Convolution Neural Networks for CIFAR-10 multiclass image classification

Melroy Elias Pereira
University of Adelaide
melroyelias.pereira@student.adelaide.edu.au

Abstract

The Image classification is the important area in the computer vision, where it is used in robotics, such as defense to track the enemy troops using drones, security it is used for face recognition system to check the past details of a person's crime, therefore, to work any of the above we need to have a classifier system where it classifies the objects, thus Image classification was introduced. Hence, in this paper, we will be classifying the images of the CIFAR10 dataset, where it contains the images of 10 objects and we use computer to classify the images into a set of labels by learning similar images. The Computer uses an Algorithm to do such image classification and nowadays there many algorithms, hence in this paper we will use Convolution neural network (CNN) architecture to classify the images. Many papers are published based on CNN to classify the CIFAR10 images and the papers we used are found to have a maximum accuracy of 99.4%, however, my proposed approach was able to get 70% accuracy using 5 epochs.

1. Introduction

Artificial Intelligence (AI) has been explored in many areas of the field, and it is growing rapidly, and around the globe, all the developed countries are now using AI in most of the work therefore we can say that AI is exploring each and every day in every corner of the world, and one of the use cases is "Computer Vision", the word itself describes that there is something for the computer by which can it has vision, like in human beings we have eyes, the computer uses cameras as a vision.

Computer vision has been used in every sector such as defense, security, robotics, self-driving cars, face recognization, etc. however, if the human being had only eyes and no processing capabilities then it would be just nothing, but when our eyes see something it sends the information for brain where it processes the detail, therefore we tried to mimic the idea and we are now using neural network in computer vision as a part intelligence driven by the data.

Computer vision uses the algorithm as a source of extracting information from the data which is then used for training the data to extract the hidden patterns and later which is tested on test data.

In this paper, we are classifying the images of CIFAR10 Dataset and we are using the Convolution neural network as an algorithm. In this dataset we have 60,000 images and of which 50000 images are used for training phase and 10,000 images are used for the testing phase which we call it as testing the unseen data, and the results are evaluated using the actual data and the classified data and results get succeeded if there is a small error in the testing data.

In this paper, I have implemented 12 different models, where 6 models are tested on two specific cases, and we observed that many papers used higher epochs for training which takes more time for computation and requires expensive GPU to train therefore, we kept condition that without using much epochs we get better accuracy, because as the number of epochs increases the time taken for training the data increases, therefore we uses baseline as 50% accuracy at epoch 5, and using these models I was able to achieve 70% of accuracy at epoch 5.

In summary, my contribution to the literature can be listed as follows;

- Training the data with less epoch to achieve better accuracy, so the computation time decreases and less use of expensive hardware's like GPU.
- Testing with more number of filters and checking its affects on the accuracy change.
- Testing with a higher size of kernel/filters and checking its affect on the accuracy change.
- Testing with a higher size of Max Pooling and checking its affect on the accuracy change.
- Using Adam optimizer, batch normalization and dropout regularization to improve the accuracy.
- Finally, trying to code the CNN from scratch to customize the network

The Organization of this paper as follows; section 2 is a Related work, section 3: starts with giving brief on the terminology of CNN architecture, section 4: starts with a Simple convolution neural network model (Base Model)

and other proposed method to improve the accuracy of base model. Section 5: gives detailed explanation on the experimental analysis. Section 6: explains the result comparison with other state of the art papers, section 7: is a conclusion and future work, and section 8: is a code section, section 9: is a References and section 10 is a supplementary material for the algorithms.

2. Related Work:

CIFAR10 dataset is a kind of small version Imagenet benchmark, and many researchers have been worked on CIFAR10 dataset, where in the data it has 10 different classes.
Researchers have applied many types of algorithm on this dataset. Such as spainal net by H M Dipu Kabir[3] et al, Big transfer by Alexander Kolesniko[2] et al and found to have an accuracy of 99.4%
The deep convolution neural network also been applied by Alex Krizhevsky[4] et al and found to have an error as 15.3%.
In this literature. My proposed approach I have analaysed the behavior of accuracy as the number of filters and size of the filters increased and was able to get an accuracy of 70%.

3. Convolution Neural Network (CNN)

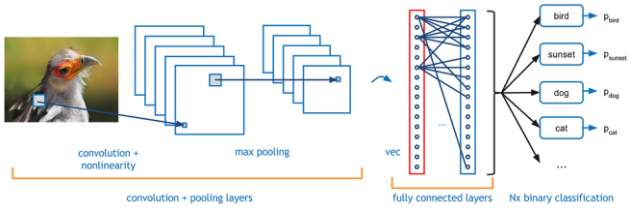


Figure 1: Convolution Neural Network.
The image was taken from [1]

The Convolution Neural Network is a type of deep learning network where it is used for image classification/recognition which is a part of the computer vision. However, the use case of CNN is not only limited in computer vision ,but it is also used in many other field such as, Natural Language Processing (NLP), Time series forecasting, etc. However, this paper primarily focuses on the use case of Image classification of CIFAR 10 dataset using CNN.
From the figure (1) CNN has following key important terminologies:

1) Convolution operation
2) Max pooling
3) Flattening for fully connected layer
4) Fully connected layer
5) Classification using Softmax

Convolution Operation

The convolution operation is a mathematical operation where two functions integrates together and explains the way it changed by the other function.
(*I have written python code for the convolution operation from scratch for1 channel and 3 channels and 1 filter, it is shown in the supplementary material)*
Since, Convolution operation is a integration of two functions, then output of the array shape will be according to the following equation.

