6.2b

July 13, 2021

1 Assignment 6.2b

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

[2]:
```
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
import pandas as pd
import matplotlib.pyplot as plt
```

[3]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

[4]:
```
x_train.shape, y_train.shape
```

[4]:
```
((50000, 32, 32, 3), (50000, 1))
```

[5]:
```
x_test.shape, y_test.shape
```

[5]:
```
((10000, 32, 32, 3), (10000, 1))
```

[6]:
```
# preprocess data
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
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_train2 = x_train[:-10000]
y_train2 = y_train[:-10000]
```

[7]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
```

```
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow(x_train2, y_train2, batch_size=32)

validation_generator = train_datagen.flow(x_val, y_val, batch_size=32)
```

[8]:
```
from keras import models
from keras import layers
```

[9]:
```
# instantiate the model
# add dropout layer
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.Dropout(0.5))
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
_________________________________________________________________
dropout (Dropout)            (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
_________________________________________________________________
```

[10]:
```
from keras import optimizers
```

[11]:
```
model.compile(optimizer=optimizers.RMSprop(lr=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

[12]:
```
history = model.fit_generator(train_generator,
                              steps_per_epoch=len(x_train2) / 32,
                              epochs = 30,
                              validation_data=validation_generator,
                              validation_steps=len(x_val) / 32)
```

```
/opt/conda/lib/python3.8/site-
packages/tensorflow/python/keras/engine/training.py:1844: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
```

```
Epoch 1/30
1250/1250 [==============================] - 44s 35ms/step - loss: 2.2275 -
accuracy: 0.1532 - val_loss: 1.9763 - val_accuracy: 0.2590
Epoch 2/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.9872 -
accuracy: 0.2551 - val_loss: 1.9239 - val_accuracy: 0.2812
Epoch 3/30
1250/1250 [==============================] - 43s 35ms/step - loss: 1.9184 -
accuracy: 0.2807 - val_loss: 1.8443 - val_accuracy: 0.3066
Epoch 4/30
1250/1250 [==============================] - 43s 35ms/step - loss: 1.8564 -
accuracy: 0.3020 - val_loss: 1.8026 - val_accuracy: 0.3318
Epoch 5/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.8197 -
accuracy: 0.3224 - val_loss: 1.7838 - val_accuracy: 0.3453
Epoch 6/30
```

```
1250/1250 [==============================] - 43s 34ms/step - loss: 1.7992 -
accuracy: 0.3302 - val_loss: 1.7359 - val_accuracy: 0.3638
Epoch 7/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.7624 -
accuracy: 0.3462 - val_loss: 1.7389 - val_accuracy: 0.3684
Epoch 8/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.7322 -
accuracy: 0.3588 - val_loss: 1.6830 - val_accuracy: 0.3871
Epoch 9/30
1250/1250 [==============================] - 43s 35ms/step - loss: 1.7167 -
accuracy: 0.3698 - val_loss: 1.6856 - val_accuracy: 0.3854
Epoch 10/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.6997 -
accuracy: 0.3772 - val_loss: 1.6819 - val_accuracy: 0.3895
Epoch 11/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.6839 -
accuracy: 0.3857 - val_loss: 1.6256 - val_accuracy: 0.4099
Epoch 12/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.6568 -
accuracy: 0.3973 - val_loss: 1.6224 - val_accuracy: 0.4113
Epoch 13/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.6414 -
accuracy: 0.4020 - val_loss: 1.6037 - val_accuracy: 0.4183
Epoch 14/30
1250/1250 [==============================] - 42s 34ms/step - loss: 1.6364 -
accuracy: 0.4049 - val_loss: 1.5827 - val_accuracy: 0.4258
Epoch 15/30
1250/1250 [==============================] - 43s 35ms/step - loss: 1.6149 -
accuracy: 0.4106 - val_loss: 1.5742 - val_accuracy: 0.4337
Epoch 16/30
1250/1250 [==============================] - 43s 35ms/step - loss: 1.6177 -
accuracy: 0.4142 - val_loss: 1.5416 - val_accuracy: 0.4440
Epoch 17/30
1250/1250 [==============================] - 43s 35ms/step - loss: 1.6007 -
accuracy: 0.4180 - val_loss: 1.5630 - val_accuracy: 0.4442
Epoch 18/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.5854 -
accuracy: 0.4289 - val_loss: 1.5411 - val_accuracy: 0.4532
Epoch 19/30
1250/1250 [==============================] - 42s 34ms/step - loss: 1.5753 -
accuracy: 0.4300 - val_loss: 1.5057 - val_accuracy: 0.4593
Epoch 20/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.5696 -
accuracy: 0.4346 - val_loss: 1.5134 - val_accuracy: 0.4594
Epoch 21/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.5556 -
accuracy: 0.4390 - val_loss: 1.4859 - val_accuracy: 0.4671
Epoch 22/30
```

