assignment_6.2a_cifar10

January 9, 2022

0.0.1 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.

[1]:
```
import json
from pathlib import Path
import os

current_dir = Path(os.getcwd()).absolute()
results_dir = current_dir.joinpath('results')

print(current_dir)
print(results_dir)
```

```
C:\Users\saman\git_repos\dsc650\dsc650\assignments\assignment06
C:\Users\saman\git_repos\dsc650\dsc650\assignments\assignment06\results
```

[2]:
```
# loading the required libraries and packages
import sys
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
import matplotlib.pyplot as plt

from keras.datasets import cifar10

from keras.layers import Flatten

from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D

from keras.utils import np_utils
from keras import optimizers
```

```
Using TensorFlow backend.
```

0.1 Data

Here we are using the CIFAR10 small images dataset to classify the images.

This is a dataset of 50,000 32X32 color training images and 10,000 test images labeled over 10 categories. Each class is represented as a unique number

0.1.1 Data Preparation

[38]:
```
# Data preparation is required before training the model
def load_dataset():

        # loading the CIFAR10 dataset and create the training and test arrays
        (X_train, y_train), (X_test, y_test) = cifar10.load_data()

        # Lines 1 and 2 reshapes the inputs
        X_train = X_train.reshape((X_train.shape[0], 32, 32, 3)).
 ↪astype('float32')
        X_test = X_test.reshape((X_test.shape[0], 32, 32, 3)).astype('float32')

        # Lines 3 and 4
        # Normalization of the input values (image pixels) from 0 and 255 to 0.1
        X_train = X_train / 255
        X_test = X_test / 255

        # Lines 5 and 6
        # one-hot encoding of the target variables
        y_train = np_utils.to_categorical(y_train)
        y_test = np_utils.to_categorical(y_test)

        num_classes = y_test.shape[1]

        return X_train, X_test, y_train, y_test
```

[39]:
```
def cnn_model():
        # function to create the CNN model
        # Create model
        model = Sequential()  #model type is sequetial
        # Stacking convolutional layers with small 3 X 3 filters
        # It is followed by a max pooling layer.
        # Each of the above blocks are repeated where the number of filters in␣
 ↪each block is increased.
        # Also the depth of the network such as 32,64 are also increased
        # Rectified Linear Activation ReLu is most widely used. It makes the␣
 ↪network sparse and efficient
```

```
    model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3),␣
↪activation='relu'))
    # model.add(Conv2D(32, (3, 3), activation='relu'))
    # Adding the pooling layer
    model.add(MaxPooling2D())

    model.add(Conv2D(64, (3, 3), activation='relu'))
    # model.add(Conv2D(64, (3, 3), activation='relu'))
    # Adding the pooling layer
    model.add(MaxPooling2D())

    model.add(Conv2D(128, (3, 3), activation='relu'))
    # model.add(Conv2D(128, (3, 3), activation='relu'))
    # Adding the pooling layer
    model.add(MaxPooling2D())

    # Flatten layer converts the 2D matrix data to a vector
    model.add(Flatten())
    # Fully connected dense layer with 128 neurons
    model.add(Dense(128, activation='relu'))
    # output layer which has 10 neurons for the 10 classes
    model.add(Dense(10, activation='softmax'))

    return model
```

[ ]:
```
# Plotting the results
def summary_plot(history):

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

    epochs = range(1, len(acc) + 1)

    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()
```

[40]:
```
# plot diagnostic learning curves
def summarize_diagnostics(history):
        plt.subplot(211)
        plt.title('Cross Entropy Loss')
        plt.plot(history.history['loss'], color='blue', label='train')
        plt.plot(history.history['val_loss'], color='orange', label='test')
        # plot accuracy
        plt.subplot(212)
        plt.title('Classification Accuracy')
        plt.plot(history.history['accuracy'], color='blue', label='train')
        plt.plot(history.history['val_accuracy'], color='orange', label='test')
        # save plot to file
        plt.savefig(f'{results_dir}\\1_plot.png')
        plt.show()
        plt.close()
```

