CIFAR – 10 Image Classification Using Deep Convolutional Neural Network

Shabnam naz Reshmi

1. INTRODUCTION

- In this project, I have worked on image classification of the CIFAR-10 data set using Deep Convolutional Neural Network Algorithm. The dataset consists of 60,000 32x32 RGB images containing one of 10 object classes, with 6000 images per class. The CIFAR-10 dataset consists of 10 classes as shown in Figure [1] and is divided into 50,000 labelled training images and 10,000 unlabelled testing images.

Figure [1]: CIFAR-10 Classes

1.1 ABOUT DATASET

- CIFAR – 10 dataset contains images grouped into 10 unique classes (rows). Each class contains a subset of images belonging to the training dataset (50,000) and testing/validation dataset (10,000 images). There are total 60,000 images in dataset where in each class has 6,000 images of small resolution (32x32) RGB color images. The dataset stands for the Canadian Institute For Advanced Research (CIFAR). CIFAR-10 is widely used for machine learning and computer vision applications. Here is Data Source https://www.cs.toronto.edu/~kriz/cifar.html

2. PROBLEM STATEMENT

- To classify CIFAR-10 images into respective classes using Deep Convolutional Neural Network.

3. Data Science Methodology

3.1 Data Analysis & Visualization

- In the CIFAR-10 data Image is represented in the binary numbers format.
- A grey scale image is system of 256 tones with values ranging from 0-255. '0' represents black and '255' represents white.
- Numbers in-between represents greys between black and white.
- Binary systems use digits '0' and '1'where '00000000' for black, to '11111111' for white (8-bit image).
- Note: binary value of '11111111' is equal to decimal value of '255'.
- The labels are in numerical format from 1 for Airplanes to 10 for Truck.

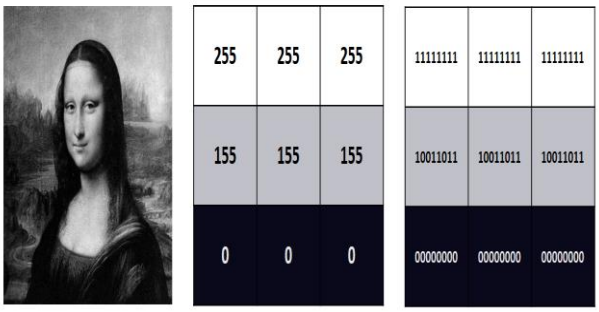

Figure [2]: https://www.pexels.com/photo/woman-art-painting-mona-lisa-40997/

3.2 Data Preparation

- I am using Keras which is a deep learning API written in Python, running on top of the machine learning platform TensorFlow where TensorFlow is a Deep Learning framework used for building complex Machine Learning models.
- Keras power faster computational results and is Highly scalable. Code here is much simpler than TensorFlow.

3.2.1 Understanding Convolutional Neural Network

- In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.
- Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and output an activation value. When you input an image in a ConvNet, each layer generates several activation functions that are passed on to the next layer.

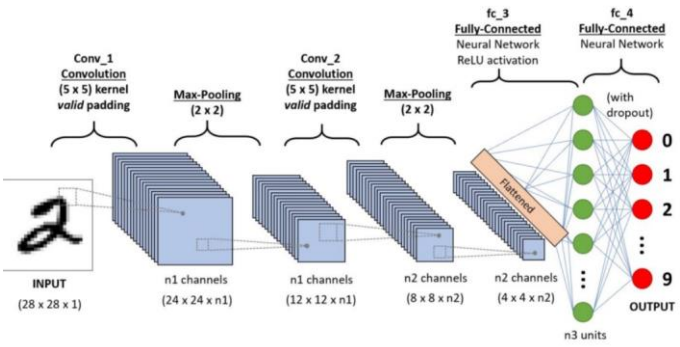

Figure [3]: CNN process

- The role of the ConvNet is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction. Convolutions use a kernel matrix to scan a given image and apply a filter to obtain a certain effect.
- An image Kernel is a matrix used to apply effects such as blurring and sharpening. Kernels are used in machine learning for feature extraction to select most important pixels of an image. Convolution preserves the spatial relationship between pixels.

