Machine Learning Engineer Nanodegree

Capstone Project

Puneet Sharma

December 24th, 2017

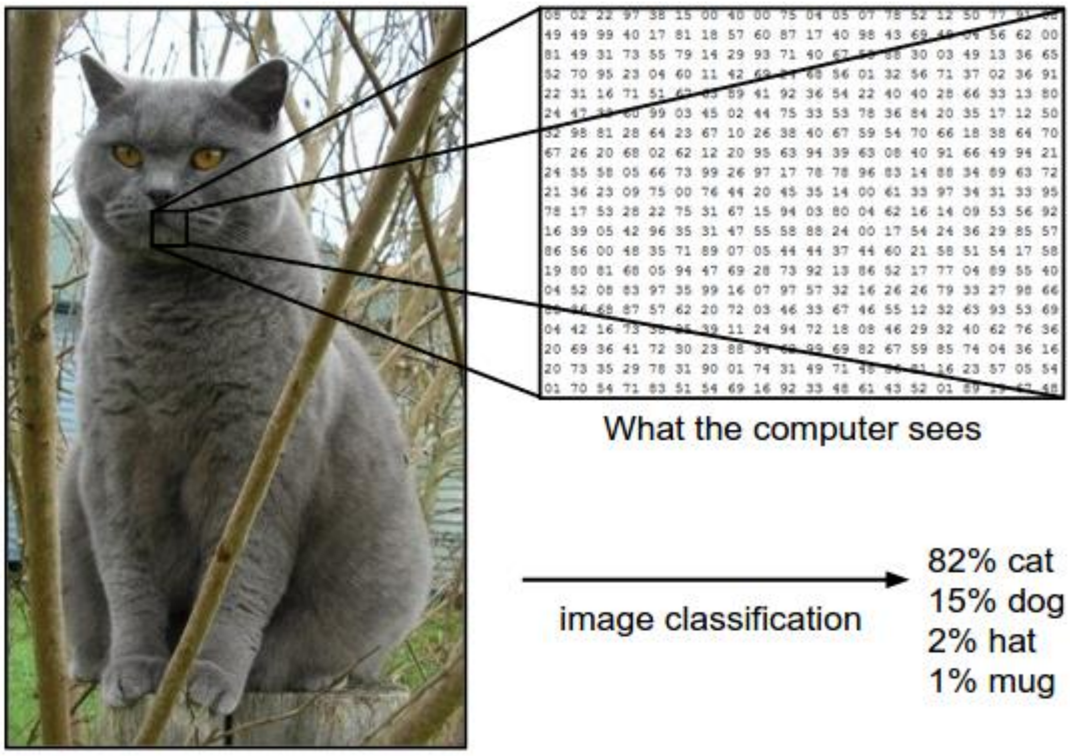
Image Recognition

Our brains make vision seem easy. It doesn't take any effort for humans to tell apart a lion and a jaguar, read a sign, or recognize a human's face. But these are actually hard problems to solve with a computer: they only seem easy because our brains are incredibly good at understanding images.

In the last few years the field of machine learning has made tremendous progress on addressing these difficult problems. In particular, we've found that a kind of model called a deep convolutional neural network can achieve reasonable performance on hard visual recognition tasks -- matching or exceeding human performance in some domains.

Image classification is the task of assigning an input image one label from a fixed set of categories. This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications. Moreover, many other seemingly distinct Computer Vision tasks (such as object detection, segmentation) can be reduced to image classification.

For example, in the image below an image classification model takes a single image and assigns probabilities to 4 labels, *{cat, dog, hat, mug}*. As shown in the image, to a computer an image is represented as one large 3-dimensional array of numbers. In this example, the cat image is 248 pixels wide, 400 pixels tall, and has three colour channels Red, Green, Blue (or RGB for short). Therefore, the image consists of 248 x 400 x 3 numbers, or a total of 297,600 numbers. Each number is an integer that ranges from 0 (black) to 255 (white). Our task is to turn this quarter of a million numbers into a single label, such as *"cat"*.



The task in Image Classification is to predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image. Images are 3-dimensional arrays of integers from 0 to 255, of size Width x Height x 3. The 3 represents the three colour channels Red, Green, Blue.

So we will using a similar problem of classifying objects in this project. The dataset we will be using is CIFAR-10. CIFAR-10 is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 colour images containing one of 10 object classes, with 6000 images per class. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are some links similar to this problem - https://www.learnopencv.com/image-recognition-and-object-detection-part1/

https://medium.com/@Killavus/naive-approaches-to-solving-cifar10-dataset-image-classification-problem-overview-and-results-636048ff1cc1

Problem Statement

The CIFAR-10 data set consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. Recognizing photos from the cifar-10 collection is one of the most common problems in the today's world of machine learning. We will be working on this problem in this project. We will train an algorithm that can classify between these objects.

