Spring 2024: CS5720

Neural Networks & Deep Learning - ICP-8

Image classification with CNN

NAME: SOWMYA DONETI ID: 700754085

GITHUB LINK: https://github.com/sowmyadoneti/NN Assignment ICP8

CODE &SCREENSHOTS FOR RESULTS:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.datasets import mnist

from tensorflow.keras.optimizers import RMSprop
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNormalization

**matplotlib inline
```

Extract data and train and test dataset

```
#cifar100 = tf.keras.datasets.cifar100
(X_train,Y_train) , (X_test,Y_test) = cifar10.load_data()

M classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

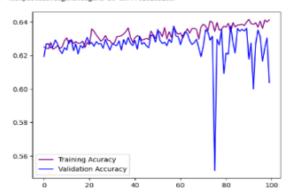
Let's look into the dataset images

```
plt.figure(figsize = (16,16))
for i in range(100):
   plt.subplot(10,10,1+i)
   plt.axis('off')
   plt.imshow(X_train[i], cmap = 'gray')
```



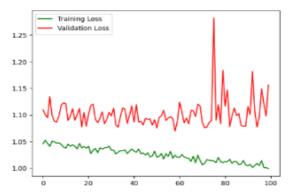
```
M from sklearn.model_selection import train_test_split
  x_train, x_val, y_train, y_val = train_test_split(X_train,Y_train,test_size=0.2)
▶ from keras.utils.np_utils import to_categorical
  y_train = to_categorical(y_train, num_classes = 10)
  y_val = to_categorical(y_val, num_classes = 10)
M print(x_train.shape)
  print(y_train.shape)
  print(x_val.shape)
  print(y_val.shape)
  print(X_test.shape)
  print(Y_test.shape)
  (40000, 32, 32, 3)
  (40000, 10)
   (10000, 32, 32, 3)
   (10000, 10)
   (10000, 32, 32, 3)
  (10000, 1)
M train_datagen = ImageDataGenerator(
      preprocessing_function = tf.keras.applications.vgg19.preprocess_input,
      rotation_range=10,
      zoom_range = 0.1,
      width_shift_range = 0.1,
      height_shift_range = 0.1,
      shear_range = 0.1,
      horizontal_flip = True
  train_datagen.fit(x_train)
  val_datagen = ImageDataGenerator(preprocessing_function = tf.keras.applications.vgg19.preprocess_input)
  val_datagen.fit(x_val)
```

```
Out[76]: <matplotlib.legend.Legend at 0x7f75101e8160>
```



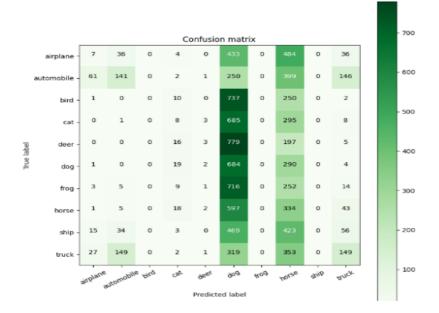
```
In [77]: M
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.figure()
plt.plot(loss,color = 'green',label = 'Training Loss')
plt.plot(val_loss,color = 'red',label = 'Validation Loss')
plt.legend()
```

Out[77]: cmatplotlib.legend.Legend at 0x7f75101e8d30>



```
M import itertools
   def plot_confusion_matrix(cm, classes,
                                 normalize=False,
                                 title='Confusion matrix',
                                 cmap=plt.cm.Greens):
       This function prints and plots the confusion matrix.
       Normalization can be applied by setting 'normalize=True'.
       plt.imshow(cm, interpolation='nearest', cmap=cmap)
       plt.title(title)
       plt.colorbar()
        tick_marks = np.arange(len(classes))
       plt.xticks(tick_marks, classes, rotation=30)
       plt.yticks(tick_marks, classes)
       if normalize:
            cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print("Normalized confusion matrix")
        else:
            print('Confusion matrix, without normalization')
       #print(cm)
       thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, cm[i, j],
    horizontalalignment="center",
    color="white" if cm[i, j] > thresh else "black")
       plt.tight_layout()
       plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Confusion matrix, without normalization



check data plt.imshow(x_train[1]) print(x_train[1].shape)

(32, 32, 3)