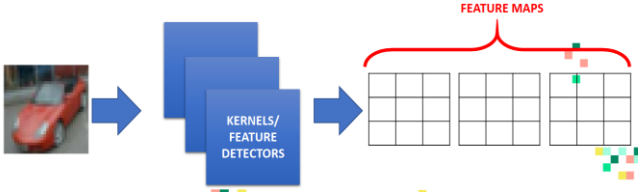

Figure [4]: CNN Feature Detector

- **Activation Function :** I am using RELU (Rectified Linear Units) Activation Function because in CNN most of the time we use RELU in Hidden layers due to the positive part of graph which is > 0 and keeps increasing and doesn't saturates with time.
- Layers are used to add non-linearity in the feature map.
- It also enhances the sparsity of how the feature map scattered is. The gradient of the RELU does not vanish as we increase x.

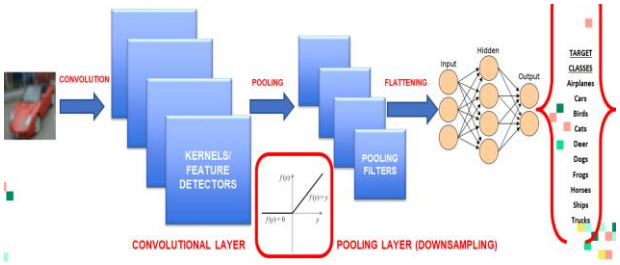

Figure [5]: RELU Activation Function

- For any negative number, it simply replaces with 0 as shown in Figure [6] therefore we get better results for Hidden layers.

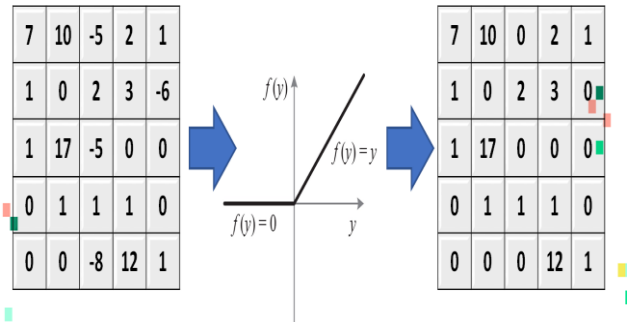

Figure [6]: RELU results

- **Pooling (Down sampling):** In order to improve the network performance Pooling or down sampling layers are placed after convolutional layers to reduce feature map dimensionality.
- This improves the computational efficiency while preserving the features.
- Pooling helps the model to generalize by avoiding overfitting.
- If one of the pixel is shifted, the pooled feature map will still be the same.
- Max pooling works by retaining the maximum feature response within a given sample size in a feature map.

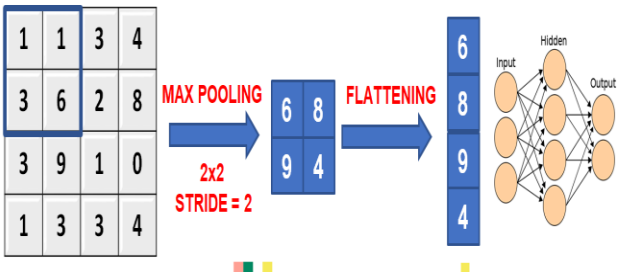

Figure [7]: Pooling on Data

- **Dropout:** Improve accuracy by adding more feature detectors/filters or adding a dropout.
- Dropout refers to dropping out units in a neural network.
- Neurons develop co-dependency amongst each other during training.
- Dropout is a regularization technique for reducing overfitting in neural networks.
- It enables training to occur on several architectures of the neural network.