Output dimension form CNN is given as follows;

$$out = \frac{(N + 2*P - F)}{S} + 1 \qquad \text{- (1)}$$

Where,
N is input pixel size
P is a padding for the input
F is a filter/kernel size
S is a Stride

Max Pooling

Max Pooling is an important operation used for CNN and while working with this paper I found that,:

- Max Pooling will reduce the size of the inputs that it receives.
- As more and more number of stride is given the total shape of the data is reduced faster.
- However, I found that using more and more number of strides will also ignores some of the high pixel value while comparing the maximum of all the set, and this is will cause loosing some of the informations.
- By using too small Max Pool, we will get better information rather than large Max Pool.
- Most of the research papers used Max Pool size of (2*2) and stride = 2.

(*I have written python code for the Max Pool operation from scratch and it is shown in the supplementary materials,)*
Since, max pool operation reduces the size of the input array therefore, the output of the array shape will be according to the following equation.

$$out = \frac{(N + 2*P - F)}{S} + 1 \qquad \text{- (2)}$$

Where,
N is input pixel size
P is a padding for the input

F is a filter/kernel size
S is a Stride

Flatten and Fully connected Layer:

Flattening is used before the fully connected layer to get the values in vector form and the fully connected layer will pass through the neurons with different layers in forward propagation and similarly the weights are then updated using back propagation algorithm and softmax is used for classifying the image.
(*I have written python code for the Fully connected operation from scratch (MLP) and it is shown in the supplementary materials)*

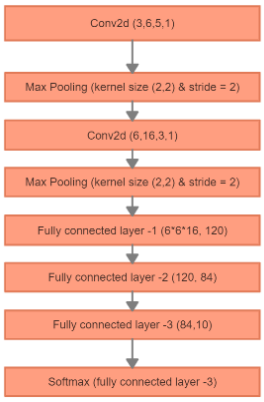
4. Methods and algorithm:

In this paper we have implemented 12 models from Pytorch and there are 2 models that are written from the scratch such as Multi-layer perceptron (MLP) and Forward propagation in Convolution Neural Network (CNN).

The 12 (including six sub-models) Pytorch implemented models are as follows;

1) Simple convolution neural network (Base Model)
2) Base Model
   2.1) CNN model 2.1
   2.2) CNN model 2.2
   2.3) CNN model 2.3
3) Base Model + Batch Normalization model
   3.1) CNN model 3.1
   3.2) CNN model 3.2
   3.3) CNN model 3.3
4) Base model + including padding and stride
5) Base model + dropout regularization using SGD
6) Base model + dropout regularization using Adam

Model 1: Simple Convolution neural Network



Flowchart 1: Model

This is a simple Convolution neural network model and is considered as a base model for this paper.

In this neural network model we have

- 2 convolution layer
- 2 Max pooling layer
- 3 fully connected layer

The first convolution layer has 3 channels of input image (3,32,32), and 6 filters/kernels of (5*5) size and striding is equal to one. After passing through the first layer we get (28,28,6) as output and later we pass the inputs to the Max pooling operation where the kernel size is (2,2) with stride is equal to two thus, the output is (14,14,6).

In the second layer we have another convolution with 6 channels, and 16 filters/kernels of (3*3) filter size here, outputs from first layer is used as inputs here we get, (12,12,6) as output and later it is again passed to the Max pooling operation, and the output from the second layer is (6,6,16), which is then flattened to (6*6*16 = 576) as inputs to the fully connected layers.

First fully connected layer has 120 neurons and the outputs from the previous layer given the as the input and after passing from the first layer we have (120, 576) as output by using activation function called as Rectified linear unit (ReLU). Later, it is then again passed to the another fully connected layer where it has (84 neurons) and gives the output as (84, 120) and then it is passed to the final layer of the model, which has 10 neurons and gives the output as (10) classes using softmax .
In this model we were able to get 60% testing data accuracy.

Model 2: Tweaking the Base Model

In this model the architecture used is same as the Base model, however, we tried to see the behavior of the CNN accuracy by changing number of filters. The analysis is shown in the experimental analysis section, however, working process is given below.

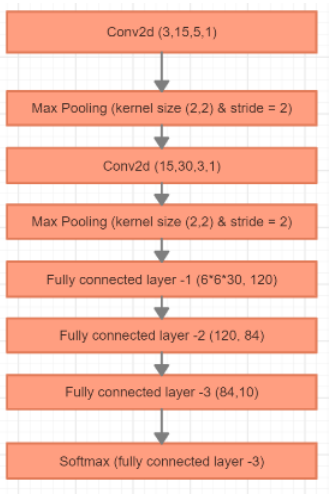
Sub-models:

- Sub-Model 2.1

In this sub-model we used 15 filters of (5*5) size kernel in the first layer and we got the output of (28,28,15)for the first convolution layer and then it is passes to the Max pooling operation where it reduces the size of images by further (14,14,15) which is later passed to the second layer.

