6.2a

July 13, 2021

1 Assignment 6.2a

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. If you are using JupyterHub, you can include those plots in your Jupyter notebook.

[1]:
```
from keras.datasets import cifar10
from keras.utils import to_categorical
import pandas as pd
```

[2]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 18s 0us/step
```

[3]:
```
x_train.shape, y_train.shape
```

[3]:
```
((50000, 32, 32, 3), (50000, 1))
```

[4]:
```
x_test.shape, y_test.shape
```

[4]:
```
((10000, 32, 32, 3), (10000, 1))
```

[5]:
```
# preprocess data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# put 10,000 aside for validation
x_val = x_train[-10000:]
y_val = y_train[-10000:]
x_train = x_train[:-10000]
y_train = y_train[:-10000]
```

[6]:
```
x_val.shape, y_val.shape
```

[6]:
```
((10000, 32, 32, 3), (10000, 10))
```

[7]:
```
from keras import models
from keras import layers
```

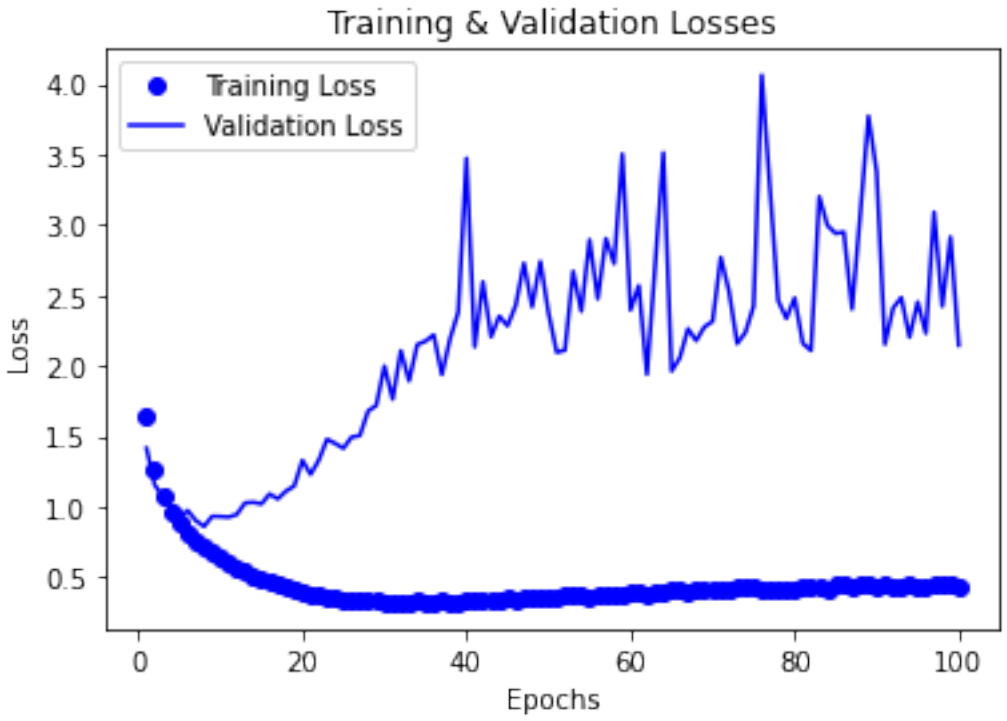
[8]:
```
# instantiate the model
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation = 'softmax'))

# view summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 64)                16448
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 73,418
Trainable params: 73,418
Non-trainable params: 0
_________________________________________________________________
```

[9]:
```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=100,
                    validation_data = (x_val, y_val), verbose=0)
```

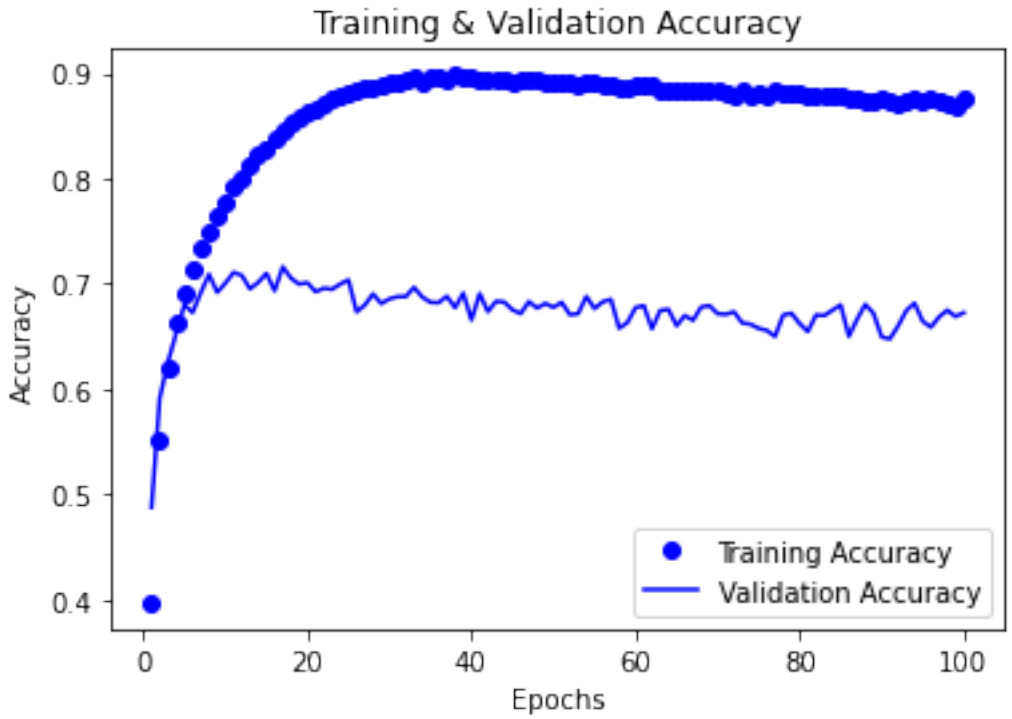
[10]:
```
import matplotlib.pyplot as plt
```

[11]:
```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(history.history['loss']) + 1)