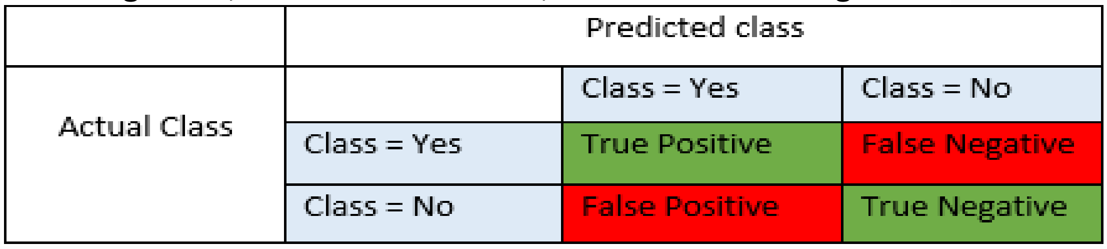
Solution

Recognizing images from the cifar-10 collection is a great classification problem and deep learning is the best way to solve this problem as recent advances in deep learning made tasks such as Image and speech recognition possible (classification algorithms like Support Vector Machines, Decision Trees, Random Forests, Gradient Boosting and K-Nearest Neighbours does not perform as well as the CNN). So, we will be using Deep Learning to solve is problem. Deep Learning is a subset of Machine Learning Algorithms that is very good at recognizing patterns but typically requires a large number of data. Deep learning excels in recognizing objects in images as it's implemented using 3 or more layers of artificial neural networks where each layer is responsible for extracting one or more feature of the image (more on that later). Neural Network is a computational model that works in a similar way to the neurons in the human brain. Each neuron takes an input, performs some operations then passes the output to the following neuron.

Metrics

We will be using 'evaluate' metrics from keras (python deep learning library). It returns the loss value & metrics values for the model in test mode. I used 'accuracy' as my evaluation metrics for testing. The function 'evaluate' of keras library take the testing images and labels as input and use the 'accuracy' metric to evaluate the model. The computation is done in batches and we can specify the batch size (If unspecified, it will default to 32). Mathematical expression for accuracy is: Accuracy = TP+TN/TP+FP+FN+TN. Where TP is True Positives, TN is

True Negatives, FP is False Positives, and FN is False Negatives.



		Predicted class	
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

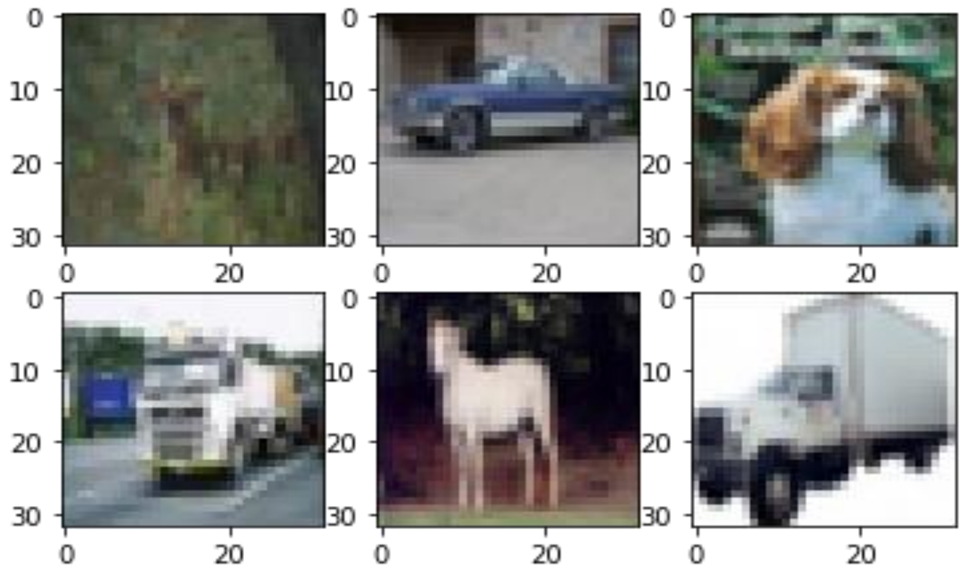
It is simply a ratio of correctly predicted observation to the total observations. I used accuracy as an evaluation metrics for my model because it is very simple and very "intuitive" measure and it is also easy to understand when it comes to visualization. I have also used 'categorical_crossentropy' as my loss function as it is very good for a model which predicts on multiple exclusive classes. ‘categorical_crossentropy’ is another term for multi-class log loss. Log Loss takes into account the uncertainty of your prediction based on how much it varies from the actual label. Log Loss Function is very useful to check whether our model is improving or not.

Data Exploration

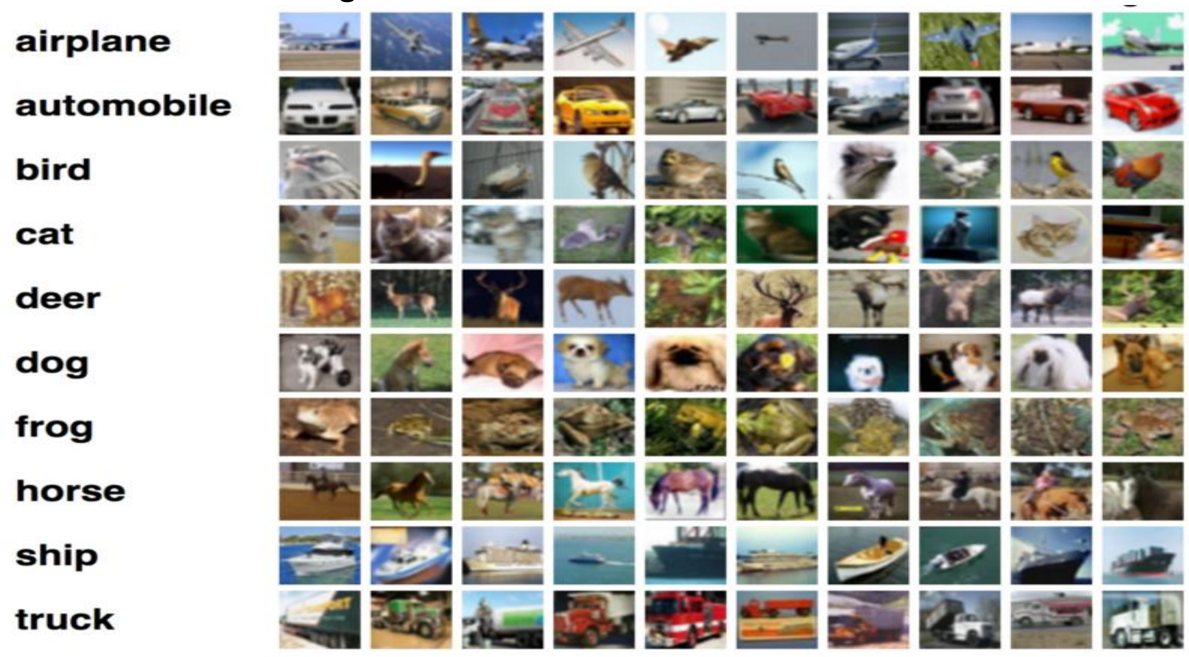
In this project we are using The CIFAR-10 dataset for our problem. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks. There are 10 different label classes for images in the dataset. Here is link of dataset - https://www.cs.toronto.edu/~kriz/cifar.html . We have converted the category information (labels) to one-hot encoding that is used by the final classification. For instance, we convert label=3 to a vector as [0 0 0 1 0 0 0 0 0 0] (labels starts from 0 to 9). We use this vector format in the classification to calculate training error at every step of the training.