In the second layer we use 30 filters for of (3*3) size kernel

and gives the output as (12,12,30) which is then again passed to the max pooling operation and reduces the size of image by further (6,6,30).



Flowchart 2: sub-model 2.1

In order to pass the input to the fully connected layer, we need to flatten the image values that is (6*6*30=1080) and then passes to the fully connected layer of nodes 120 in first layer which gives the output as (120, 1080) which is then used by the layer 2 fully connection and gives the result as (the second convolution layer(84,120) and sent to the last layer containing (10) neuron and gives the output as for ten classes. The accuracy obtained using this model is 50%.

- Sub-model 2.2

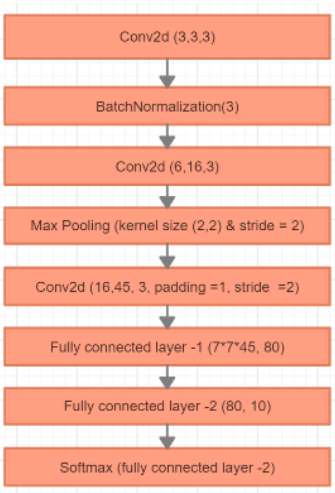
In this sub-model 2.2 the architecture is same as sub-model 2.1 and In this layer 1, we use 30 filters of (5*5) sizes and then it is passed to the max pooling containing kernel size of (2,2) and after finishing, first layer we pass the outputs to the next layer containing 60 filters of (3*3) size filter which is then again passed to the max pool of (2,2) and the output of this is then passed to the fully connected layer having same neurons in the layers of sub-model 2.1.
The accuracy obtained by the model is 40% which is dropped by 10% from last model.

- Sub-model 2.3

In this sub-model 2.3 the architecture is same as sub-model 2.1 and In this layer 1, we use 60 filters of (5*5) sizes and then it is passed to the max pooling containing kernel size of (2,2) and after finishing the first layer we pass the outputs to the next layer containing 120 filters of (3*3) size filter which is then again passed to the max pool of (2,2) and the output of this is then passed to the fully connected layer having same neurons in the layers of sub-model 2.1.
The accuracy obtained by the model is 0% which is dropped completely to 0% from last model.

From these sub-models we can see that , even though using all hyper-parameters same and only changing filters we get different accuracy, and the accuracy obtained is getting lower and lower as the number of filters increased.
We know that filters are used for finding low level features such as edges, curves and using many filters we might introduce unwanted details to the image to the neural network.
Analysis of these models is shown in Experiment Analysis.

Model 3: Base Model + Batch Normalization model



Flowchart 3: Model 3

In this Model 3, the CNN had 3 convolutions layer and one Max pooling operation and two fully connected layers and one batch normalization and the activation function used for within the layer is Rectified linear unit (ReLU) and for the last layer we used softmax for multi-class classfication.

In the first layer we used one convolution layer of (3,3,3) where it has 3 filters of (3*3) shape filter for the 3 input feature image (3,32,32)

After passing through the first layer we have the input as (30,30,3) and we pass this through batch normalization to normalize the values of output to the input of the second layer convolution layer of (3,16,3) where it has 16 filters of (3*3) filter size for the input of (30,30,3) image and gives the output as (28,28,16) which is then passed to the Max Pooling with kernel size of (2,2) and stride = 2. Which gives the output of (14, 14, 16).

After passing through the second layer we pass the

(14,14,16) image to the third convolution layer, where it has (16,45,3,padding = 1, stride = 2), that is , 45 filters of (3*3) size filter for the 16 feature input image with adding one more layer of padding =1 and stride =2, which gives output as (7,7,45) image.

After passing through the third layer we pass the image to the Fully connected layer by flattening (7,7,45) into (2205) into 80 node layer and gives the output as (80, 2205) and later it is passed to the second fully connected layer of 10 nodes, it has softmax as a activation function and gives the output as classes of (10).

In this model I was able to get accuracy of 70%.

For this model we have three other sub-models, where we will check the behavior of CNN accuracy as the filter size is increased. Working process for the sub-models are given below and experimental analysis is shown in experiment analysis section.

- Sub-model 3.1:

In this sub-model we used 3 filters of (7*7) size for the first convolution layer and later output of first layer is Max pooled with kernel size of (2,2) and stride = 1, later the output of max pool is fed into second layer convolution network which used 3 filters of (9*9) filter size and the output of this layer is late used for the max pool which uses kernel size of (2,2) and stride = 2 for the second convolution layer. After the convolution operation we use the output to flatten so that it can be used for fully connected layers and in first fully connected layer has (120 neuron) and second layer with (84 neuron) and the last layer has (10neuron) which gives the output using softmax function.
The accuracy obtained using this model is 50%.

- Sub-model 2

In this sub-model we used 3 filters of (13*13) size for the first convolution layer and later output of first layer is Max pooled with kernel size of (2,2) and stride = 1, later the output of max pool is fed into second layer convolution network which used 3 filters of (13*13) filter size and the output of this layer is late used for the max pooling, which uses kernel size of (2,2) and stride = 2 for the second convolution layer. After the convolutions operation we use the output to flatten so that it can be used for fully connected layers of firs layer with (120 neuron) and second layer with (84) and the last layer has10 neuron which gives the output using softmax function.
The accuracy obtained using this model is 40%.

