20/04/2021

CS6005-DEEP LEARNING MINI PROJECT-CNN DOCUMENT

Image Classification using CNN on CIFAR-10 Dataset

-Syed Adnan Moin.S

2018103074

Problem Statement:

Neural networks have achieved appreciable performance at image classification and they are widely accepted. It is a tiresome process that results in poor generalization of the test data.

Moreover, when the images are taken in varying lighting conditions and at different orientation or angles, and since these are coloured images, there are many variations in the colour itself of similar objects (for example, the colour of hair,animals,cars,objects etc). The artificial neural network is unable to classify or predict with a higher accuracy. As it involves a lot of weights to be taken into account.

The simple CNN architecture that we use in the basic datasets such as MNIST, we will get a low validation accuracy of around 60% which is never desirable. Hence, to improve the prediction metrics desirable for higher datasets such as CIFAR-10, I'm presenting a convolutional neural network (CNN) approach which works to achieve improved performances. Learnable filters and pooling layers are used to extract underlying image features. Dropout, regularization along with variation in convolution strategies are applied to reduce overfitting while ensuring increased accuracies in validation and testing. Better test accuracy with reduced overfitting is achieved with a deeper network.

CIFAR-10 is a dataset that consists of several images divided into the following 10 classes:

- Airplanes
- Cars
- Birds
- Cats
- Deers
- Dogs
- Frogs
- Horses
- Ships
- Trucks

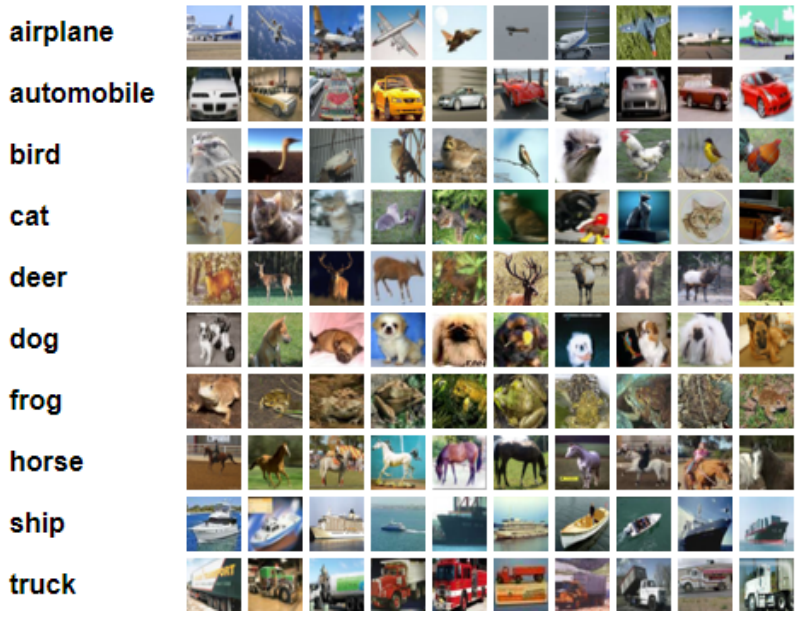
- Dataset consists of 60,000 32x32 color images, i.e. 6000 images per class.
- So the main objective of this project is to design and implement(Train and Test) a properly functional CNN model which is able to predict the labels of these images present in the CIFAR-10 Dataset correctly with higher accuracy and precision.

CIFAR-10 Dataset Description:

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

The classes in the dataset with 10 random images from each class:



The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks. url- https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

Module Description:

- **Loading the dataset:**
  - The dataset used is the CIFAR-10 dataset which is available in the dataset module of tensorflow.Keras. The dataset is loaded using the datasets.cifar10.load_data() function to the train and test parts being the x_train, y_train and x_test, y_test.
  - Training images i.e from x_train are plotted using the pyplot package in the matplotlib library.These images are plotted to describe the layout of the figure. The rows and columns of the frame which is for plotting is given in the form of arguments, and the third argument is for providing the index of the current plot.
- **Data Preprocessing:**
  - **Reshaping:**
    - The row vector for an image has the exact same number of elements if you calculate 32*32*3 == 3072. In order to reshape the row vector into (width x height x num_channel) form, there are two steps required. The first step is to use reshape function, and the second step is to use the transpose function in numpy.
    - reshape transforms an array to a new shape without changing its data. Here, the phrase without changing its

data is an important part since we don't want to affect the data.