Exploratory Visualization

Visualization of some random images from the dataset:

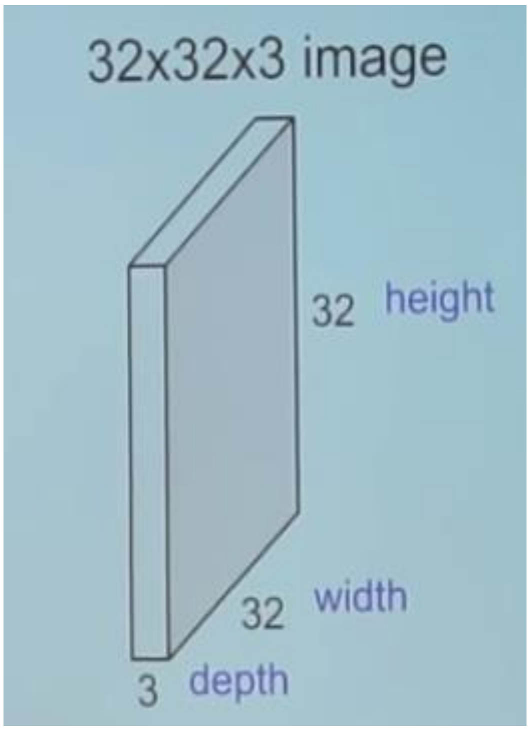


We also need to take a look at dimensions of the input images from the dataset so that we can decide whether it needs dimension reshaping or not before getting it as input to the neural networks.
Following were the shape of training and testing dataset:
Shape of training dataset:
(50000, 3, 32, 32)
Shape of testing dataset:
(10000, 3, 32, 32)
CIFAR-10 contains images of these 10 classes:



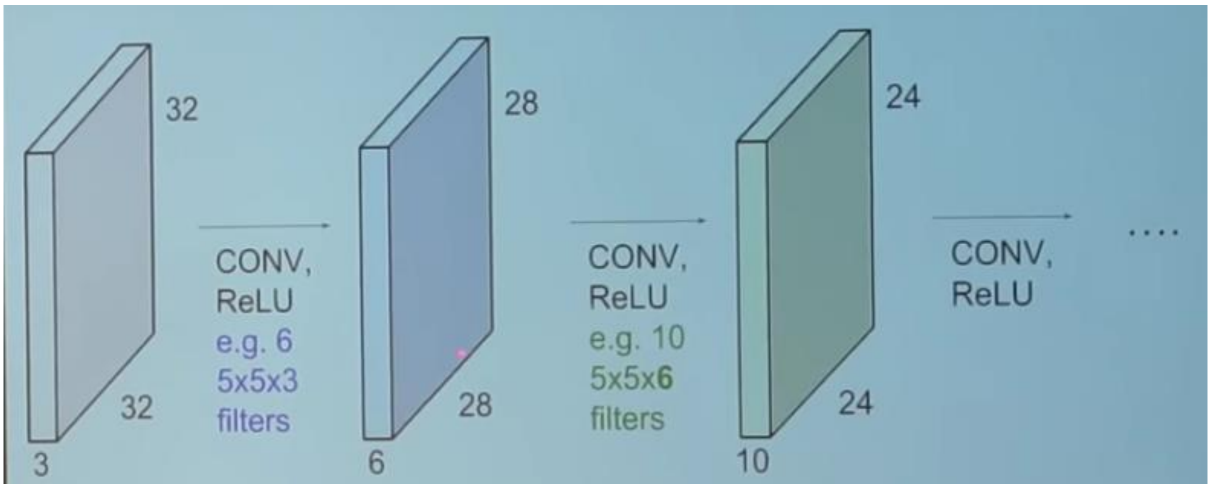
Convolutional Neural Network

CNNs have wide applications in image and video recognition, recommender systems and natural language processing. CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function.



Unlike neural networks, where the input is a vector, here the input is a multi-channelled image (3 channelled in this case).

The convolution layer is the main building block of a convolutional neural network. Every convolution layer decreases the width and height of image and increases the depth of image which results in learning new patterns within the image.


