

Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classifies images CIFAR10 small images classification dataset. Do not use dropout or data-augmentation in this part. Save the model, predictions, metrics, and validation plots in the dsc650/assignments/assignment06/results directory.

b.

Using section 5.2 in Deep Learning with Python as a guide, create a ConvNet model that classifies images CIFAR10 small images classification dataset. This time includes dropout and data-augmentation. Save the model, predictions, metrics, and validation plots in the dsc650/assignments/assignment06/results directory.

https://bellevue-university.github.io/dsc650/lessons/12-week/week06/

[110]:
```
import keras
keras.__version__
```

[110]:
```
'2.4.3'
```

[111]:
```
import os, shutil
```

[112]:
```
#def LRN2D(x):
    #import tensorflow as tf
    #return tf.nn.lrn(x, alpha=1e-4, beta=0.75)
```

[113]:
```
from keras import models, layers
import tensorflow as tf
(trainx, trainy), (testx, testy) = tf.keras.datasets.cifar10.load_data()
print(type(trainx))
```

```
<class 'numpy.ndarray'>
```

As a sanity check, let's count how many pictures we have in each training split (train/validation/test):

So we have indeed 2000 training images, and then 1000 validation images and 1000 test images. In each split, there is the same number of samples from each class: this is a balanced binary classification problem, which means that classification accuracy will be an appropriate measure of success. ### Building our network

We've already built a small convnet for MNIST in the previous example, so you should be familiar with them. We will reuse the same general structure: our convnet will be a stack of alternated Conv2D (with relu activation) and MaxPooling2D layers.

However, since we are dealing with bigger images and a more complex problem, we will make our network accordingly larger: it will have one more Conv2D + MaxPooling2D stage. This serves both to augment the capacity of the network, and to further reduce the size of the feature maps, so that they aren't overly large when we reach the Flatten layer. Here, since we start from inputs of size 150x150 (a somewhat arbitrary choice), we end up with feature maps of size 7x7 right before the Flatten layer.

Note that the depth of the feature maps is progressively increasing in the network (from 32 to 128), while the size of the feature maps is decreasing (from 148x148 to 7x7). This is a pattern that you will see in almost all convnets.

Since we are attacking a binary classification problem, we are ending the network with a single unit (a Dense layer of size 1) and a sigmoid activation. This unit will encode the probability that the network is looking at one class or the other.

https://github.com/fchollet/deep-learning-with-python-notebooks/issues/16

[114]:
```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
#model.add(layers.MaxPooling2D((2, 2)))
#model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='sigmoid'))
```

[115]:
```
from keras import optimizers

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

[116]:
```
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow(
        # This is the target directory
        trainx,
        trainy,
        batch_size=20)

validation_generator = test_datagen.flow(
        # This is the target directory
        testx,
        testy,
        batch_size=20)
```

[117]:
```
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
```

