

Image classification on Cifar-10 dataset

The cifar-10 dataset contains 32x32 sized 60,000 labelled images of 10 different classes. The goal is to build a model that would accurately classify the images into the following classes.

0: Aeroplane, 1: Automobile, 2: Bird, 3: Cat, 4: Deer, 5: Dog, 6: Frog, 7: Horse, 8: Ship, 9: Truck

I have used Convolutional Neural Network (CNN) for this image classification problem.

I have used Keras with TensorFlow backend.

Initially I was using Spyder as IDE on my local computer, but the performance was really low. Then, I switched to Google Colab, and used the GPU provided by them to execute my program.

Implementation

The dataset is imported and categorized into training data (train_X, train_y) and testing data (test_X, test_y). The labels (train_y, test_y) contained integer values from 0-9. The labels column is one hot encoded using **keras.util.to_categorical ()** function. This transformed the column to 10-dimension binary vector with all the categories being the columns of the vector and having 1 for the specific class.

The feature vectors i.e. the pixel values (train_X, test_X) are having values from 0 – 255 with 0 being the lightest shade and 255 being the darkest. Each image is of size 32x32 with 3 -channels (Red, Green, Blue). The pixel values are normalized to the range of 0 – 1.

Here I have used Convolutional Neural Network to build the model. CNN has the following steps:

>> Convolution : **Conv2D** function adds a 2-d convolution layer. Here a feature detector or filter is used over the input image to generate a feature map. The feature detector is of size 3x3. And 32 or 64 signifies the number of feature detectors. The output will retain some necessary features and get rid of unwanted information.

>> MaxPooling : **MaxPooling2D** is used for pooling. This reduces dimensionality of the feature maps and retain the most important information. Here 2x2 matrix is used on the feature map to find the maximum value from those particular pixels. This is done across all the pixels to generate a Pooled feature map which preserved the important feature and ignore spatial differences.

>> Flattening : **Flatten** is used to convert the feature map to a 1-D vector. The elements are taken row by row and put it into one column. This vector is given as input to the Fully connected Neural Network for further processing.

>> Full Connection Layer : **Dense** function is used to initialize a fully connected network. In CNN, the hidden layers must be fully connected. The final layer contains 10 neurons to signify 10 output classes.

>> **Dropout**: **Dropout** is a regularization technique that is used to prevent overfitting. This layer randomly sets some data points to 0. I've used Dropout(0.3), means this layer will randomly assign 0 to 30% of the data.

An **Activation function** is used to introduce non-linearity in the data. This is a function which maps the input data to the output of that neuron. This output is given as input to another neuron, and so on until the desired output is received. The activation function helps the network to learn the complex pattern in the data.

Here **ReLU (Rectified Linear Unit)** is mainly used. ReLU function is defined as $y = \max(0, x)$. This function returns 0 if it receives any negative value and for any positive value x , it returns the value. **Softmax** function is used in the output layer. This function adds the value of output to a form that the sum of the output becomes 1. It takes input vector of K-dimensions and normalize it into a probability distribution consisting of K probabilities proportional to the exponents of the input numbers.

For learning the parameters **Stochastic Gradient Descent(SGD)** is used. This is a classical optimization algorithm to increase the model accuracy by minimizing the loss using standard weight update formula of: $W_{t+1} = W_t + n * \text{gradient}$ (n : learning rate).

In SGD, the parameter update for an epoch is done by using just one or a small subset (mini-batch) data point from the batch of training data, unlike vanilla gradient descent, where update is done by calculating all the data points of the training data for one epoch.

The learning rate chosen is 0.01 and a momentum of 0.9, which is an ideal situation. A decay is used ($\text{decay} = \text{learning rate}/\text{epoch}$) to keep on decreasing the learning rate so that smaller steps are taken as we move towards finding the optimal weights so that the loss function becomes minimum (approaching 0).

The loss function used is **Categorical Cross Entropy**. This is used for multi-class classification problems where we have to choose 1 class out of k classes.

Through many trial and error experiments, by changing different parameters of the layers of CNN, trying different combinations of the layers of CNN, like adding a convolution layer having different number of filter, adding Dropout layers in between, trying out different learning rates and different optimization algorithms, trying various activation functions, and many more such manipulation, I've finally reached the **Accuracy of 81.690%**.

The CNN model predicted the correct class for most of the image during testing phase. The images used for testing are downloaded from Internet.

The screenshot of code and output is given below:

```

# -*- coding: utf-8 -*-
# importing the necessary libraries
import sys
import tensorflow as tf
from matplotlib import pyplot
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.optimizers import SGD
import warnings
warnings.filterwarnings("ignore")

```

```

[80] #load the cifar dataset
(train_X, train_y),(test_X,test_y) = cifar10.load_data()

#one hot encoding the labels, transforming integers into 10 dimension binary vector, having 1 for that specific class.
train_y = to_categorical(train_y)
test_y = to_categorical(test_y)

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
#normalize to the range of 0-1
train_X = train_X / 255.0
test_X = test_X / 255.0