32
32
3
CONV, ReLU e.g. 6 5x5x3 filters
28
28
6
CONV, ReLU e.g. 10 5x5x6 filters
24
24
10
CONV, ReLU
....

We need three basic components to define a basic convolutional network.

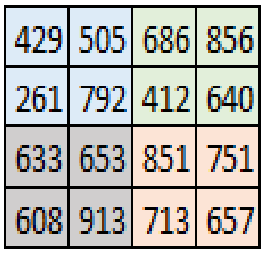
1. The convolutional layer
2. The Pooling layer[optional]
3. The Fully-Connected Layer

The Convolution Layer

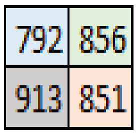
CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.

The Pooling Layer

Sometimes when the images are too large, we would need to reduce the number of trainable parameters. It is then desired to periodically introduce pooling layers between subsequent convolution layers. Pooling is done for the sole purpose of reducing the spatial size of the image. Pooling is done independently on each depth dimension, therefore the depth of the image remains unchanged. The most common form of pooling layer generally applied is the max pooling.



429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657



792	856
913	851

Here we have taken stride as 2, while pooling size also as 2. The max operation is applied to each depth dimension of the convolved output. As you can see, the 4*4 convolved output has become 2*2 after the max pooling operation.

The Fully-connected Layer

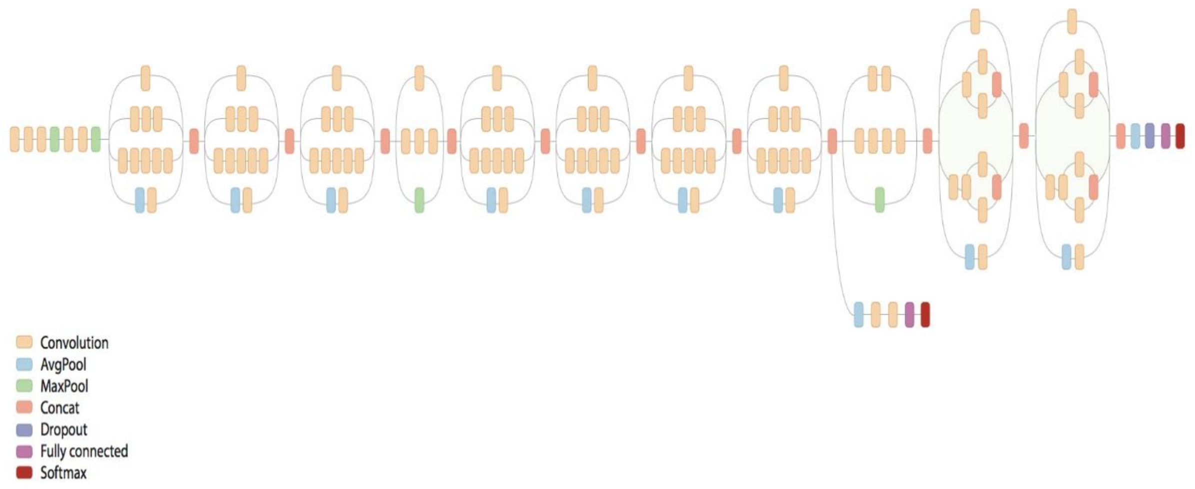
FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

Benchmark

Researchers have demonstrated steady progress in computer vision by validating their work against ImageNet -- an academic benchmark for computer vision. Successive models continue to show improvements, each time achieving a new state-of-the-art result: QuocNet, AlexNet, Inception (GoogLeNet), BN-Inception-v2. Researchers both internal and external to Google have published papers describing all these models but the results are still hard to reproduce. We'll be now using Inception-v3 (latest model) as our benchmark model.

Inception-v3 is trained for the ImageNet Large Visual Recognition Challenge using the data from 2012. This is a standard task in computer vision, where models try to classify entire images into 1000 classes, like "Zebra", "cars", and "Dishwasher".

This network achieves 21.2% top-1 and 5.6% top-5 error for single frame evaluation with a computational cost of 5 billion multiply-adds per inference and with using less than 25 million parameters. Below is a visualization of the model architecture.



Data Preprocessing

Normalizing Data

Each of the pixels that represents an image stored inside a computer has a pixel value which describes how bright that pixel is, and/or what colour it should be. In the simplest case of binary images, the pixel value is a 1-bit number indicating either foreground or background. For a grayscale images, the pixel value is a single number that represents the brightness of the pixel. The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to be black, and 255 is taken to be white. Values in between make up the different shades of gray. It is almost always a good idea to perform some scaling of input values when using neural network models. Because the scale is well known and well behaved. Here also the pixel values are in the range of 0 to 255 for each of the red, green and blue channels, we can very quickly normalize the pixel values to the range 0 and 1 by dividing each value by the maximum of 255 and this will normalize our data values to the range [0, 1]. It's good practice to work with normalized data. Because the input values are well understood.

Preprocess class labels for Keras

This is a multi-class classification problem. Here the labels is a list of 10000 numbers in the range 0-9. Currently our labels is an array containing 10 classes and we need the labels to be in 10 distinct classes. We can fix this easily by using one hot encoding. We can use a one hot encoding to transform them into a binary matrix in order to best model the classification problem. We know there are 10 classes for this problem, so we can expect the binary matrix to have a width of 10.

Implementation

Neural Networks receive an input (a single vector), and transform it through a series of *hidden layers*. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the “output layer” and in classification settings it represents the class scores. In CIFAR-10, images are only of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Networks take advantage of the fact that the input consists of images. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: **width, height, depth**.

