EE21B055 DSP Mini project Report
EE21B057

Image categorization using CNN's

Introduction :

In this project, we will apply deep convo-lutional neural networks to classify images. We are using a famous dataset known as "Cifar-ten". The dataset consists of several images divided into 10 categories such as planes, dogs, frogs, horses and trucks etc.

Understandings from this project :

* The theory and intution behind deep convolutional neural networks.
* Basics of digital image processing, performing 2D convolutions, max pooling and average pooling.
* Building a CNN model to perform image classification using keras.
* Optimize network weights using optimizer (Adam).
* Evaluating the model and presenting the results using confusion matrices.
* Save and retrieving trained network weights.

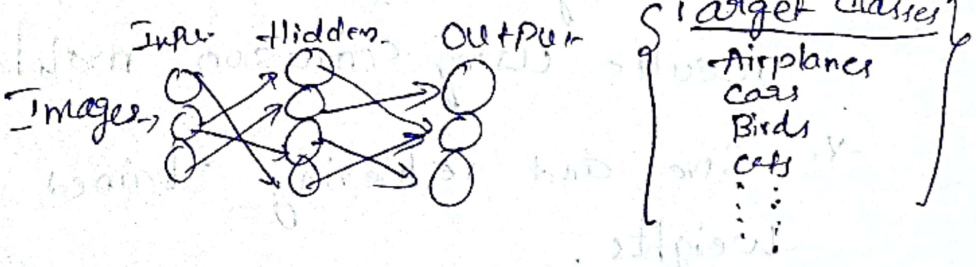
PROJECT OVERVIEW:

* CIFER-10 is dataset that consists of several images divided into the following 10 classes:
  - Airplanes, Cars, Birds, Cats, Deer, Dogs, Frogs, Horses, Ships, Trucks.
* CIFAR-10 is widely used for machine learning and computer vision applications.
* The dataset consists of 60,000 images, each image is 32 x 32 pixels, (6000 images of each class).

In this we take all the data provided and then we divide it into: 80% for training and 20% for testing.

We adjust the weights here within our classifier model, and then once the network is trained we bring the testing data. Finally, will give the output of classified images.

CNN Overview:



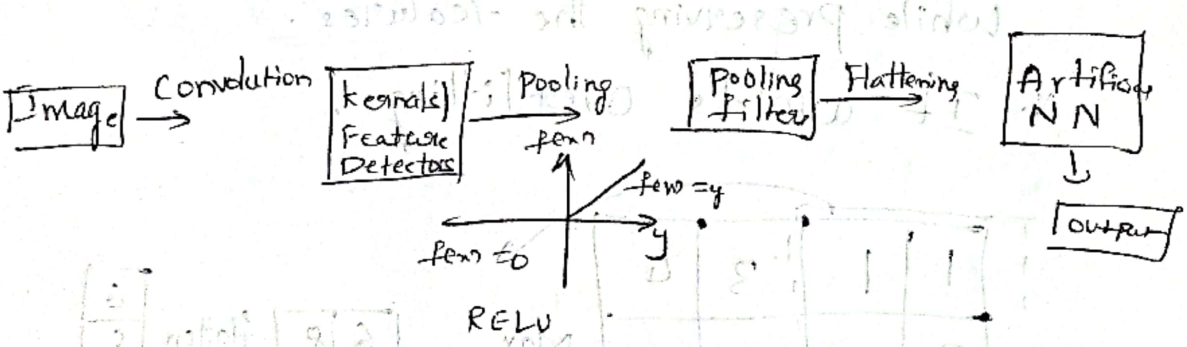
Step1: Convolution (feature extraction / feature detection)

Step2: adding (RELU Layer) Activation function

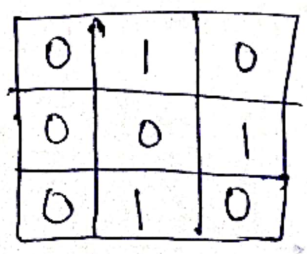
Step3: Applying pooling Layer (Down sampling) (like Compressing an image) (which will improve the Computional performance of the Network and reduces the Computational complexity)

Step4: Flattening (pixes, we make them in one array)

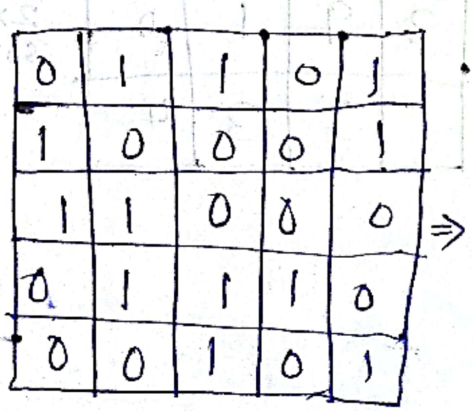
Step5: Feeding it in Artificial neural network.