- Sub-model 3

In this sub-model we used 3 filters of (15*15) size for the first convolution layer and later output of first layer is Max pooled with kernel size of (2,2) and stride = 1, later the output of max pool is fed into second layer convolution network which used 3 filters of (7*7) filter size and the output of this layer is late used for the max pooling, which uses kernel size of (2,2) and stride = 2 for the second convolution layer. After the convolutions operation we use the output to flatten so that it can be used for fully connected layers and it has (120 neuron) in the first fully connected layer and second layer with (84 neuron) and the last layer has10 neuron which gives the output of (10) classes using softmax function.
The accuracy obtained using this model is 40%.

We can see that the accuracy is dropping as the size of filter is increasing, but it also depends upon the total dataset we use, because if we use too small data and if we use large filter then we don't get a chance to convolute all the pixels of the image so the details from the image will be missing.

Model 4: Base model + including padding and stride

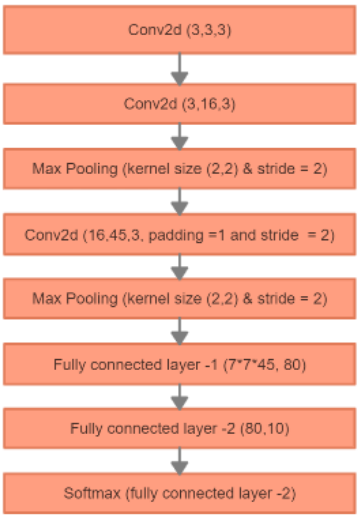


Figure 4: Model 4

In this Model 4, the CNN had 3 convolutions layer and two Max pooling operation and two fully connected layers and the activation function used for within the layer is Rectified linear unit (ReLU) and for the last layer we used softmax for multi-class classification

In the first layer we used one convolution layer of (3,3,3) where it has 3 filters of (3*3) shape filter for the 3 input feature image (3,32,32)

After passing through the first layer we have the input as (30,30,3) and we pass this input to the second layer convolution layer of (3,16,3) where it has 16 filters of (3*3) filter size for the input of (30,30,3) image and gives the

output as (28,28,16) which is then passed to the Max Pooling with kernel size of (2,2) and stride = 2. Which gives the output of (14, 14, 16).

After passing through the second layer we pass the (14,14,16) image to the third convolution layer, where it has (16,45,3,padding = 1, stride = 2), that is , 45 filters of (3*3) size filter for the 16 feature input image with adding one more layer of padding =1 and stride =2, which gives output as (7,7,45) image.

After passing through the third layer we pass the image to the Fully connected layer by flattening (7,7,45) into (2205) into 80 node layer and gives the output as (80, 2205) and later it is passed to the second fully connected layer of 10 nodes, it has softmax as a activation function and gives the outpur as 10 classifications.

The output from the softmax is a prediction for the classes and it is evaluated using loss function Cross entropy loss.

In this model, I was able to achive 70% accuracy and the loss of 1.35.

Model 5: Base model + dropout regularization using SGD (Stochastic Gradient Descent)

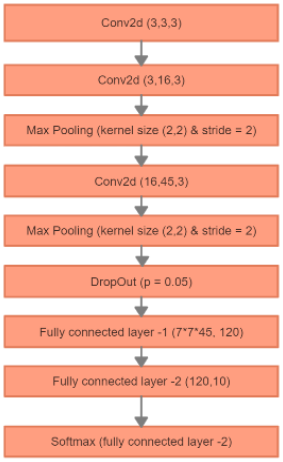


Figure 5: Model 5

In this Model 5,the CNN had 3 convolutions layer and two Max pooling operation and two fully connected layers and the activation function used for within the layer is Rectified linear unit (ReLU) and for the last layer we used softmax for multi-class classification, however we also use dropout regularization of p = 0.05.

In the first layer we used one convolution layer of (3,3,3) where it has 3 filters of (3*3) shape filter for the 3 input feature image (3,32,32)

After passing through the first layer we have the input as (30,30,3) and we pass this input to the second layer convolution layer of (3,16,3) where it has 16 filters of (3*3) filter size for the input of (30,30,3) image and gives the output as (28,28,16) which is then passed to the Max Pooling with kernel size of (2,2) and stride = 2. Which gives the output of (14, 14, 16).

After passing through the second layer we pass the (14,14,16) image to the third convolution layer, where it has (16,45,3), that is , 45 filters of (3*3) size filter for the 16 feature input image which gives output as (14,14,45) and then It is passed to the Max Pooling of (2,2) with stride = 2 and gives the output of (6,6,45) image.

After the third layer we pass the input through the drop out then it is passed to the fully connected layer by flattening (6*6*45 = 1620) to the 120 node layer and it gives the output of (120, 1620) which is then again passed to the next fully connected layer of 10 nodes and now it gives the output of (10) classes which is the clsssification from the softmax activation function.

In this model I was able to get accuracy of 60%.