The input images in CIFAR-10 are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). Input [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three colour channels R,G,B. The final output layer would for CIFAR-10 have depth of 10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

We will define a simple ConvNet for CIFAR-10 classification having a combination of 6 Convolutional layers with alternative drop out and maxpooling layers.

Every Convolutional layer will have ‘relu’ activation. ‘relu’ corresponds to Rectified Linear Units. Instead of sigmoids, most recent deep learning networks use rectified linear units (ReLUs) for the hidden layers. A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input.

$$f(x) = max(x, 0)$$

The concept of stride and padding

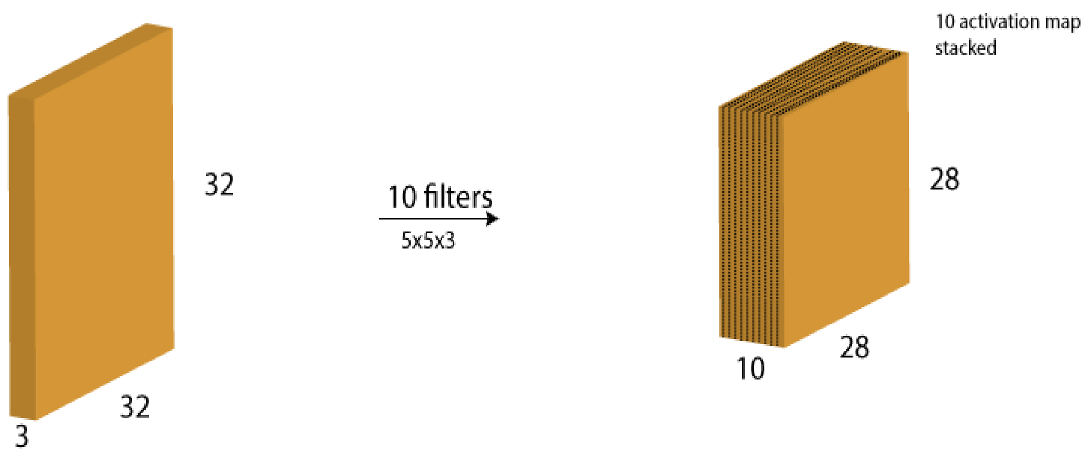
The filter or the weight matrix, moves across the entire image moves **one** pixel at a time. We can define it like a hyperparameter, as to how we would want the weight matrix to move across the image. If the weight matrix moves 1 pixel at a time, we call it as a stride of 1. When we have a stride of one we move across and down a single pixel. With higher stride values, we move large number of pixels at a time and hence produce smaller output volumes.

We are using 'same' padding. **Same padding** means that we considered only the valid pixels of the input image. In this we retained more information from the borders and have also preserved the size of the image.

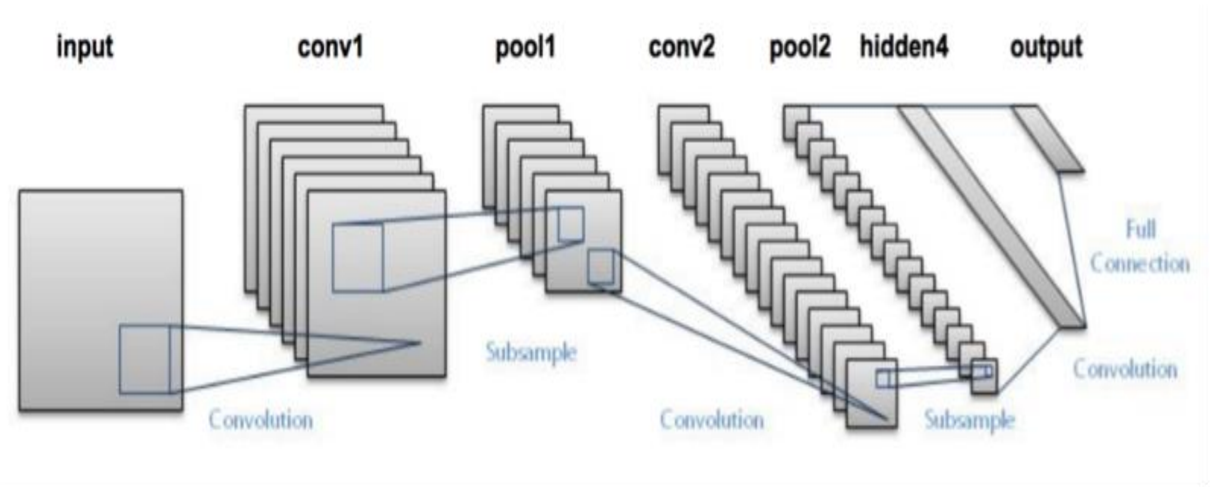
Now comes the dropout layer. **Dropout** is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. Dropout layers provide a simple way to avoid overfitting. The primary idea is to randomly drop components of neural network (outputs) from a layer of neural network. This results in a scenario where at each layer more neurons are forced to learn the multiple characteristics of the neural network.

Now comes the Pooling layers. POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].