plt.plot(epochs, train_loss, 'bo', label = 'Training Loss')
plt.plot(epochs, val_loss, 'b', label = 'Validation Loss')
plt.title('Training & Validation Losses')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
plt.savefig('results/6_2A_Loss.png')
```




```
<Figure size 432x288 with 0 Axes>
```

[12]:
```
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epcohs = range(1, len(history.history['accuracy']) + 1)

plt.plot(epochs, train_acc, 'bo', label = 'Training Accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation Accuracy')
plt.title('Training & Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
plt.savefig('results/6_2A_Accuracy')
```




```
<Figure size 432x288 with 0 Axes>
```

[13]:
```
# retrain model & evaluate
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# preprocess data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=10)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/10
1563/1563 [==============================] - 17s 10ms/step - loss: 0.7951 -
accuracy: 0.7852
Epoch 2/10
1563/1563 [==============================] - 16s 10ms/step - loss: 0.6774 -
accuracy: 0.7976
Epoch 3/10
1563/1563 [==============================] - 16s 10ms/step - loss: 0.6282 -
accuracy: 0.8066
Epoch 4/10
1563/1563 [==============================] - 16s 10ms/step - loss: 0.6002 -
accuracy: 0.8117
Epoch 5/10
1563/1563 [==============================] - 16s 10ms/step - loss: 0.5938 -
accuracy: 0.8113
Epoch 6/10
1563/1563 [==============================] - 16s 10ms/step - loss: 0.5676 -
accuracy: 0.8212
Epoch 7/10
1563/1563 [==============================] - 16s 10ms/step - loss: 0.5466 -
accuracy: 0.8238
Epoch 8/10
1563/1563 [==============================] - 16s 10ms/step - loss: 0.5554 -
accuracy: 0.8224
Epoch 9/10
1563/1563 [==============================] - 16s 10ms/step - loss: 0.5545 -
accuracy: 0.8162
Epoch 10/10
1563/1563 [==============================] - 16s 10ms/step - loss: 0.5507 -
accuracy: 0.8233
313/313 [==============================] - 1s 4ms/step - loss: 1.5735 -
accuracy: 0.6725
```

[14]:
```
model.save('results/6_2A_model.h5')
```

[15]:
```
prediction_results = model.predict(x_test)
```

[16]:
```
prediction_results
```

[16]:
```
array([[1.45420117e-05, 4.81691586e-06, 3.56239570e-06, …,
        2.17049255e-05, 1.09994132e-02, 1.62153709e-04],
       [8.13679828e-04, 1.10762246e-01, 1.37421219e-14, …,
        3.79440843e-14, 8.88385475e-01, 3.86396896e-05],
       [4.37420234e-03, 1.20730232e-02, 3.81997634e-05, …,
        1.58175232e-03, 9.59185123e-01, 1.21063441e-02],
       …,
       [3.28650163e-11, 5.68203995e-16, 3.56616511e-07, …,
        1.60563843e-06, 3.80617282e-12, 7.23070701e-11],
       [1.07275538e-01, 1.57782082e-02, 5.50686538e-01, …,
        2.40683369e-03, 2.12981395e-05, 1.66484058e-01],
       [0.00000000e+00, 0.00000000e+00, 2.00358356e-25, …,
        1.00000000e+00, 0.00000000e+00, 0.00000000e+00]], dtype=float32)
```

[17]:
```
with open('results/6_2A_metrics.txt', 'w') as f:
    f.write('Training Loss: {}'.format(str(history.history['loss'])))
    f.write('\nTraining Accuracy: {}'.format(str(history.history['accuracy'])))
    f.write('\nTest Loss: {}'.format(results[0]))
    f.write('\nTest Accuracy: {}'.format(results[1]))
```

[18]:
```
preds = pd.DataFrame(prediction_results,
                     columns = ['0','1','2','3','4','5','6','7','8','9'])

preds.to_csv('results/6_2A_predicitons.csv', index = False)
```

[ ]:
```

```