[37]:
```
print(X_train)
# model = cnn_model()
# model.compile(loss='categorical_crossentropy', optimizer='rmsprop',␣
↪metrics=['accuracy'])
```

```
[[[[0.23137255 0.24313726 0.24705882]
   [0.16862746 0.18039216 0.1764706 ]
   [0.19607843 0.1882353  0.16862746]
   …
   [0.61960787 0.5176471  0.42352942]
   [0.59607846 0.49019608 0.4       ]
   [0.5803922  0.4862745  0.40392157]]

  [[0.0627451  0.07843138 0.07843138]
   [0.         0.         0.        ]
   [0.07058824 0.03137255 0.        ]
   …
   [0.48235294 0.34509805 0.21568628]
   [0.46666667 0.3254902  0.19607843]
   [0.47843137 0.34117648 0.22352941]]

  [[0.09803922 0.09411765 0.08235294]
   [0.0627451  0.02745098 0.        ]
   [0.19215687 0.10588235 0.03137255]
   …
   [0.4627451  0.32941177 0.19607843]
   [0.47058824 0.32941177 0.19607843]
   [0.42745098 0.28627452 0.16470589]]


  …
```

```
 [[0.8156863  0.6666667  0.3764706 ]
  [0.7882353  0.6        0.13333334]
  [0.7764706  0.6313726  0.10196079]
  …
  [0.627451   0.52156866 0.27450982]
  [0.21960784 0.12156863 0.02745098]
  [0.20784314 0.13333334 0.07843138]]

 [[0.7058824  0.54509807 0.3764706 ]
  [0.6784314  0.48235294 0.16470589]
  [0.7294118  0.5647059  0.11764706]
  …
  [0.72156864 0.5803922  0.36862746]
  [0.38039216 0.24313726 0.13333334]
  [0.3254902  0.20784314 0.13333334]]

 [[0.69411767 0.5647059  0.45490196]
  [0.65882355 0.5058824  0.36862746]
  [0.7019608  0.5568628  0.34117648]
  …
  [0.84705883 0.72156864 0.54901963]
  [0.5921569  0.4627451  0.32941177]
  [0.48235294 0.36078432 0.28235295]]]


[[[0.6039216  0.69411767 0.73333335]
  [0.49411765 0.5372549  0.53333336]
  [0.4117647  0.40784314 0.37254903]
  …
  [0.35686275 0.37254903 0.2784314 ]
  [0.34117648 0.3529412  0.2784314 ]
  [0.30980393 0.31764707 0.27450982]]

 [[0.54901963 0.627451   0.6627451 ]
  [0.5686275  0.6        0.6039216 ]
  [0.49019608 0.49019608 0.4627451 ]
  …
  [0.3764706  0.3882353  0.30588236]
  [0.3019608  0.3137255  0.24313726]
  [0.2784314  0.28627452 0.23921569]]

 [[0.54901963 0.60784316 0.6431373 ]
  [0.54509807 0.57254905 0.58431375]
  [0.4509804  0.4509804  0.4392157 ]
  …
  [0.30980393 0.32156864 0.2509804 ]
  [0.26666668 0.27450982 0.21568628]
```

```
  [0.2627451  0.27058825 0.21568628]]

 …

 [[0.6862745  0.654902   0.6509804 ]
  [0.6117647  0.6039216  0.627451  ]
  [0.6039216  0.627451   0.6666667 ]
  …
  [0.16470589 0.13333334 0.14117648]
  [0.23921569 0.20784314 0.22352941]
  [0.3647059  0.3254902  0.35686275]]

 [[0.64705884 0.6039216  0.5019608 ]
  [0.6117647  0.59607846 0.50980395]
  [0.62352943 0.6313726  0.5568628 ]
  …
  [0.40392157 0.3647059  0.3764706 ]
  [0.48235294 0.44705883 0.47058824]
  [0.5137255  0.4745098  0.5137255 ]]

 [[0.6392157  0.5803922  0.47058824]
  [0.61960787 0.5803922  0.47843137]
  [0.6392157  0.6117647  0.52156866]