Our aim is to decrease the height and width of the image while increasing the depth. This makes the model to learn more and more patterns within images. These patterns helps the model to predict on unseen dataset. We can visualize it as –



Putting it all together. Now our network will look similar to this –



This is the CNN architecture that I used in project for predicting on CIFAR-10 Dataset -

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 64)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         73856     
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 128)         0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         147584    
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                20490     
=================================================================
Total params: 307,498
Trainable params: 307,498
Non-trainable params: 0
```

There were some difficulties faced while defining the architecture as we need to take care of the dimensions while making an architecture. Like what dimensions architecture need as input and what dimensions are getting out after every layer, do we need to add dropout layer (if the model is overfitting. I was also confused on what should be the steps per epochs while fitting the model after data augmentation. The most difficult part was the implementation of benchmark model because we need to remove the last fully connected layer and add some new layers which can work for our dataset well. Changing dimension of images according to the needs of inception-v3 model was also a difficult task.

Refinement

Image Augmentation for Deep Learning

Deep networks need large amount of training data to achieve good performance. To build a powerful image classifier using very little training data, image augmentation is usually required to boost the performance of deep networks. Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips, etc.
An augmented image generator can be easily created using ImageDataGenerator API in Keras. ImageDataGenerator generates batches of image data with real-time data augmentation. This includes capabilities such as:

- Sample-wise standardization.
- Feature-wise standardization.
- ZCA whitening.
- Random rotation, shifts, shear and flips.
- Dimension reordering.
- Save augmented images to disk.

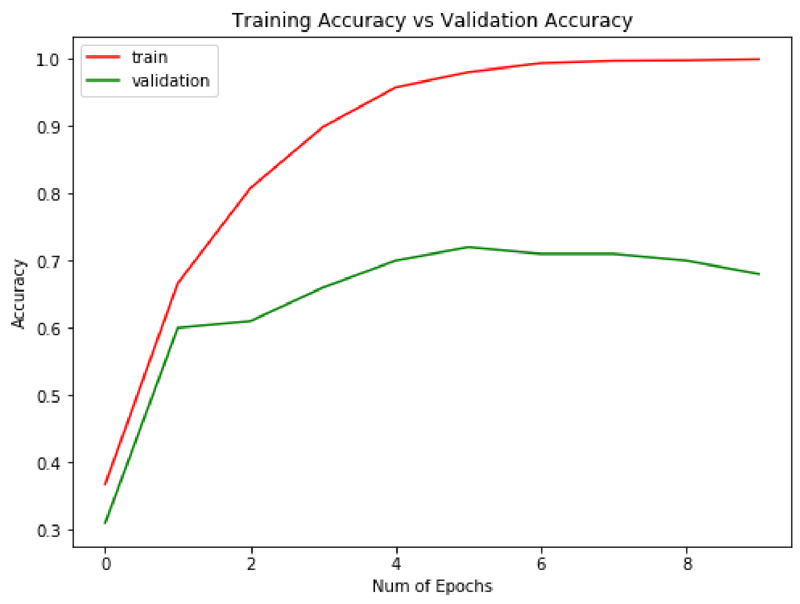
When we use Image Augmentation we need to change the steps per epoch. The new steps per epochs will be:

```
steps_per_epoch = number_of_samples
                  -----------------
                     batch_size
```

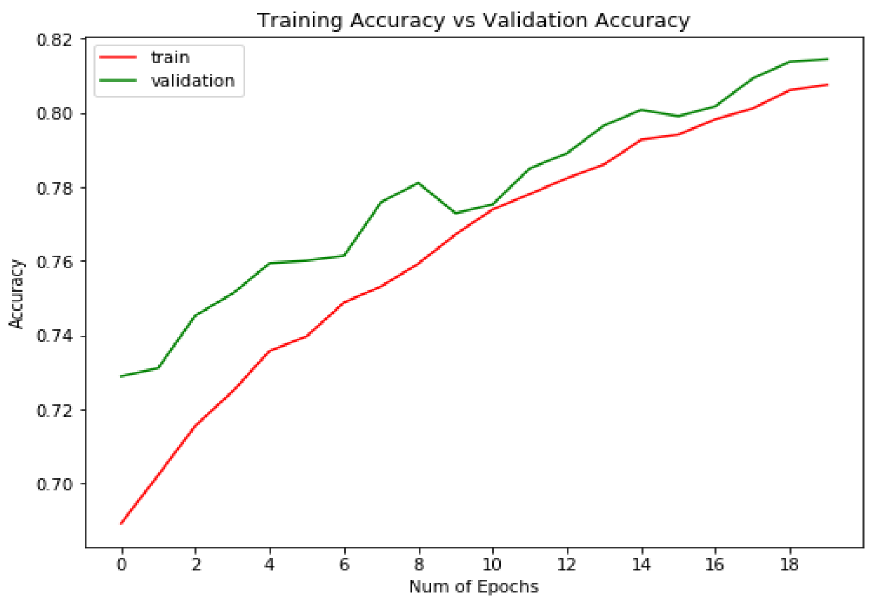
In my project the Image Augmentation increased the accuracy by 11% which is great improvement.

Model Evaluation and Validation

We have trained simple CNN algorithm of size 32x32 using Cifar-10 image set and evaluated on test set which got an accuracy of 70.39% which is good according to 20 epochs. After applying Data Augmentation my model got an accuracy of 81.45%.



This is Training and testing results of the benchmark model we can see clearly that the benchmark model is overfitting the data as the epochs goes up from 8 because the training accuracy is increasing and the testing accuracy attains its peak and then starts decreasing again. This is due to the reason that we trained the dataset only on 3000 images. This might need more regularization.



The results above are the training and testing results of the final model as we can see the model performs well on training data without compromising its performance on the test data. So here the model is not memorizing the images, it is truly learning patterns within images through different layers of CNN. So there is no underfitting and overfitting in the final model which makes the model robust enough for this problem as it can perform well on unseen dataset.
Yes, the final model aligns with solution expectations as it get above 80% accuracy on the unseen dataset which is good.

Justification

Benchmark model (Inception-v3) got an accuracy of 75.39% and my final model got an accuracy of 81.45% which is better than the benchmark model but the benchmark model can perform a lot better than this as we trained the model only on 3000 images for 10 epochs. If we train the model on the full dataset of 50000 images with a good amount of epochs the accuracy will definitely improve alot. I ran the model for only 3000 images and only for 10 epochs because of lack of time as i am running is code on CPU due to some problems. Running it on GPU will definitely improve performance.

Conclusion

Visualisations

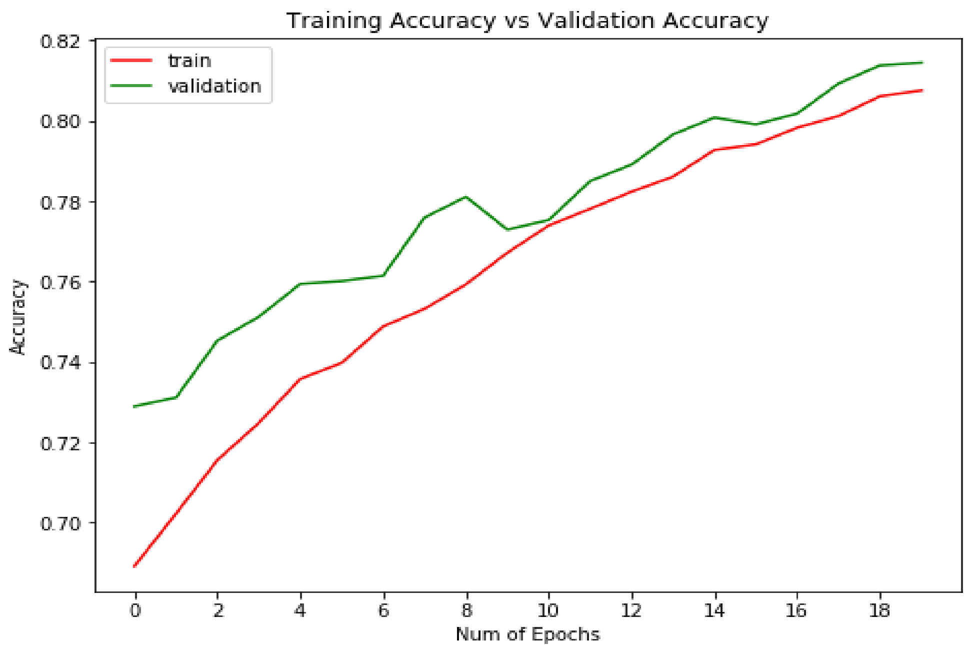


Fig. 1