```
data batch shape: (20, 32, 32, 3)
labels batch shape: (20, 1)
```

[118]:
```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

```
Epoch 1/30
100/100 [==============================] - 2s 17ms/step - loss: 103.2029 - acc:
0.0760 - val_loss: 107.0017 - val_acc: 0.0020
Epoch 2/30
100/100 [==============================] - 1s 14ms/step - loss: 106.4257 - acc:
0.0925 - val_loss: 103.0178 - val_acc: 0.3960
Epoch 3/30
100/100 [==============================] - 1s 14ms/step - loss: 103.5934 - acc:
0.0880 - val_loss: 105.8729 - val_acc: 0.0570
Epoch 4/30
100/100 [==============================] - 1s 14ms/step - loss: 104.6295 - acc:
0.0935 - val_loss: 104.6065 - val_acc: 0.0870
Epoch 5/30
100/100 [==============================] - 1s 14ms/step - loss: 105.7002 - acc:
0.0815 - val_loss: 104.4453 - val_acc: 0.0810
```

```
Epoch 6/30
100/100 [==============================] - 1s 14ms/step - loss: 102.6723 - acc:
0.0920 - val_loss: 106.2643 - val_acc: 0.0000e+00
Epoch 7/30
100/100 [==============================] - 1s 14ms/step - loss: 104.0768 - acc:
0.1330 - val_loss: 103.9156 - val_acc: 0.0080
Epoch 8/30
100/100 [==============================] - 1s 15ms/step - loss: 102.7183 - acc:
0.1215 - val_loss: 99.8401 - val_acc: 0.0000e+00
Epoch 9/30
100/100 [==============================] - 1s 14ms/step - loss: 103.4436 - acc:
0.0880 - val_loss: 103.9387 - val_acc: 0.0000e+00
Epoch 10/30
100/100 [==============================] - 1s 14ms/step - loss: 102.2118 - acc:
0.0715 - val_loss: 104.0308 - val_acc: 0.0000e+00
Epoch 11/30
100/100 [==============================] - 1s 14ms/step - loss: 101.3713 - acc:
0.0910 - val_loss: 104.7216 - val_acc: 0.0000e+00
Epoch 12/30
100/100 [==============================] - 1s 13ms/step - loss: 102.2233 - acc:
0.0905 - val_loss: 104.3762 - val_acc: 0.0000e+00
Epoch 13/30
100/100 [==============================] - 1s 14ms/step - loss: 102.9947 - acc:
0.1005 - val_loss: 107.2314 - val_acc: 0.9770
Epoch 14/30
100/100 [==============================] - 1s 14ms/step - loss: 105.1936 - acc:
0.1215 - val_loss: 104.5834 - val_acc: 0.0000e+00
Epoch 15/30
100/100 [==============================] - 1s 14ms/step - loss: 102.6723 - acc:
0.1190 - val_loss: 98.1362 - val_acc: 0.0030
Epoch 16/30
100/100 [==============================] - 1s 15ms/step - loss: 104.0883 - acc:
0.1385 - val_loss: 105.4123 - val_acc: 0.0000e+00
Epoch 17/30
100/100 [==============================] - 1s 15ms/step - loss: 104.4222 - acc:
0.1200 - val_loss: 101.7282 - val_acc: 0.0000e+00
Epoch 18/30
100/100 [==============================] - 1s 14ms/step - loss: 105.4584 - acc:
0.1110 - val_loss: 105.3202 - val_acc: 0.0000e+00
Epoch 19/30
100/100 [==============================] - 1s 14ms/step - loss: 102.7989 - acc:
0.1105 - val_loss: 104.5374 - val_acc: 0.0000e+00
Epoch 20/30
100/100 [==============================] - 1s 14ms/step - loss: 102.7298 - acc:
0.1300 - val_loss: 101.3828 - val_acc: 0.0010
Epoch 21/30
100/100 [==============================] - 1s 14ms/step - loss: 101.7742 - acc:
0.1300 - val_loss: 105.0900 - val_acc: 0.0000e+00
```