  …
  [0.56078434 0.52156866 0.54509807]
  [0.56078434 0.5254902  0.5568628 ]
  [0.56078434 0.52156866 0.5647059 ]]]


[[[1.         1.         1.        ]
  [0.99215686 0.99215686 0.99215686]
  [0.99215686 0.99215686 0.99215686]
  …
  [0.99215686 0.99215686 0.99215686]
  [0.99215686 0.99215686 0.99215686]
  [0.99215686 0.99215686 0.99215686]]

 [[1.         1.         1.        ]
  [1.         1.         1.        ]
  [1.         1.         1.        ]
  …
  [1.         1.         1.        ]
  [1.         1.         1.        ]
  [1.         1.         1.        ]]

 [[1.         1.         1.        ]
  [0.99607843 0.99607843 0.99607843]
  [0.99607843 0.99607843 0.99607843]
```

```
  …
  [0.99607843 0.99607843 0.99607843]
  [0.99607843 0.99607843 0.99607843]
  [0.99607843 0.99607843 0.99607843]]

 …

 [[0.44313726 0.47058824 0.4392157 ]
  [0.43529412 0.4627451  0.43529412]
  [0.4117647  0.4392157  0.41568628]
  …
  [0.28235295 0.31764707 0.3137255 ]
  [0.28235295 0.3137255  0.30980393]
  [0.28235295 0.3137255  0.30980393]]

 [[0.43529412 0.4627451  0.43137255]
  [0.40784314 0.43529412 0.40784314]
  [0.3882353  0.41568628 0.38431373]
  …
  [0.26666668 0.29411766 0.28627452]
  [0.27450982 0.29803923 0.29411766]
  [0.30588236 0.32941177 0.32156864]]

 [[0.41568628 0.44313726 0.4117647 ]
  [0.3882353  0.41568628 0.38431373]
  [0.37254903 0.4        0.36862746]
  …
  [0.30588236 0.33333334 0.3254902 ]
  [0.30980393 0.33333334 0.3254902 ]
  [0.3137255  0.3372549  0.32941177]]]


…


[[[0.13725491 0.69803923 0.92156863]
  [0.15686275 0.6901961  0.9372549 ]
  [0.16470589 0.6901961  0.94509804]
  …
  [0.3882353  0.69411767 0.85882354]
  [0.30980393 0.5764706  0.77254903]
  [0.34901962 0.5803922  0.7411765 ]]

 [[0.22352941 0.7137255  0.91764706]
  [0.17254902 0.72156864 0.98039216]
  [0.19607843 0.7176471  0.9411765 ]
  …
  [0.6117647  0.7137255  0.78431374]
```

```
  [0.5529412  0.69411767 0.80784315]
  [0.45490196 0.58431375 0.6862745 ]]

 [[0.38431373 0.77254903 0.92941177]
  [0.2509804  0.7411765  0.9882353 ]
  [0.27058825 0.7529412  0.9607843 ]
  …
  [0.7372549  0.7647059  0.80784315]
  [0.46666667 0.5294118  0.5764706 ]
  [0.23921569 0.30980393 0.3529412 ]]

 …

 [[0.28627452 0.30980393 0.3019608 ]
  [0.20784314 0.24705882 0.26666668]
  [0.21176471 0.26666668 0.3137255 ]
  …
  [0.06666667 0.15686275 0.2509804 ]
  [0.08235294 0.14117648 0.2       ]
  [0.12941177 0.1882353  0.19215687]]

 [[0.23921569 0.26666668 0.29411766]
  [0.21568628 0.27450982 0.3372549 ]
  [0.22352941 0.30980393 0.40392157]
  …
  [0.09411765 0.1882353  0.28235295]
  [0.06666667 0.13725491 0.20784314]
  [0.02745098 0.09019608 0.1254902 ]]

 [[0.17254902 0.21960784 0.28627452]
  [0.18039216 0.25882354 0.34509805]
  [0.19215687 0.3019608  0.4117647 ]
  …
  [0.10588235 0.20392157 0.3019608 ]