Model 6: Base model + dropout regularization using Adam

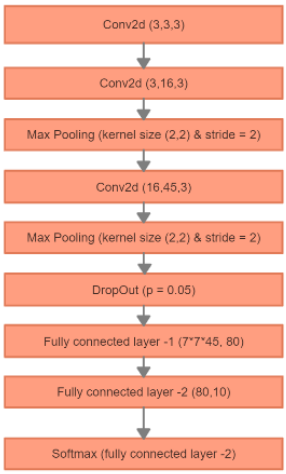


Figure 6: Model 6

In this Model 6,the CNN had 3 convolutions layer and two Max pooling operation and two fully connected layers and the activation function used for within the layer is Rectified linear unit (ReLU) and for the last layer we used softmax for multi-class classification, however we also use dropout regularization of p = 0.05.

In the first layer we used one convolution layer of (3,3,3) where it has 3 filters of (3*3) shape filter for the 3 input feature image (3,32,32).

After passing through the first layer we have the input as (30,30,3) and we pass this input to the second layer convolution layer of (3,16,3) where it has 16 filters of (3*3)

filter size for the input of (30,30,3) image and gives the output as (28,28,16) which is then passed to the Max Pooling with kernel size of (2,2) and stride = 2. Which gives the output of (14, 14, 16).

After passing through the second layer we pass the (14,14,16) image to the third convolution layer, where it has (16,45,3), that is , 45 filters of (3*3) size filter for the 16 feature input image which gives output as (14,14,45) and then It is passed to the Max Pooling of (2,2) with stride = 2 and gives the output of (6,6,45) image.

After the third layer we pass the input through the drop out then it is passed to the fully connected layer by flattening (6*6*45 = 1620) to the 80 node layer and it gives the output of (80, 1620) which is then again passed to the next fully connected layer of 10 nodes and now it gives the output of (10) classes which is the clsssification from the softmax activation function. In this model I was able to get accuracy of 70%.

5. Experimental Analysis

In this paper we used Convolution neural network for classifying the images of CIFAR 10 and we were able to get 70% as the highest accuracy score from all the models. In this paper we implemented 12 models (including 6 sub-models) and the accuracy of these models are given in the table below.

Models	Accuarcy
Simple convolution neural network (CNN) – Base Model	60%
Base Model + CNN Model 2.1	50%
Base Model + CNN Model 2.2	40%
Base Model + CNN Model 2.3	0%
Base Model + Batch Normalization Model	70%
Base Model + CNN Model 3.1	50%
Base Model + CNN Model 3.2	40%
Base Model + CNN Model 3.3	40%
Base Model + including padding & stride	60%
Base Model + dropout regularization model using SGD	60%
Base Model + dropout regularization model using Adam	70%

Table 1: Performance of the models

From the Model 1, we were able to get an accuracy of 60% and it is better to show how does this model performed while train and test phase.

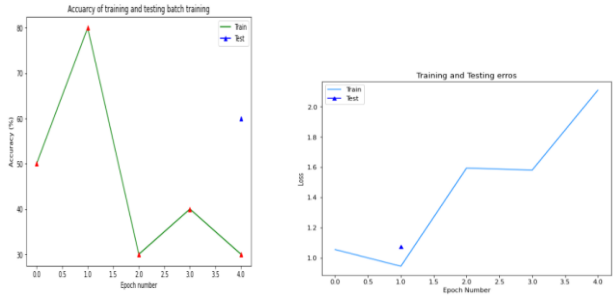


Figure 2: Accuracy & Loss vs Number of filters used

From the above performance chart, we can see that accuracy of the model while training (green line) and accuracy of the model in testing data (Blue point).

We can see that in The figure(2) when the accuracy was 80 at one epoch the loss in the figure(2) was below 1.0, however, the loss has been increased just right after this epoch and started to over fit. However, in our test accuracy we were closer to best results of train data and we got accuracy of

60% with closer to 1.0 loss.

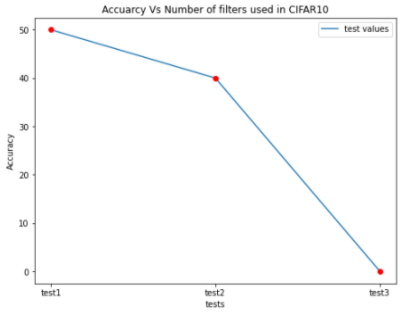


Figure 3: Accuracy vs Number of filters used

Tests 1 = [filter 1: 15 & 30]

Test 2 = [filter 2: 30 & 60] and Test 3 = [filter 3: 60 & 120]

From the model 2 and it's sub-models we found that, as the number of filters used for depth increased, the accuracy dropped. We know that, filters are used for finding edges, curves and other low features. However, for smaller dataset if we use large filters then it might not work better as It would work for larger dataset, there might be a loss of detail capturing so it couldn't give better accuracy.

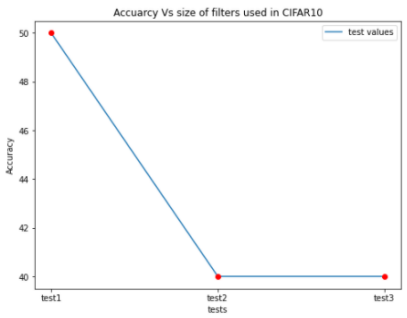
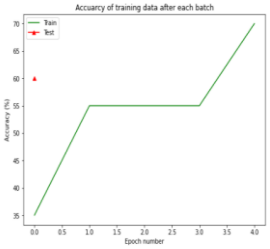


Figure 4: Accuracy vs size of the filters used

Tests 1 = [filter 1 (7*7) & (9,9)]
Test 2 = [filter 2(13, 13) & (13, 13)] and Test 3 = [filter 3 (15*15) & (7*7)]
From the model 3and its sub-models we can see that the accuracy dropped as the filter size increased in each tests, it could be because it was not able to pick all the points because small image and the filter was used here is bigger. So, the accuracy dropped, however, using small size filters in the earlier models we were able to get better accuracy.