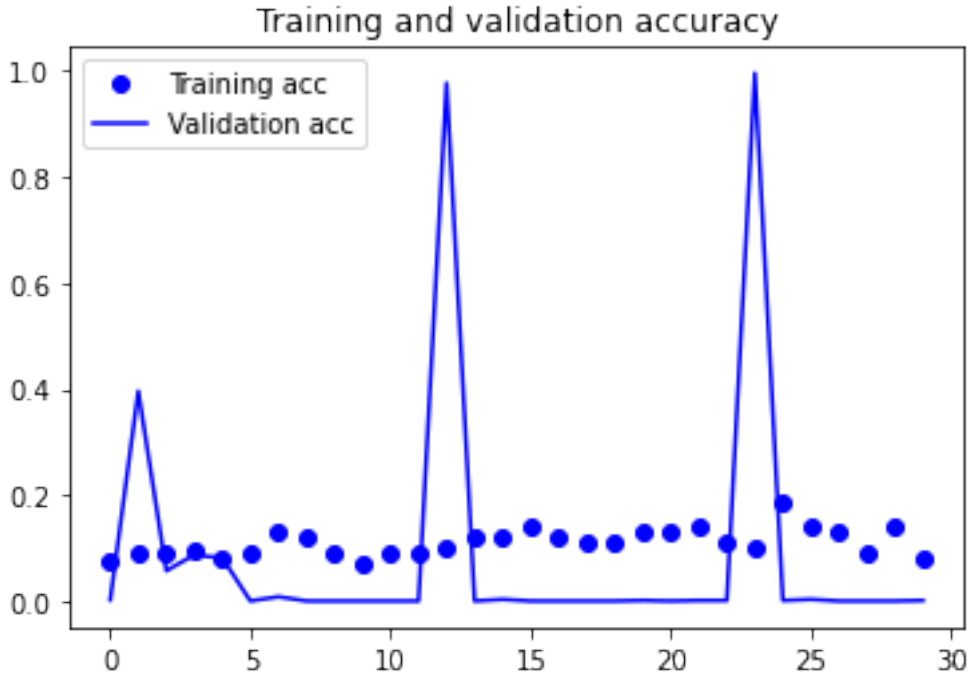
```
Epoch 22/30
100/100 [==============================] - 1s 15ms/step - loss: 104.0193 - acc:
0.1410 - val_loss: 104.8827 - val_acc: 0.0010
Epoch 23/30
100/100 [==============================] - 1s 14ms/step - loss: 104.0308 - acc:
0.1080 - val_loss: 102.4881 - val_acc: 0.0010
Epoch 24/30
100/100 [==============================] - 1s 14ms/step - loss: 102.7528 - acc:
0.0985 - val_loss: 107.8301 - val_acc: 0.9960
Epoch 25/30
100/100 [==============================] - 1s 13ms/step - loss: 104.6755 - acc:
0.1885 - val_loss: 104.7216 - val_acc: 0.0010
Epoch 26/30
100/100 [==============================] - 1s 15ms/step - loss: 104.1114 - acc:
0.1395 - val_loss: 100.7842 - val_acc: 0.0030
Epoch 27/30
100/100 [==============================] - 1s 14ms/step - loss: 106.3219 - acc:
0.1295 - val_loss: 101.4059 - val_acc: 0.0000e+00
Epoch 28/30
100/100 [==============================] - 1s 14ms/step - loss: 103.9272 - acc:
0.0895 - val_loss: 102.2808 - val_acc: 0.0000e+00
Epoch 29/30
100/100 [==============================] - 2s 19ms/step - loss: 103.4321 - acc:
0.1385 - val_loss: 103.1558 - val_acc: 0.0000e+00
Epoch 30/30
100/100 [==============================] - 1s 15ms/step - loss: 102.0391 - acc:
0.0800 - val_loss: 106.4946 - val_acc: 0.0010
```

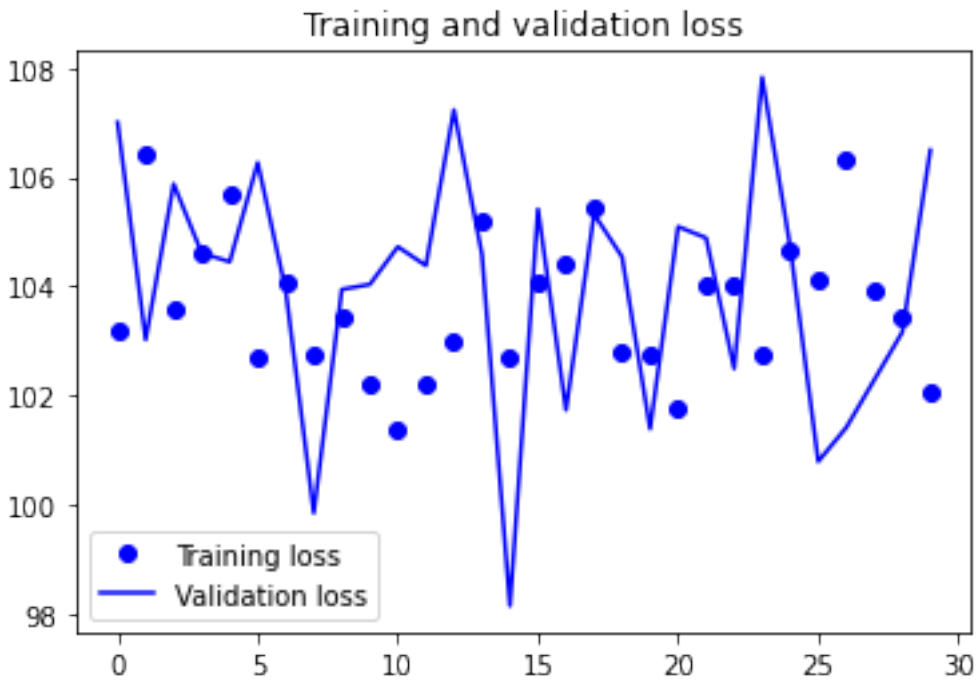
It is good practice to always save your models after training:

[119]:
```
model.save('things-10.h5')
```

[120]:
```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
```

```
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





[129]:
```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

[ ]:
```

```