  [0.08235294 0.16862746 0.25882354]
  [0.04705882 0.12156863 0.19607843]]]


[[[0.7411765  0.827451   0.9411765 ]
  [0.7294118  0.8156863  0.9254902 ]
  [0.7254902  0.8117647  0.92156863]
  …
  [0.6862745  0.7647059  0.8784314 ]
  [0.6745098  0.7607843  0.87058824]
  [0.6627451  0.7607843  0.8627451 ]]

 [[0.7607843  0.8235294  0.9372549 ]
  [0.7490196  0.8117647  0.9254902 ]
```

```
  [0.74509805 0.80784315 0.92156863]
  …
  [0.6784314  0.7529412  0.8627451 ]
  [0.67058825 0.7490196  0.85490197]
  [0.654902   0.74509805 0.84705883]]

 [[0.8156863  0.85882354 0.95686275]
  [0.8039216  0.84705883 0.9411765 ]
  [0.8        0.84313726 0.9372549 ]
  …
  [0.6862745  0.7490196  0.8509804 ]
  [0.6745098  0.74509805 0.84705883]
  [0.6627451  0.7490196  0.84313726]]

 …

 [[0.8117647  0.78039217 0.70980394]
  [0.79607844 0.7647059  0.6862745 ]
  [0.79607844 0.76862746 0.6784314 ]
  …
  [0.5294118  0.5176471  0.49803922]
  [0.63529414 0.61960787 0.5882353 ]
  [0.65882355 0.6392157  0.5921569 ]]

 [[0.7764706  0.74509805 0.6666667 ]
  [0.7411765  0.70980394 0.62352943]
  [0.7058824  0.6745098  0.5764706 ]
  …
  [0.69803923 0.67058825 0.627451  ]
  [0.6862745  0.6627451  0.6117647 ]
  [0.6862745  0.6627451  0.6039216 ]]

 [[0.7764706  0.7411765  0.6784314 ]
  [0.7411765  0.70980394 0.63529414]
  [0.69803923 0.6666667  0.58431375]
  …
  [0.7647059  0.72156864 0.6627451 ]
  [0.76862746 0.7411765  0.67058825]
  [0.7647059  0.74509805 0.67058825]]]


[[[0.8980392  0.8980392  0.9372549 ]
  [0.9254902  0.92941177 0.96862745]
  [0.91764706 0.9254902  0.96862745]
  …
  [0.8509804  0.85882354 0.9137255 ]
  [0.8666667  0.8745098  0.91764706]
  [0.87058824 0.8745098  0.9137255 ]]
```

```
[[0.87058824 0.8666667  0.8980392 ]
 [0.9372549  0.9372549  0.9764706 ]
 [0.9137255  0.91764706 0.9647059 ]
 …
 [0.8745098  0.8745098  0.9254902 ]
 [0.8901961  0.89411765 0.93333334]
 [0.8235294  0.827451   0.8627451 ]]

[[0.8352941  0.80784315 0.827451  ]
 [0.91764706 0.9098039  0.9372549 ]
 [0.90588236 0.9137255  0.95686275]
 …
 [0.8627451  0.8627451  0.9098039 ]
 [0.8627451  0.85882354 0.9098039 ]
 [0.7921569  0.79607844 0.84313726]]


…


[[0.5882353  0.56078434 0.5294118 ]
 [0.54901963 0.5294118  0.49803922]
 [0.5176471  0.49803922 0.47058824]
 …
 [0.8784314  0.87058824 0.85490197]
 [0.9019608  0.89411765 0.88235295]
 [0.94509804 0.94509804 0.93333334]]

[[0.5372549  0.5176471  0.49411765]
 [0.50980395 0.49803922 0.47058824]
 [0.49019608 0.4745098  0.4509804 ]
 …
 [0.70980394 0.7058824  0.69803923]
 [0.7921569  0.7882353  0.7764706 ]
 [0.83137256 0.827451   0.8117647 ]]

[[0.47843137 0.46666667 0.44705883]
 [0.4627451  0.45490196 0.43137255]
 [0.47058824 0.45490196 0.43529412]
 …
 [0.7019608  0.69411767 0.6784314 ]
 [0.6431373  0.6431373  0.63529414]
 [0.6392157  0.6392157  0.6313726 ]]]]
```