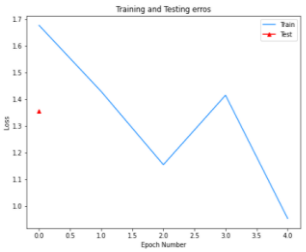


Figure 5: Accuracy & Loss vs Number of filters used

From the Model 4 results, we were able to get an accuracy of 60% at loss near to 1.3 in the figure(4), however, it is observed that at epoch 4, we have very loss loss and the highest train accuracy of 73.33%

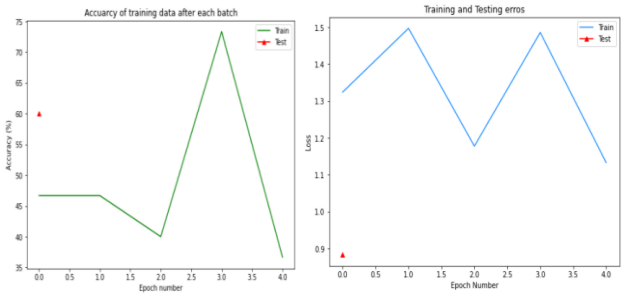


Figure 6: Accuracy vs Number of filters used

From the Model 5 results, we were able to get an accuracy of 60% at loss near to 0.5 in the figure(5), however, it is observed that at epoch 2, we had low loss in training phase, but in testing phase we got loss lower than the train phase.

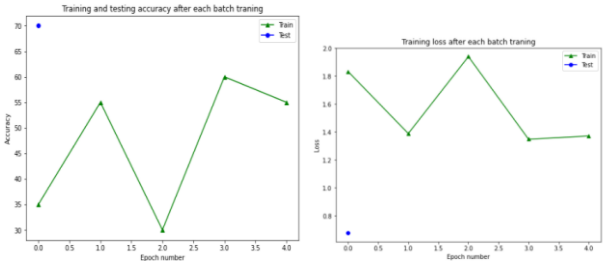


Figure 7: Accuracy & Loss vs Number of filters used

From the Model 6 results, we were able to get an accuracy of 70% at loss near to 0.5 in the figure(6), however, it is observed that at epoch 3, we had high accuracy in training phase but in testing phase we got better than the training phase with lesser loss compared to training phase.

6. Comparative study

A comparative study between my proposed approach and the other state of the art papers is shown in the table .

It is clear from the table that my approach was able to classify better than the spinal net, and the expectation of this paper was reached by crossing the baseline of 50% accuracy at 5 epoch. The algorithms used in the state of the art papers are better and advanced and can extract more information from the images, and they were able to achieve more than 95% accuracy with more training time.

Author	Model	Result
S.H Shabbeer Basha, et al	CIFAR-VGG variants	Accuracy = 92.05%
Alexander Kolesnikov, et al	Big Transfer (BiT)	Accuarcy = 99.04%
H M Dipu Kabir, et al	Spinal Net	Accuracy = 62%
Alex Krizhevsky, et al	Deep CNN with Fisher's vector	Test error = 15.3%

Table 2: Comparative table

7. Conclusion and future work

In this literature, for classifying the images of CIFAR10

dataset, we used Convolution neural network and we were able to achieve 70% accuracy, which is higher than the baseline accuracy of 50% at epoch 5, however, to improve the accuracy we used batch normalization method, dropout regularization, and Adam optimization. We have also analyzed the case of change in accuracy as the number of filters and filter size increased.

My future work on CNN are as follows;

- I want get more into CNN because, as of now I am having limited knowledge on it, so I will look into research papers.

- I want explore more about images and its pixels for processing because, if we use better data, we can get better prediction, so I want to know more about processing of an image.

- Finally, Continuing the scratch code of forward propagation of CNN with back ward propagation inorder to customize the architecture, and applying that to CIFAR10 to check the accuracy.

8. Code section:

I have uploaded the python notebook file in the github, With repository name as "CIFAR-10".
Here is the link to get the notebook files
https://github.com/Optimus-Q/CIFAR-10

9. References

[1]https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[2] S.H Shabbeer Basha, Shiv Ram Dubey, Viswanath Pulabaigari, Snehasis Mukherjee. Impact of fully connected layers on performance of Convolution neural network for image classification, 2019 Journal of arxviv:1902.02771v3

[3] Alexander Kolesnikov, Lucas Beyer , Xiaohua Zhai , Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big Transfer (BiT) general visual representation learning, 2020, arXiv:1912.11370v3

[4] H M Dipu Kabir, Moloud Abdar, Seyed Mohammad Jafar Jalali, Abbas Khosravi, Member, IEEE; Amir F Atiya, Senior Member, IEEE; Saeid Nahavandi, Dipti Srinivasan, Fellow, IEEE.Actual, SpinalNet: Deep neural network with gradual input.

[5] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, Imagenet classification deep convolutional neural networks.

