IMAGE RECOGNITION

CE6051 – Machine Learning in Civil Engineering

Team member	Roll number
M. Prabhat	CE16B041
Sanket Thakre	MM18D702
P. V. Nitesh Kumar	CE16B006
M. Srikanth	CE16B040
T. Hari Krishna	CE16B014
P. Kranthi Kumar	CE16B007
Ch. Vishwa Vignan	CE16B026

Problem Statement:

Object detection and image classification is one of the key theories of computer vision technology. With increase of applications in areas like robotics, medical etc., the method is evolving with better accuracies and can be incorporated in real time systems. There are different computer vision techniques like sliding window, deformable part models etc. but the accuracy is far too low. Combining with deep learning techniques one can reap the benefits of better predictions and classifications. Deep learning uses many layers for determining non-linear relationships which helps us to classify data more accurately.

Introduction:

Machine learning involves writing a set of algorithms which mimic human learning to perform a specific task. Deep learning is a part of machine learning which has some structure or architecture so as to solve some complicated non-linear problems. Convolutional neural networks are deep learning algorithms which basically have a multilayer perceptron structure. The process of convolution ie. Applying an operator or a mask over a specified window which is continuously moving. Different from conventional neural networks, CNNs automatically extracts features from raw data which essentially reduces the expertise of domain knowledge. These convolutions take advantage by establishing local connectivity patterns which allow then to have few weights (shared parameters), thus extracting relevant information at low computational cost. The CNN architecture consists mainly of convolutional layer, pooling layer and fully connected layer.

Image processing involves image identification, classification and further analysis. It one of the upcoming fields in many engineering disciplines. Image data which is nothing but a matrix of 2D or 3D (if channels are included), is processed so as to extract features which can be later quantified on the basis of presence and absence of that feature in some other image. Generally we use many filters for transforming the matrix into a feature map so as to detect some patterns. There are various filters available like prewit operator, North-west direction mask etc.

Dataset: We use CIFAR10 dataset. It contains 60000 images of 10 different classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks), containing 6000 images each.

Step 1: Importing libraries and packages

Library	Utility
numpy	Contains functions for various mathematical operations
keras	Contains the functions of various machine learning algorithms like convolutions, stochastic gradient descent, etc
keras.utils	Used to import np_utils which is used to convert categorical data to one-hot encodings
keras.datasets	Used to load CIFAR10 dataset
from matplotlib import pyplot	Used for visualising the data
from PIL import image	Used to do various functions with images like loading them, printing them

Libraries used: Numpy (np.utils package), Keras (using TensorFlow in the backend), Matplotlib

Step 2: Loading the data

We can load the dataset directly using **keras.datasets**.

```
# load the data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

```
# Lets determine the dataset characteristics
print('Training Images: {}'.format(X_train.shape))
print('Testing Images: {}'.format(X_test.shape))
```

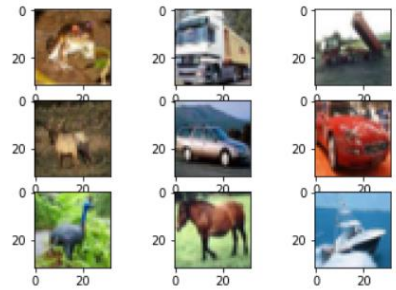
```
Training Images: (50000L, 3L, 32L, 32L)
Testing Images: (10000L, 3L, 32L, 32L)
```

```
# Now for a single image
print(X_train[0].shape)
```

```
(3L, 32L, 32L)
```

```
# create a grid of 3x3 images
for i in range(0,9):
    plt.subplot(330 + 1 + i)
    img = X_train[i].transpose([1,2,0])
    plt.imshow(img)

# show the plot
plt.show()
```



Step 3: Pre-processing the data

We split the dataset into training and test data and normalise the data by doing $\frac{x-Min}{Max-Min}$

We label the classes from 0 to 9. Now, we need to convert these into one-hot encodings, else the algorithm will treat a class with label '6' as six times more likely than a class with label '1'. To do this, we use **to_categorical** function in **np.utils**

```
# Preprocessing the dataset

# fix random seed for reproducibility
seed = 6
np.random.seed(seed)

# load the data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# normalize the inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

X_train /= 255.0
X_test /= 255.0
```

```
# class labels shape
print(y_train.shape)
print(y_train[0])
```

```
(50000L, 1L)
[6]
```

```
# [6] = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0] one-hot vector

# hot encode outputs
Y_train = np_utils.to_categorical(y_train)
Y_test = np_utils.to_categorical(y_test)
num_class = Y_test.shape[1]
print(num_class)

print(Y_train.shape)
print(Y_train[0])
```

```
10
(50000L, 10L)
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

Step 4: Building the model

Keras provides pre-defined functions like convolution, global pooling, etc and optimizers like Stochastic Gradient Descent. We will utilise those functions to build our model. Now we build our model according to the below architecture:

Input: $32 \times 32\ RGB\ images \rightarrow 32\ \times 32 \times 3$

Layers are as follows:

- $3 \times 3\ convolution, 96\ ReLu$
- $3 \times 3\ convolution, 96\ ReLu$
- $3 \times 3\ max - pooling,\ stride = 2$
- $3 \times 3\ convolution, 192\ ReLu$
- $3 \times 3\ convolution, 192\ ReLu$
- $3 \times 3\ max - pooling,\ stride = 2$
- $3 \times 3\ convolution, 192\ ReLu$
- $1 \times 1\ convolution, 192\ ReLu$
- $1 \times 1\ convolution, 10\ ReLu$
- $Global\ averaging\ over\ 6 \times 6\ spatial\ dimensions$
- $10\ or\ 100 - way\ softmax$

We introduce dropout in between the layers because it helps the algorithm to generalise and prevent over-fitting. We define initially define the weights randomly, which will be learned by the algorithm as it gets trained. We also define the stochastic gradient descent optimizer. Now, we set the number of epochs to train the data.

Step 5: Training the data

We can vary the optimizer inputs, like learning rate, weight decay and the number of epochs and check which combination provides the best results. After going through a lot of research, we finalized the following details:

	Parameter	Value
Optimizer	Learning rate	0.01
	Weight Decay	10^{-6}
	Momentum	0.9
Training	Epochs	350
	Batch Size	32

Step 6: Testing the data

After training the data, we evaluate the algorithm with the test data and determine the accuracy of the model. We prepare a dictionary of classes and labels to improve the ease of reading the data. We can give a sample images for the algorithm to predict and identify the accuracy of the predictions.

Results:

The research paper claims to have achieved an accuracy of 90.92% without using any other the techniques. However, if we use aggressive data augmentation, the accuracy increases to 92.75%. Since, we haven't completely finished the coding, we have not kept our achieved accuracy.

Use cases in Civil Engineering:

- Smart Traffic Light Control System: We can use the live feed from cameras installed near traffic signals as input and change the signal timings dynamically thereby effectively reducing the travel time of any vehicle.
- Driver Sleepiness Detection: Sleepiness during driving is one of the major reasons for accidents. If we can monitor the movement of his eyes and his head, then we can use deep learning techniques to alert the driver whenever possible.
- Infrastructure Health Assessment: With careful visual inspection of cracks, joints, etc using image recognition techniques, we can identify structures that are in a dangerous condition and warn the concerned officials about the same.