```

```

#define CNN model
model = Sequential()
#convolution layer
#input layer: 32x32 image size having 3 channels
model.add(Conv2D(32, (3, 3), input_shape=(32,32,3), activation='relu', padding='same'))
#convolution layer
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
#dropout regularization layer, assigning 0 to random 30% of the data. done to prevent overfitting
model.add(Dropout(0.3))
#max pooling layer with 2x2 sized filter
model.add(MaxPooling2D(pool_size=(2, 2)))
#convolution layer
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
#convolution layer
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
#dropout layer
model.add(Dropout(0.3))
#max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))
#latching layer. The feature map is converted to a 1-D matrix
model.add(Flatten())
#dropout layer
model.add(Dropout(0.3))
#full connection layer with output of 512 neurons
model.add(Dense(512, activation='relu'))
#dropout layer
model.add(Dropout(0.3))
#full connection layer with output of 128 neurons
model.add(Dense(128, activation='relu'))
#dropout layer
model.add(Dropout(0.3))
#full connection layer with output of 10 neurons, for 10 output classes
model.add(Dense(10, activation='softmax'))

```

```

#learning rate
lr_rate = 0.01
#decay of learning rate over time. decay = learning_rate / no_of_epochs
decay = lr_rate/100
#stochastic gradient descent optimization algorithm for learning the parameters
sgd = SGD(lr=lr_rate, momentum=0.9, decay = decay, nesterov=False)
#compile model using categorical cross-entropy loss function
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

```

```

[83] # Training the CNN model with cifar-10 data
history = model.fit(train_X, train_y, epochs=100, batch_size=32, validation_data=(test_X, test_y))
_, acc = model.evaluate(test_X, test_y)
print('Accuracy = %.3f' % (acc*100.0))
# model is saved
model.save('project_model.h5')

# Analyze the loss and accuracy for training and validation data using history function
pyplot.subplot(211)
pyplot.title('cross entropy loss')
pyplot.plot(history.history['loss'], color = 'blue', label = 'training data')
pyplot.plot(history.history['val_loss'], color = 'green', label = 'test data')

pyplot.subplot(212)
pyplot.title('classification accuracy')
pyplot.plot(history.history['accuracy'], color = 'blue', label = 'training data')
pyplot.plot(history.history['val_accuracy'], color = 'green', label = 'test data')

```

```

from keras.models import load_model
import numpy as np
from keras.preprocessing import image
from google.colab import drive

drive.mount('/content/drive')
#load the previously saved model
model = load_model('project_model.h5')

```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```

#image is loaded and shrunk to 32x32 size, i.e same size as the training image data
test_image1 = image.load_img('/content/drive/My Drive/image_classification_cifar-10/photo9.jpg', target_size = (32,32))
#image type is PIL, which stands for Python Image Library.

#image is converted to numpy array as the model is trained on numpy array object and it can recognize numbers only.
test_image = image.img_to_array(test_image1)

#image is reshaped according to input shape of 4D tensor (batch, rows, columns,channels )
test_image = test_image.reshape(1,32,32,3)

#normalizing the img like the training data
test_image = test_image.astype('float32')
test_image = test_image / 255.0

#predicting the class in which the given image belongs
result = model.predict_classes(test_image)

#display the image
import matplotlib.pyplot as plt
plt.imshow(test_image1)
print('Class no: ',result)

```

```
#display name of the class according to class number
if result==0:
    print('Aeroplane')
elif result==1:
    print('Automobile')
elif result==2:
    print('Bird')
elif result==3:
    print('Cat')
elif result==4:
    print('Deer')
elif result==5:
    print('Dog')
elif result==6:
    print('Frog')
elif result==7:
    print('Horse')
elif result==8:
    print('Ship')
elif result==9:
    print('Truck')
```

Outputs

image-classification-cifar-10.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
elif result==3:
    print('Cat')
elif result==4:
    print('Deer')
elif result==5:
    print('Dog')
elif result==6:
    print('Frog')
elif result==7:
    print('Horse')
elif result==8:
    print('Ship')
elif result==9:
    print('Truck')
```

Class no: [1]
Automobile




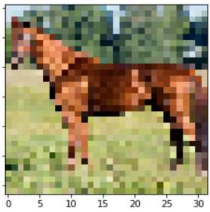
image-classification-cifar-10.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
elif result==3:
    print('Cat')
elif result==4:
    print('Deer')
elif result==5:
    print('Dog')
elif result==6:
    print('Frog')
elif result==7:
    print('Horse')
elif result==8:
    print('Ship')
elif result==9:
    print('Truck')
```

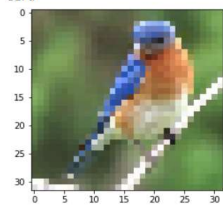
Class no: [7]
Horse



+ Code + Text

```
elif result==3:
    print('Cat')
elif result==4:
    print('Deer')
elif result==5:
    print('Dog')
elif result==6:
    print('Frog')
elif result==7:
    print('Horse')
elif result==8:
    print('Ship')
elif result==9:
    print('Truck')
```

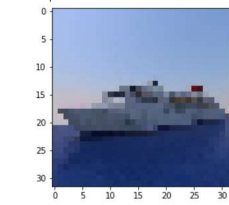
Class no: [2]



+ Code + Text

```
elif result==3:
    print('Cat')
elif result==4:
    print('Deer')
elif result==5:
    print('Dog')
elif result==6:
    print('Frog')
elif result==7:
    print('Horse')
elif result==8:
    print('Ship')
elif result==9:
    print('Truck')
```

Class no: [8]



+ Code + Text

```
print('dog')
elif result==6:
    print('frog')
elif result==7:
    print('Horse')
elif result==8:
    print('Ship')
elif result==9:
    print('Truck')
```

Class no: [6]