Fig. 1 is the Accuracy graph of my final model. As we are increasing the number of epochs the train and validation accuracy is also increasing.

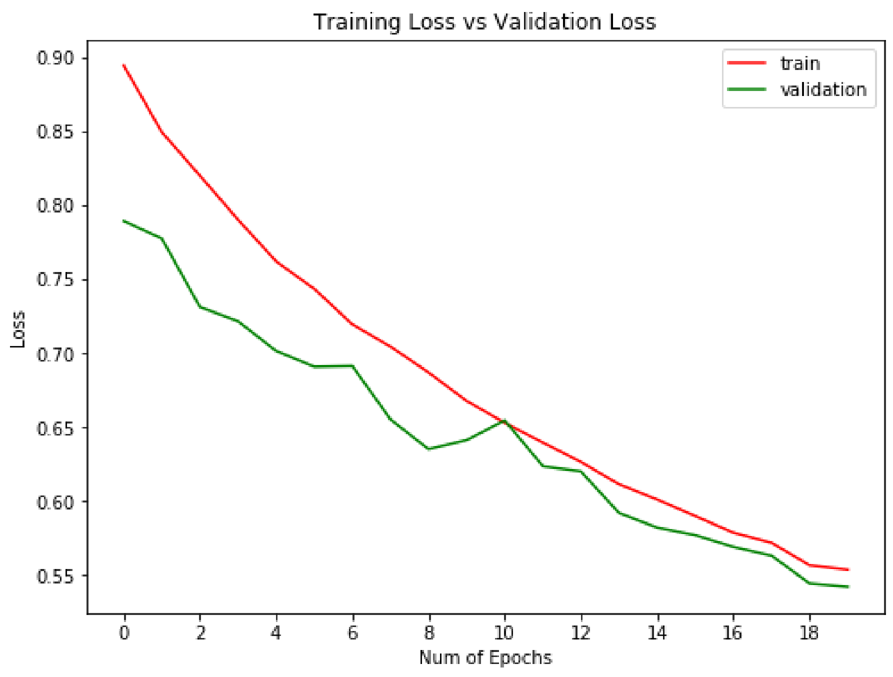


Fig. 2

Fig. 2 is the loss graph of my final model. As the number of epochs are increasing train and validation loss is decreasing.

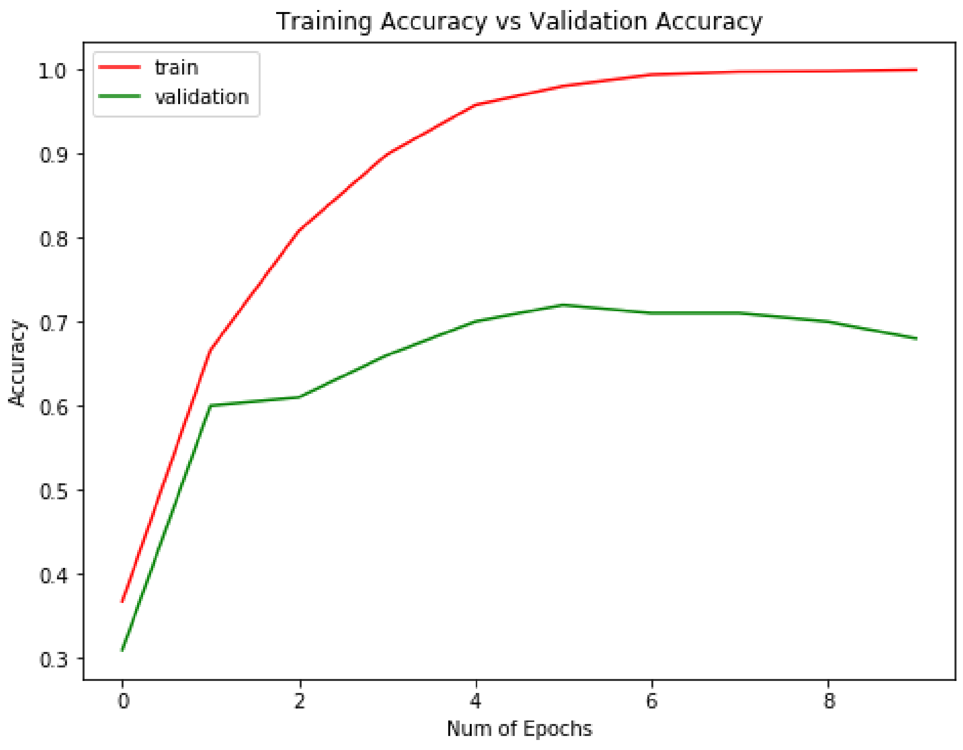


Fig. 3

Fig. 3 is the Accuracy graph of my inception-v3 model. As we are increasing the number of epochs the train and validation accuracy is also increasing. But because of lower training dataset the validation accuracy starts decreasing in the end as the model is overfitting.

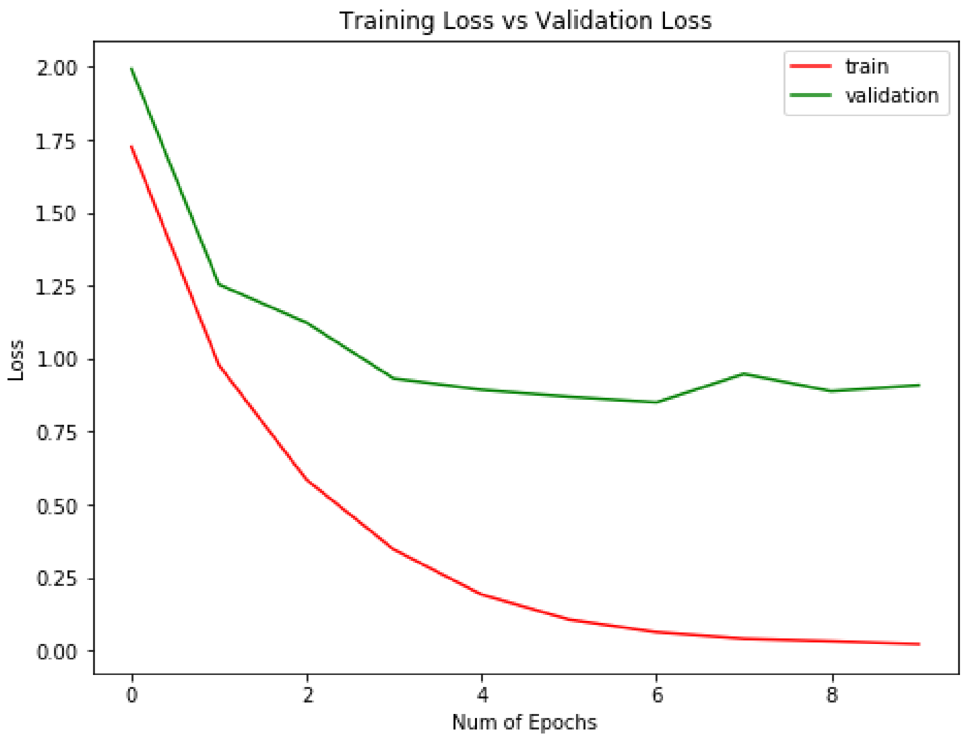


Fig. 4

Fig. 4 is the loss graph of my inception-v3 model. As the number of epochs are increasing train and validation loss are decreasing. In the last epochs the validation loss starts to increase again as the model tries to overfit.

Reflection

So for solving this problem of image classification on CIFAR-10 dataset we used Convolutional Neural network (a deep learning technique). We pre-processed the images from the CIFAR-10 dataset to make it eligible for the CNN architecture. We did the pre-processing by normalizing the inputs and providing each label a separate number (class) by one hot encoding. Then we defined an architecture which gave an accuracy of 70%. We needed to improve the result so we applied Data Augmentation on the training set which resulted in the final model with an accuracy of 81%. The project the totally interesting but the most difficult part of the project was the implementation of benchmark model on the CIFAR-10 dataset as the inputs needed to be resized and the fully-connected layer also needed to be redefined. Yes, the final model is up to my expectations and it can also be used to solve general image classification problems.

Improvement

Yes, the final model can be improved by increasing the number of epochs and we can also use the following methods of Algorithm Tuning:

1. Diagnostics.
2. Weight Initialization.
3. Learning Rate.
4. Activation Functions.
5. Network Topology.
6. Batches and Epochs.
7. Regularization.
8. Optimization and Loss.
9. Early Stopping.