10. Supplementary Material:

Algorithm for convolution operation:

Step 1: Select the size of the filter/kernel

Step 2: Get the shape of input image that is total columns, rows.

Step 3: Now while convoluting make sure that you are within these total shape of column and rows limit.

Step 4: Initializing the parameters: startC = 0 (Start of the column), startR = 0 (Start of the row), endC = 0 (End of the column), endR = 0 (End of the rows), Padding P, Stride S.

Step 5: Start from the (0,0) position of array/tensor and move along the columns side/ row side as per stride.

Step 6: Select the first channel of the image and extract

Step 7: calculated convolution will be saved in list, and for next and every iteration use the following conditions:

strtC = startC + stride
*if (total column – startC) >= filter size (eg: 3*3)*
then continue along moving column for the convolution and not changing any endR.

else:
move to next row using startR = startR + stride and
endR = startR + filter and endC = startC + filter

if (total rows – startR < filter)
then exist from the convolution operation using break statement.
And continue for the other two channels.
the elements of the array according to the shape of filter and compute the element multiplication and after the operation use overall summation to calculate convolute value.

Algorithm for Max Pooling operation:

The Algorithm is similar to the convolution operation:

*Step 1: Select the size of the Max Pool, preferably (2*2).*

Step 2: Get the shape of input image that is total columns, rows.

Step 3: Now while max pooling make sure that you are within these total shape of column and rows limit.

Step 4: Initializing the parameters: startC = 0 (Start of the

column), startR = 0 (Start of the row), endC = 0 (End of the column), endR = 0 (End of the rows), Padding P, Stride S.

Step 5: Start from the (0,0) position of array/tensor and move along the columns side/ row side as per stride.

Step 6: Select the first channel of the image and extract the elements of the array according to the shape of filter and select the maximum of all those array element.

Step 7: calculated max pool value will be saved innthe list, and for next and every iteration use the following conditions:

strtC = startC + stride
*if (total column – startC) >= filter size (eg: 2*2)*
then continue along moving column for the max pooling and not changing any endR.

else:
move to next row using startR = startR + stride and endR = startR + filter and endC = startC + filter

if (total rows – startR < filter)
then exit from the max pooling operation using break statement.
And continue for the other two channels.

Algorithm for CNN in pytorch

Step 1: Import Pytorch libraries
Step 2: Get the CIFAR 10 dataset from the inbuilt pytorch function.
Step 3: Train and Test data is then passed to the DataLoader, where we give input parameters for the batch size of the data for each epoch in training.
Step 4: Define the neural network and call it using any object so as to access it's methods.
Step 5: Initiate the loss function and optimizer Algorithm.
Step 6: In this base model we used Stochastic Gradient Descent is used as a optimizer.
Step 7: Loop over the training phase and at each epoch calculate the accuracy score of training data.
Step 8: Reset the gradient from pytorch and call the optimizer to update the weight
Step 9: After getting minimum loss values, pass the weights for prediction .

Pyhton code for Forward propagation of CNN:

```
import numpy as np

Convolution operation:

def ConVolution_Operation(dataset, padding, stride, channels, filter):

### INITALIZARION...

 data = dataset
 data = np.pad(data, pad_width = padding, mode = 'constant', constant_values = 0 )

 f = filter.shape[1]
 arrC = int(data.shape[1])
 arrR = int(data.shape[0])


 convC, convR , startC ,startR , endC , endR = 0,0,0,0,0,0

 endC = endC+f
 endR = endR +f

 nf = int(((arrC + 2*padding - f)/stride)+1)
 n_c = channels
 nf_sq = nf*nf
 #emp = np.empty(shape = (n_c, nf_sq))
 convS = []
 #conv_vault = []

#### CONVOLUTION...

 for i in range((int(nf))*int(nf)):

  convA = data[startR:(endR), startC:(endC)]
  conv = np.multiply(convA, filter)
  conv = np.sum(conv)
  convS.append(conv)
  startC +=stride

  if ((arrC - startC) >= f):
   startR = startR
```

```
    endC = startC + f
    endR = startR + f
  else:
    startR = startR +  stride
    startC = 0
    endC = startC + f
    endR = startR + f

  if ((arrR -startR) < f):
    #emp[channel] = convS
    #conv_vault.append(emp)
    break;

return (np.asarray(convS).reshape(-1, int(nf)))
```

Max pool:

```
def MaxPool(p, f, stride, data):
  mpool = []
  startC, startR, endC, endR = 0,0,0,0
  n = data.shape[0]
  nf = int(((n+2*p-f)/stride)+1)
  arrC = np.shape(data)[1]
  arrR = np.shape(data)[0]
  endC += f
  endR +=f
  for nfs in range(nf*nf):
    dat = data[startR:endR, startC:endC]
    mx = np.max(dat)
    mpool.append(mx)
    startC += stride

    if ((arrC - startC) >= f):
      startR = startR
      endC = endC + stride
      endR = startR + stride
    else:
      startR = startR +  stride
      startC = 0
      endC = startC + stride
      endR = startR + stride

    if ((arrR -startR) < f):
      #emp[channel] = convS
      #con_vault.append(emp)
      break;

  return (np.asarray(mpool).reshape(-1, nf))


X = np.random.randint(low = 1, high = 4, size = (32,32))

filter = np.array([[1,0,-1], [1,0,-1], [1,0,-1]])
```