- **Normalization:**
  - normalize function takes data, x, and returns it as a normalized Numpy array. x can be anything, and it can be an N-dimensional array.
  - It will be a 3-D array for an image. Min-Max Normalization (y = (x-min) / (max-min)) technique is used.
  - Here Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each value in the channel can range from 0 to 255. Hence to normalize in 0-->1 range, we need to divide it by 255.

- **CNN Model:**
  - The image classification using CNN model is able to achieve an accuracy of 82%.The main objective is to perform image classification using CNN and comparing its accuracy in predicting the test image with the accuracy given by the deep neural network model.
  - Here multiple layers are much better at generalizing because they learn all the intermediate features between the raw_input and the high-level classification.

- The 4 building blocks of Convolutional Neural Network:

  Convolution layer=>Pooling layer=>Flattening layer=>Output layer

- **Mode**l:
  - Here, I have built a 7 layer convolutional neural network followed by a flatten layer. In these 7 layers, 2 layers have the number of filters as 32 and the kernel size as (3x3), 2 layers have the number of filters as 64 and the kernel size as (3x3) and the activation function as relu with padding as same i.e. adds padding to the input image. After this, the max pooling function is done with a pool_size of (2x2). The output layer is a dense layer of 10 nodes (as there are 10 classes) with softmax activation.
  - Convolution
    - The **convolutional layer** is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume.
  - Pooling
    - its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. It operates on each feature map independently.
  - Adding a second convolutional layer
    - Flattening
      - Flattening is converting the data into a 1-dimensional array for inputting it to the next

layer. We flatten the output of the convolutional layers to create a single long feature vector

- Full Connection
    - They form the last few layers in the network. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.

- Compiling the CNN
    - Before training the model we need to compile it and define the loss function, optimizers, and metrics for prediction. We compile the model using compile() method
    - Optimizer, loss, and metrics are the necessary arguments.
    - We use ‘adam’ optimizer here. Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models
    - Binary cross entropy is a loss function that is used in binary classification tasks. These are tasks that answer a question with only two choices
    - Accuracy is the performance metric we use to evaluate the model

- Training CNN and evaluation
    - We use fit() for training the model with the given inputs (and corresponding training labels).
    - In terms of artificial neural networks, an epoch refers to one cycle through the full training dataset

- **Prediction:**

  We can predict quantities with the finalized regression model by calling the predict() function on the finalized model. The predict() function takes an array of one or more data instances. Prediction and display using consoleA result dictionary is made to map the output classes and make predictions from the model. The image which has to be classified is opened by the Image.open(path) function in the PIL library. The image is resized to (32x32) as the images in the dataset are of that dimension and the image has to be an RGB image. The image is converted to an array and then the label or class of the image is predicted and displayed.

CNN Model Summary:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
```

activation_4 (Activation)	(None, 8, 8, 128)	0
batch_normalization_4 (Batch	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
activation_5 (Activation)	(None, 8, 8, 128)	0
batch_normalization_5 (Batch	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 10)	20490

Total params: 309,290
Trainable params: 308,394
Non-trainable params: 896

Code Snippet:

Module 1(Loading dataset):

#Importing required libraries and retrieving cifar-10 dataset from tensorflow.Keras

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

#Splitting the dataset into two for Training and Testing.