Feature map:



Feature Detector



Image

Feature map

4. RELU Layer:

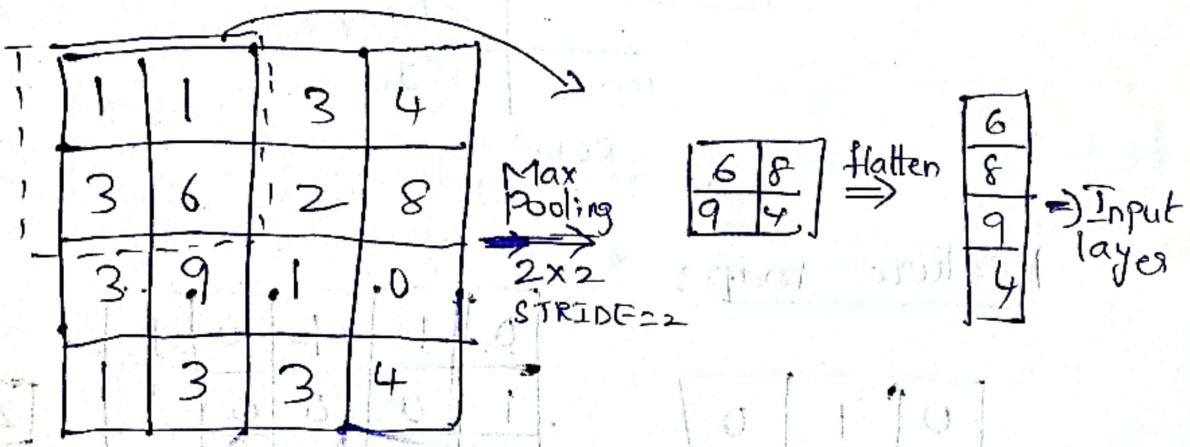
This is used to add non linearity in the feature map

It gives: $f(y) = \begin{cases} 0 & \text{for } y < 0 \\ y & \text{for } y \geq 0 \end{cases}$

It enhances the sparcity or how scattered the feature map is.

Pooling (Down Sampling):

- Pooling or down sampling layers are placed after convolutional layers to reduce feature map dimensionality.
- This improves computational efficiency while preserving the features.
- It avoids overfitting.

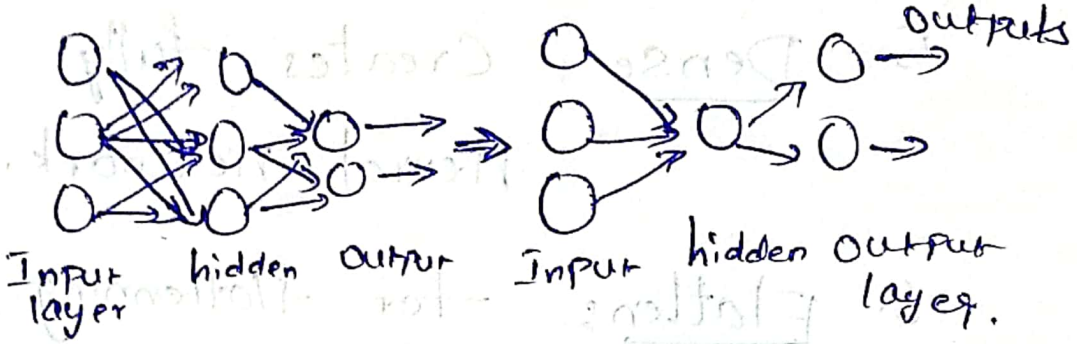


Improving Network performance:

* Improve accuracy by adding feature detectors / filters or adding a dropout.

Dropout ⇑ ⇒ Co-dependancy ↓ ⇒ Overfitting ↓.

64 instead of 32



Code Involves:

Step 1: Importing Libraries / Datasets.

Includes: pandas, numpy, matplotlib, seaborn.