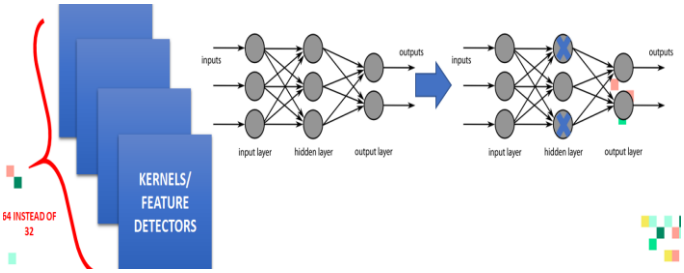

Figure [8]: Reduce Data using Dropout

- **Flattening:** This technique converts the 2-D array into 1-D which is then fed to model.

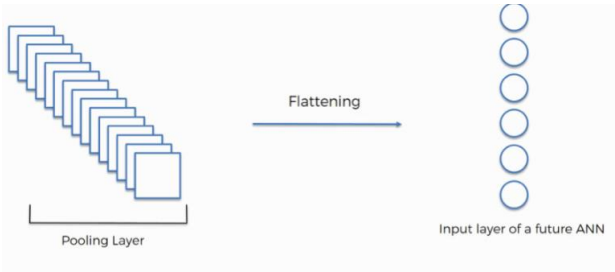

Figure [9]: Flattening Data

3.3 Model Training and Testing

- The data is split in 8:2 ratio and the model is trained on Sequential CNN model. Sequential is the easiest way to build a model in Keras. It allows you to build a model layer by layer.
- The first 2 layers are Conv2D layers. These are convolution layers that will deal with our input images, which are seen as 2-dimensional matrices.
- The 32 in the first layer and 64 in the second layer are the number of nodes in each layer.
- Kernel size is the size of the filter matrix for our convolution. So a kernel size of 3 means we will have a 3x3 filter matrix.
- The activation function we will be using for our first 2 layers is the ReLU, or Rectified Linear Activation. This activation function has been proven to work well in neural networks.
- Our first layer also takes in an input shape. This is the shape of each input image, 28,28,1 as seen earlier on, with the 1 signifying that the images are greyscale.
- In between the Conv2D layers and the dense layer, there is a 'Flatten' layer. Flatten serves as a connection between the convolution and dense layers.
- 'Dense' is the layer type we will use in for our output layer. Dense is a standard layer type that is used in many cases for neural networks. I will have 10 nodes in our output layer, one for each possible outcome (0–9).
- The activation is **'softmax'**. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.
- **Compiling the model:** Compiling the model takes three parameters: optimizer, loss and metrics.
- The optimizer controls the learning rate. I will be using RMS optimiser. We can use Adam for best results.
- The learning rate determines how fast the optimal weights for the model are calculated. A smaller learning rate may lead to more accurate weights (up to a certain point), but the time it takes to compute the weights will be longer.
- I will use 'categorical_crossentropy' for our loss function. This is the most common choice for classification. A lower score indicates that the model is performing better. To make things even easier to interpret, we will use the 'accuracy' metric to see the accuracy score on the validation set when we train the model.
- **Training Data Results:** With batch_size = 32, learning_rate = 0.0001 and for 10 epochs the model resulted ***70.49% accuracy and 84.09% loss*** which is still better result for 10 epochs. This accuracy can further improved by improving the epochs.

```
# Train model
history = cnn_model.fit(X_train, y_train, batch_size = 32, epochs = 10, shuffle = True)

Epoch 1/10
1563/1563 [==============================] - 141s 90ms/step - loss: 1.8151 - accuracy: 0.3308
Epoch 2/10
1563/1563 [==============================] - 138s 88ms/step - loss: 1.4796 - accuracy: 0.4596
Epoch 3/10
1563/1563 [==============================] - 138s 88ms/step - loss: 1.3307 - accuracy: 0.5210
Epoch 4/10
1563/1563 [==============================] - 138s 88ms/step - loss: 1.2194 - accuracy: 0.5656
Epoch 5/10
1563/1563 [==============================] - 137s 88ms/step - loss: 1.1275 - accuracy: 0.5995
Epoch 6/10
1563/1563 [==============================] - 133s 85ms/step - loss: 1.0530 - accuracy: 0.6284
Epoch 7/10
1563/1563 [==============================] - 131s 84ms/step - loss: 0.9906 - accuracy: 0.6532
Epoch 8/10
1563/1563 [==============================] - 130s 83ms/step - loss: 0.9331 - accuracy: 0.6729
Epoch 9/10
1563/1563 [==============================] - 141s 90ms/step - loss: 0.8889 - accuracy: 0.6871
Epoch 10/10
1563/1563 [==============================] - 138s 88ms/step - loss: 0.8409 - accuracy: 0.7049
```