```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

Training set:

```
Out[2]: (50000, 32, 32, 3)
```

Testing set

```
In [3]: X_test.shape
Out[3]: (10000, 32, 32, 3)
```

Module 2(Data Preprocessing):

#Transforming(reshape) 2d array into 1d for faster processing

```
In [5]: y_train[:5]
Out[5]: array([[6],
               [9],
               [9],
               [4],
               [1]], dtype=uint8)

In [6]: y_train = y_train.reshape(-1,)
        y_train[:5]
Out[6]: array([6, 9, 9, 4, 1], dtype=uint8)

In [7]: y_test = y_test.reshape(-1,)
```

#Assigning labels

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

Module 3(Normalization):

#Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each value in the channel can range from 0 to 255. Hence to normalize in 0-->1 range, we need to divide it by 255

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

#Testing accuracy with artificial neural network:

In [13]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 139s 87ms/step - loss: 1.9311 - accuracy: 0.3080
Epoch 2/5
1563/1563 [==============================] - 200s 128ms/step - loss: 1.6415 - accuracy: 0.4222
Epoch 3/5
1563/1563 [==============================] - 212s 136ms/step - loss: 1.5519 - accuracy: 0.4500
Epoch 4/5
1563/1563 [==============================] - 210s 134ms/step - loss: 1.4900 - accuracy: 0.4747
Epoch 5/5
1563/1563 [==============================] - 222s 142ms/step - loss: 1.4360 - accuracy: 0.4966
```

Out[13]:
```
<tensorflow.python.keras.callbacks.History at 0x1daa1070ac0>
```

#printing performance metrics for artificial neural network:

In [14]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.51      0.52      1000
           1       0.59      0.63      0.61      1000
           2       0.36      0.31      0.33      1000
           3       0.36      0.31      0.33      1000
           4       0.40      0.40      0.40      1000
           5       0.50      0.23      0.32      1000
           6       0.33      0.83      0.47      1000
           7       0.78      0.27      0.40      1000
           8       0.48      0.75      0.58      1000
           9       0.74      0.24      0.37      1000

    accuracy                           0.45     10000
   macro avg       0.51      0.45      0.43     10000
weighted avg       0.51      0.45      0.43     10000
```

The artificial neural network model gives a poor accuracy of 45%.

Module 4(CNN model):

#Building the model:

In [15]:
```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

In [16]:
```
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

#Training the model for 10 epochs:

```
cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 140s 88ms/step - loss: 1.6629 - accuracy: 0.3950
Epoch 2/10
1563/1563 [==============================] - 138s 88ms/step - loss: 1.1018 - accuracy: 0.6106
Epoch 3/10
1563/1563 [==============================] - 139s 89ms/step - loss: 0.9472 - accuracy: 0.6680
Epoch 4/10
1563/1563 [==============================] - 137s 88ms/step - loss: 0.8565 - accuracy: 0.7023
Epoch 5/10
1563/1563 [==============================] - 137s 87ms/step - loss: 0.7770 - accuracy: 0.7315
Epoch 6/10
1563/1563 [==============================] - 139s 89ms/step - loss: 0.6963 - accuracy: 0.7583
Epoch 7/10
1563/1563 [==============================] - 138s 88ms/step - loss: 0.6464 - accuracy: 0.7735
Epoch 8/10
1563/1563 [==============================] - 141s 90ms/step - loss: 0.5903 - accuracy: 0.7959
Epoch 9/10
1563/1563 [==============================] - 144s 92ms/step - loss: 0.5473 - accuracy: 0.8084
Epoch 10/10
1563/1563 [==============================] - 141s 90ms/step - loss: 0.5009 - accuracy: 0.8259

<tensorflow.python.keras.callbacks.History at 0x1da97bacee0>
```

#printing testing accuracy:

```
In [18]: cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 8s 24ms/step - loss: 0.9749 - accuracy: 0.7036
```

```
Out[18]: [0.97488933801651, 0.7035999894142151]
```

Module 5(Prediction):