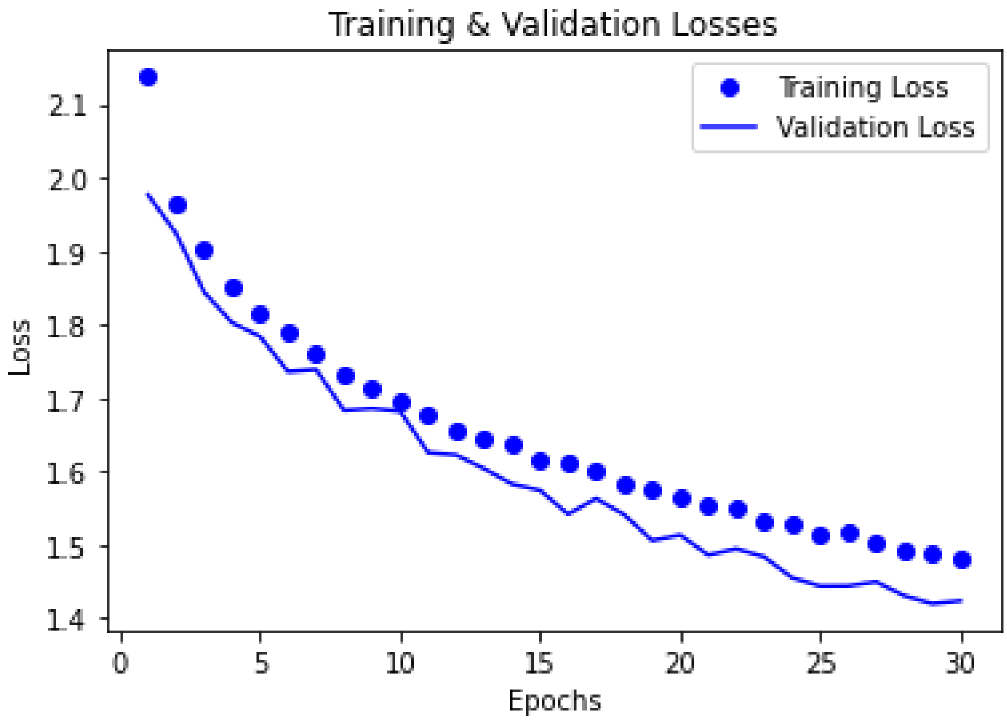
```
1250/1250 [==============================] - 43s 34ms/step - loss: 1.5516 -
accuracy: 0.4496 - val_loss: 1.4945 - val_accuracy: 0.4653
Epoch 23/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.5313 -
accuracy: 0.4507 - val_loss: 1.4835 - val_accuracy: 0.4711
Epoch 24/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.5294 -
accuracy: 0.4515 - val_loss: 1.4549 - val_accuracy: 0.4774
Epoch 25/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.5158 -
accuracy: 0.4554 - val_loss: 1.4437 - val_accuracy: 0.4837
Epoch 26/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.5110 -
accuracy: 0.4578 - val_loss: 1.4439 - val_accuracy: 0.4864
Epoch 27/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.5045 -
accuracy: 0.4573 - val_loss: 1.4495 - val_accuracy: 0.4853
Epoch 28/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.4981 -
accuracy: 0.4616 - val_loss: 1.4305 - val_accuracy: 0.4903
Epoch 29/30
1250/1250 [==============================] - 43s 34ms/step - loss: 1.4855 -
accuracy: 0.4697 - val_loss: 1.4197 - val_accuracy: 0.5021
Epoch 30/30
1250/1250 [==============================] - 43s 35ms/step - loss: 1.4827 -
accuracy: 0.4669 - val_loss: 1.4237 - val_accuracy: 0.4965
```

[13]:

```
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_2B_Loss.png')
```