[36]:
```
# loading the CIFAR10 dataset and create the training and test arrays
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Lines 1 and 2 reshapes the inputs
```

```
        X_train = X_train.reshape((X_train.shape[0], 32, 32, 3)).
 ↪astype('float32')
        X_test = X_test.reshape((X_test.shape[0], 32, 32, 3)).astype('float32')

        # Lines 3 and 4
        # Normalization of the input values (image pixels) from 0 and 255 to 0.1
        X_train = X_train / 255
        X_test = X_test / 255

        # Lines 5 and 6
        # one-hot encoding of the target variables
        y_train = np_utils.to_categorical(y_train)
        y_test = np_utils.to_categorical(y_test)

        num_classes = y_test.shape[1]
```

[33]:
```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=5, batch_size=150, verbose=0)
```

```
       ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-33-ac3bcd717409> in <module>
   ----> 1 history = model.fit(X_train, y_train, validation_data=(X_test,␣
↪y_test), epochs=5, batch_size=150, verbose=0)


       NameError: name 'X_train' is not defined
```

[ ]:
```
plt.subplot(211)
plt.title('Cross Entropy Loss')
plt.plot(history.history['loss'], color='blue', label='train')
plt.plot(history.history['val_loss'], color='orange', label='test')
# plot accuracy
plt.subplot(212)
plt.title('Classification Accuracy')
plt.plot(history.history['accuracy'], color='blue', label='train')
plt.plot(history.history['val_accuracy'], color='orange', label='test')
# save plot to file
plt.savefig(f'{results_dir}\\1_plot.png')
plt.show()
plt.close()
```

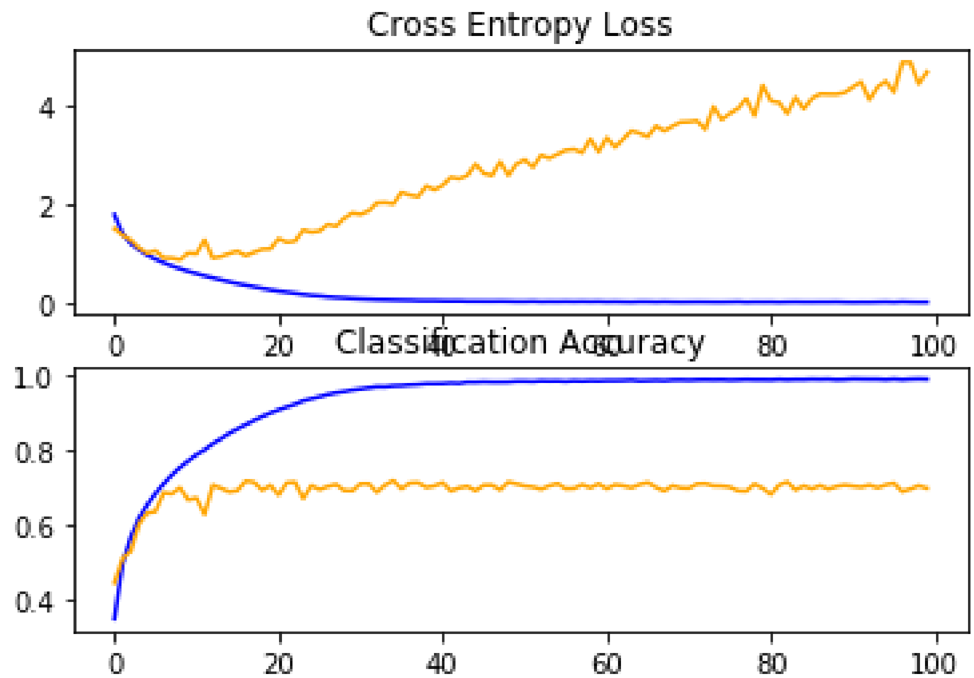
[ ]:
```
scores = model.evaluate(X_test, y_test, verbose=0)

print("CNN Accuracy: %.3f%%" % (scores[1]*100.0))
```

[41]:
```
def run_model():

        print('Load dataset')
        load_dataset()

        print('dataset loaded')
        print(f'Training set: {X_train.shape}')
        print(f'Test Set: {X_test.shape}')
        print(f'Number of categories : {num_classes}')

        print('Build model')
        model = cnn_model()
        print('Model is defined')
        print('Summary of the model.')
        model.summary()

        print('Compile Model')
        model.compile(loss='categorical_crossentropy', optimizer='rmsprop',␣
 ↪metrics=['accuracy'])
        print('Model compiled')

        print('Model fitting Considering 5 epochs and a batch size of 150')
        history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=100, batch_size=150, verbose=0)

        print('Saving the model')
        model.save(f'{results_dir}\\assignment_6.2a_cifar10.h5')

        print('Evaluating the model on the test data')
        scores = model.evaluate(X_test, y_test, verbose=0)

        print("CNN Accuracy: %.3f%%" % (scores[1]*100.0))

        print('Output summary')
        # summary_plot(history)
        summarize_diagnostics(history)
```

[42]:
```
run_model()
```

```
Load dataset
dataset loaded
Training set: (50000, 32, 32, 3)
Test Set: (10000, 32, 32, 3)
Number of categories : 10
```

```
Build model
Model is defined
Summary of the model.
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_34 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
=================================================================
Total params: 160,202
Trainable params: 160,202
Non-trainable params: 0
_________________________________________________________________
Compile Model
Model compiled
Model fitting Considering 5 epochs and a batch size of 150
Saving the model
Evaluating the model on the test data
CNN Accuracy: 69.660%
Output summary
```



Cross Entropy Loss
0
2
4
0
20
40
60
80
100
Classification Accuracy
0.4
0.6
0.8
1.0
0
20
40
60
80
100