Techstacks Used:

STEP #1: IMPORT LIBRARIES/DATASETS

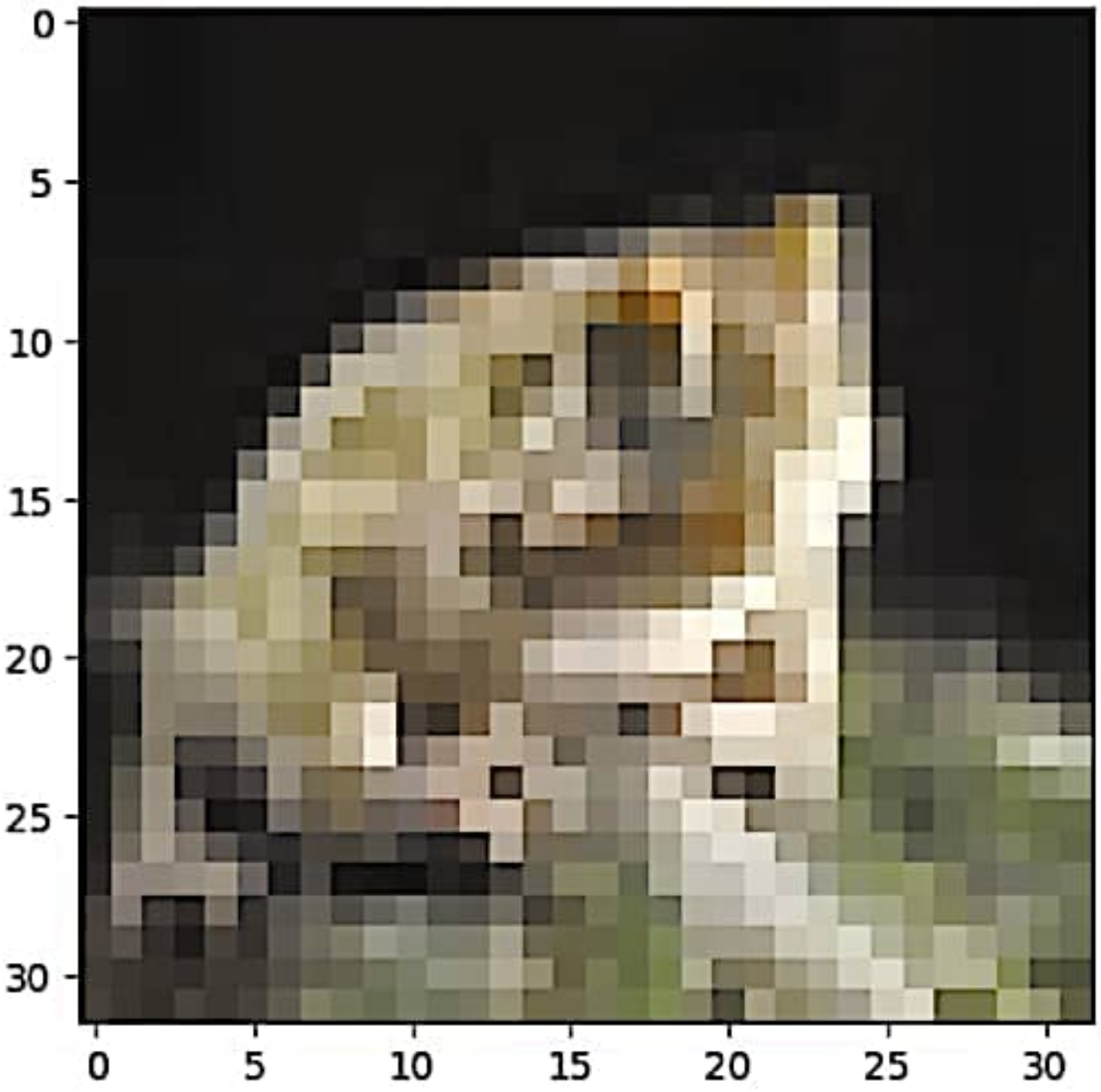
```
]  import pandas as pd
   import numpy as np
   import matplotlib.pyplot as plt
   import seaborn
```

Step 2: Visualising Data:

STEP #2: VISUALIZE DATA

```
i = 10001
plt.imshow(X_train[i])
print(y_train[i])
```

```
[6]
```



Step3: Data preparation:

This step involves, converting the data from decimal to binary numbers.

step4: Training the model.