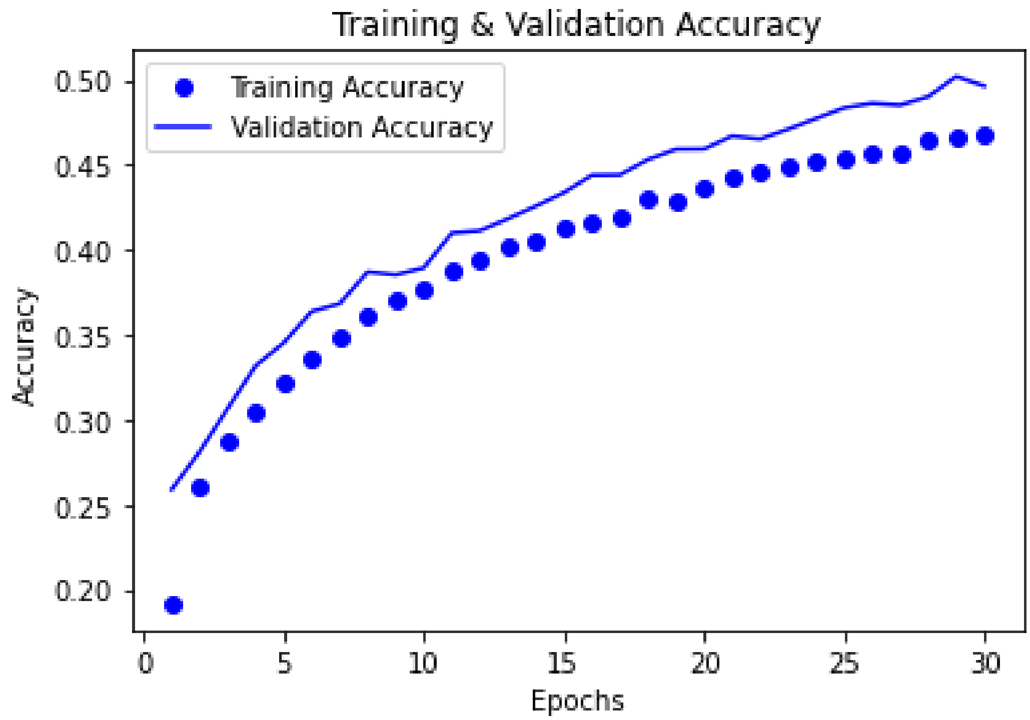


<Figure size 432x288 with 0 Axes>

[14]:
```
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_2B_Accuracy')
```




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

[15]:
```
# retrain model & evaluate
train_generator = train_datagen.flow(x_train, y_train, batch_size=32)

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

history = model.fit_generator(train_generator,
                              steps_per_epoch=len(x_train) / 32,
                              epochs = 16)

results = model.evaluate(x_test, y_test)
```

```
Epoch 1/16
1562/1562 [==============================] - 45s 28ms/step - loss: 1.4893 -
accuracy: 0.4694
Epoch 2/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4692 -
accuracy: 0.4742
Epoch 3/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4655 -
accuracy: 0.4750
```

```
Epoch 4/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4514 -
accuracy: 0.4830
Epoch 5/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4534 -
accuracy: 0.4832
Epoch 6/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4350 -
accuracy: 0.4857
Epoch 7/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4365 -
accuracy: 0.4890
Epoch 8/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4232 -
accuracy: 0.4934
Epoch 9/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4239 -
accuracy: 0.4897
Epoch 10/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4199 -
accuracy: 0.4930
Epoch 11/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.4094 -
accuracy: 0.4991
Epoch 12/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.3971 -
accuracy: 0.5016
Epoch 13/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.3961 -
accuracy: 0.5012
Epoch 14/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.3908 -
accuracy: 0.5060
Epoch 15/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.3894 -
accuracy: 0.5045
Epoch 16/16
1562/1562 [==============================] - 44s 28ms/step - loss: 1.3856 -
accuracy: 0.5065
313/313 [==============================] - 1s 4ms/step - loss: 239.9552 -
accuracy: 0.3348
```

[16]:
```
model.save('results/6_2B_model.h5')
```

[17]:
```
prediction_results = model.predict(x_test)
```

[18]:
```
prediction_results
```

[18]:
```
array([[0.0000000e+00, 0.0000000e+00, 0.0000000e+00, …, 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [0.0000000e+00, 1.0000000e+00, 0.0000000e+00, …, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [5.8108697e-17, 1.0000000e+00, 0.0000000e+00, …, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       …,
       [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, …, 1.1507828e-14,
        0.0000000e+00, 0.0000000e+00],
       [0.0000000e+00, 1.0000000e+00, 0.0000000e+00, …, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, …, 1.0000000e+00,
        0.0000000e+00, 0.0000000e+00]], dtype=float32)
```

[19]:
```
with open('results/6_2B_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]))
```

[20]:
```
preds = pd.DataFrame(prediction_results,
                     columns = ['0','1','2','3','4','5','6','7','8','9'])

preds.to_csv('results/6_2B_predicitons.csv', index = False)
```

[ ]:
```

```