Figure [10]: Training Data

4. MODEL EVALUATION

- The model is evaluated on Test Data and it resulted in ***69.99 ~ 70% accuracy with 85.66% loss*** which is much better in most case. However, the model performance can be improved by tweaking CNN model hyper-parameters such as learning rate, epochs, batch size, layers etc.
- From Figure [11] we can see the Predicted and True value are same for majority of the sample images.

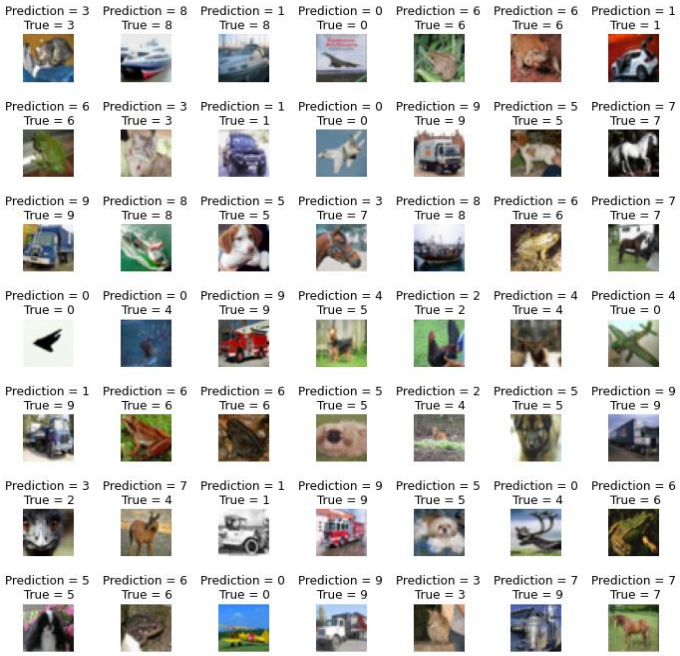

Figure [11]: Predicted v/s True value for sample data

5. CONCLUSION & FUTURE SCOPE

- In this project, I have achieved 70% accuracy by building a straightforward model. However, the model can be further improved by tweaking the hyper-parameters.
- Additionally, as part of future scope different algorithms such ensemble techniques such as FCNN (Fully Connected Neural Network) can also be experimented. There is room for improvement in any case.
- Another way to improve is by using Data Augmentation. Image Augmentation is the process of artificially increasing the variations of the images in the datasets by flipping, enlarging, rotating the original images. Augmentations also include shifting and changing the brightness of the images.
- Real world data can be in any way therefore Image Augmentation would serve well for the purpose by further adding more features for model computation.

6. REFERENCES

- Udemy – Machine Learning Practical Workout By Dr. Ryan Ahmed
- CIFAR-10 IMAGE CLASSIFICATION USING FEATURE ENSEMBLES, https://arxiv.org/pdf/2002.03846.pdf
- IMAGE CLASSIFICATION WITH CIFAR 10 DATASET, http://www.jctjournal.com/gallery/19-aug2021.pdf
- Introduction to Convolutional Neural Networks (CNN), https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/
- Convolutional Neural Networks (CNN), https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening
- Building a Convolutional Neural Network (CNN) in Keras, https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5