Tech Stack used:

1. Sequential - Building our model in a sequence fashion.

2. Conv 2D: Convolution Process

3. Max pooling: will do down sampling

4. Average Pooling: we just average the values of the pixels.

5. Dense: Creates fully connected artificial neural network.

6. Flatten: for flattenning the map into a single array.

7. Dropout: performs regularization, means dropping a couple of neurons, just to improve the generalization capability of the network.

STEP #4: TRAIN THE MODEL

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D, Dense, Flatten, Dropout
from keras.optimizers import Adam
from keras.callbacks import TensorBoard
from keras.optimizers import RMSprop
```

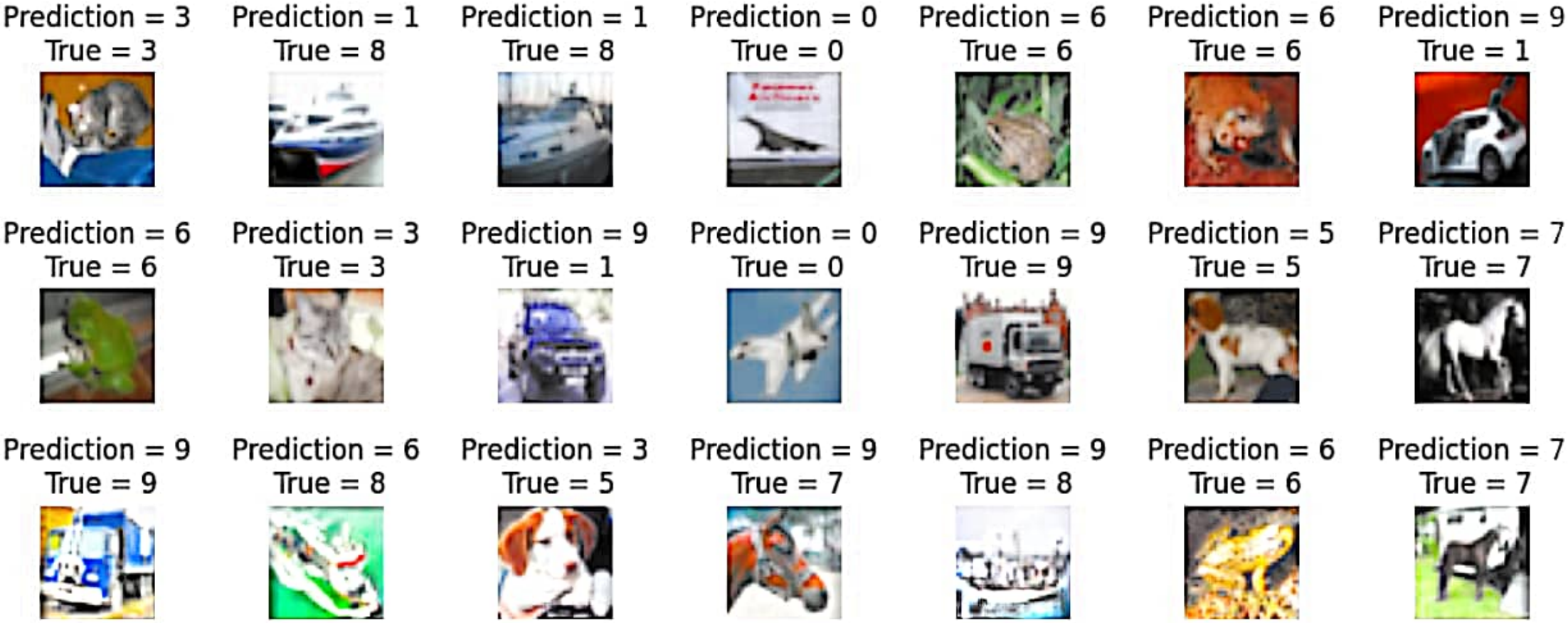
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
from keras.optimizers import RMSprop

cnn_model = Sequential()
cnn_model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', input_shape=Input_shape))
cnn_model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
cnn_model.add(MaxPooling2D(2, 2))
cnn_model.add(Dropout(0.4))
```

Step5:

Evaluating the model:

Obtained, outputs (predicted) are almost 90% accurate.

Note: Each and every time, it will take different set of images, so the accuracy might vary.



Confussion matrix:

This tells how many, images have been correctly categorized.

Seaborn is used to plot heatmaps

Heatmaps