#performing prediction using cnn.predict and using np.argmax to pick the max value from the array:

```
In [19]: y_pred = cnn.predict(X_test)
         y_pred[:5]
```

```
Out[19]: array([[1.35963538e-03, 1.49373533e-04, 1.29289215e-03, 9.68111336e-01,
                 1.05673389e-03, 7.70801818e-03, 1.51946861e-02, 3.83928527e-05,
                 5.04459115e-03, 4.42828459e-05],
                [1.77865295e-05, 2.30023070e-04, 3.54659613e-09, 1.21826673e-08,
                 7.43588108e-11, 2.33842078e-11, 3.43071220e-13, 9.53027479e-13,
                 9.99749959e-01, 2.25194185e-06],
                [1.63594171e-01, 3.73529866e-02, 1.42420795e-05, 1.14025497e-04,
                 1.19233982e-05, 2.19643402e-06, 4.03453168e-06, 5.41781992e-05,
                 7.98576593e-01, 2.75694969e-04],
                [9.95112956e-01, 3.12948367e-04, 4.03344282e-04, 2.10965445e-05,
                 3.01279251e-05, 4.28109018e-07, 1.02025890e-08, 6.72925466e-08,
                 4.09234501e-03, 2.66188054e-05],
                [1.47009018e-07, 1.16872582e-06, 2.09870399e-03, 7.76511338e-03,
                 4.01659191e-01, 1.29596528e-03, 5.87177694e-01, 1.79366708e-07,
                 1.78963558e-06, 2.03864374e-08]], dtype=float32)
```

```
In [20]: y_classes = [np.argmax(element) for element in y_pred]
         y_classes[:5]
```

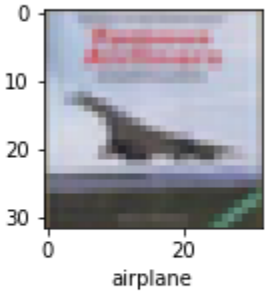
```
Out[20]: [3, 8, 8, 0, 6]
```

```
In [21]: y_test[:5]
```

```
Out[21]: array([3, 8, 8, 0, 6], dtype=uint8)
```

#predicting for a particular plot:

```
In [22]: plot_sample(X_test, y_test,3)
```




```
In [23]: classes[y_classes[3]]
Out[23]: 'airplane'
```

Hence the CNN model is able to predict with a higher accuracy 82% than the artificial neural network.

Results

Sample Images

#Plotting images to view dataset image:

```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

Image 1:

```
plot_sample(X_train, y_train, 0)
```

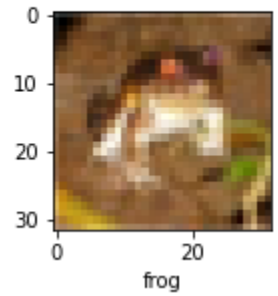


Image 2:

```
plot_sample(X_train, y_train, 1)
```

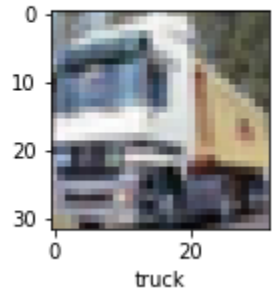
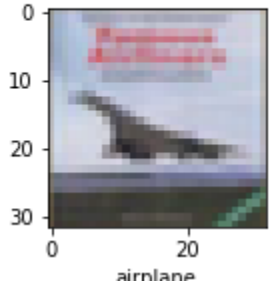


Image to be predicted:

```
: plot_sample(X_test, y_test,3)
```



Prediction

```
: classes[y_classes[3]]
: 'airplane'
```

Conclusion

The images in the 10 classes namely, airplanes, automobile, birds, cats, deer, dogs, frogs, horses, ships, and trucks of the CIFAR-10 dataset are classified into their respective classes with an **accuracy of 82.59%**. This accuracy can be improved by increasing the number of epochs. For instance, we can obtain an accuracy of 95% and above by increasing the number of epochs to 50 and more. But the increase should be as such that the model is not overfitted.

References

1. Image Classification– CIFAR-10 Dataset by Tushar Jajodia and Pankaj Garg published in 2020.
2. Improvement in Convolutional Neural Network for CIFAR-10 Dataset Image Classification by Suyesh Pandit and Sushil Kumar published in 2020.