Defining the network:

```
conv1 = ConVolution_Operation(dataset = X, padding = 0, stride = 1, channels = 3, filter = filter)

conv2 = ConVolution_Operation(dataset = conv1, padding = 0, stride = 1, channels = 3, filter = filter)

pool1 = MaxPool(0,2,2,conv2)

conv3 = ConVolution_Operation(dataset = pool1, padding = 0, stride = 1, channels = 3, filter = filter)

pool2 = MaxPool(0,2,2,conv3)
```

MLP:

```
import numpy as np
from sklearn.metrics import accuracy_score, f1_score


def sigmoid(z):
  A = 1/(1+np.exp(-z))
  return (A)


def GetIntialWeights(layers):
  np.random.seed(50)
  weights = {}
  for weightNode in range(1, len(layers)):
    weights['W'+str(weightNode)] = np.random.randn(layers[weightNode], layers[weightNode-1])
    weights['W'+str(weightNode)][:, 0:1] = 0
  return (weights)


def ForwardNetwork(weight, x):
  vault = []
  Z = np.dot(weight, x)
  A = sigmoid(Z)
  elements = [weight,x,Z,A]
  vault.append(elements)
  return (A,vault)

def ForwardCall(x, layers, weight):
  a = x
  W = weight
  fc_vault = {}
  for layer in range(1, len(layers)):
    A, vault = ForwardNetwork(W['W'+str(layer)],a)
    a = A
    fc_vault['FC'+str(layer)] = np.asarray(vault)
  return (A, fc_vault)


def Get_dLdA(fc_vault, total_layers, Y):

  dA  = {}
  L = total_layers
  A = fc_vault['FC'+str(L-1)][0][3]
  DA_L =  - (np.divide(Y, A) - np.divide(1 - Y, 1 - A))
```

```
    dA['dA'+str(L-1)] = DA_L

    for layers in reversed(range(1, L-1)):

      # FROM CHAIN RULE FOR DL/DA
      old_dA = dA['dA'+str(layers+1)]
      A = fc_vault['FC'+str(layers+1)][0][3]
      dAdZ = A * (1 - A)
      w = fc_vault['FC'+str(layers+1)][0][0]

      #CALCULATING DA
      DA_L = np.multiply(old_dA, dAdZ)
      DA_L = np.dot(w.T, DA_L)
      dA['dA'+str(layers)] = DA_L

    return (dA)


def Get_dLdZ(dA, fc_vault, total_layers):

    dZ = {}
    L = total_layers
    for layers in range(1, L):

###CALCULATING DL/DZ FROM CHAIN RULE

      DA_L = dA['dA'+str(layers)]
      A = fc_vault['FC'+str(layers)][0][3]
      dAdZ = A * (1- A)
      dLdZ = np.multiply(DA_L, dAdZ)

      dZ['dZ'+str(layers)] = dLdZ

    return (dZ)


def Get_UpdatedWeights(dZ, fc_vault, total_layers, lr):

    W = {}
    L = total_layers
    ##INPUT LAYER TO LAYER 1...

    dLdZ = dZ['dZ1']
    A = fc_vault['FC1'][0][1]    # PREVIOUS A value
    dLdW = np.dot(dLdZ, A.T)
    w = fc_vault['FC1'][0][0]
    W['W1'] =  w - (lr*dLdW)

    ## REST CONNECTED LAYERS...

    for layers in range(2, L):

      dLdZ = dZ['dZ'+str(layers)]
      A = fc_vault['FC'+str(layers-1)][0][3]    # PREVIOUS
A value
      dLdW = np.dot(dLdZ, A.T)
      w = fc_vault['FC'+str(layers)][0][0]

      W['W'+str(layers)] =  w - (lr*dLdW)

    return (W)


def decision_fn(A):
    dec = np.where(A > 0.5, 1, 0)
    return(dec)

def predict(W, Xtest, Ytest, layers):
    L = len(layers)
    ypred, vault = ForwardCall(Xtest, layers, W)
    ypred = decision_fn(ypred)
    return (ypred)


def MultiLayerPerceptron(layers, X, Y, lr, simulation):
    w = []
    costs = []
    m = X.shape[0]
    W = GetIntialWeights(layers)

    for iter in range(simulation):

      L =  len(layers)

      w.append(W)

      A, fc = ForwardCall(x, layers, W)
```

```
    loss = np.dot(Y.T, np.log(A)) + np.dot((1-Y.T),np.log(1-
A))

    cost = -np.sum(loss)/m

    costs.append(cost)

    das = Get_dLdA(fc, L, Y)

    dzs = Get_dLdZ(das, fc, L)

    W = Get_UpdatedWeights(dzs, fc, L, lr)

  return (W, costs, w)

layers = [8,50,30,20,15,8,6,5,4,3,1]
x = np.random.randn(8, 50)
lr = 0.0001
simulation = 1000
para, costs, W = MultiLayerPerceptron(layers, x, Y, lr, si
mulation)

Y = np.array([1,0,1,1,0,0,1,1,0,0,1,1,1,1,0,0,1,1,0,0,1,1,0,
1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,1,0,1,0,1,0,1]).resha
pe(-1,50)
```