Group Assignment 3 report

Ajay RR (IMT2017502) Ronak Doshi (IMT2017523) Ram S (IMT2017521)

Visual Recognition Course
International Institute of Information Technology Bangalore

Contents

1 Importance of Feature Extraction and Feature Learning

1.1 Feature Extraction

Feature Extraction is an important aspect when it comes to applying Machine Learning to solve image classification problems. Feature Extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and classification.
It is also inherently related to dimensionality reduction. Essentially it's all about reducing the amount of data required to represent each image (accurately for the task) while maintaining/increasing accuracy for any image task (such as classification etc.).

1.2 Feature Learning

Feature Learning is an inherent and important aspect of Deep Learning that, in retrospect, helps us identify the core features of an image that have been learned automatically by virtue of training the neural network.
By training the network through supplying the image data, the network itself learns how to recognize the important features from the image and extracts them to help with the task. This approach, as shown through results, has significantly increased the performance benchmarks for image related tasks.
This can be reasoned through the fact that humans might not always extract the right set of features for the given data set. In fact, we look for a general technique that we can apply to all images to extract the features. For some data sets, some of those features might be irrelevant while other features could be more relevant. By allowing the network to extract them itself, we allow for these feature extraction optimizations which can help increase performance.

1.3 Observations from the MNIST data set

1.4 Neural Network Classification

The basic network architecture that was first used consisted of just one convolutional layer. Such a network would capture the basic features. In order to capture more complex features, another convolutional layer was added. In both cases, the training data was split as 20% for validation. The two models are shown in Figure 1.

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 26, 26, 28)	280
max_pooling2d_2 (MaxPooling2	(None, 13, 13, 28)	0
flatten_2 (Flatten)	(None, 4732)	0
dense_3 (Dense)	(None, 128)	605824
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 10)	1290

Total params: 607,394
Trainable params: 607,394
Non-trainable params: 0

(a) One convolutional layer network

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 128)	1179776
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0

(b) Two convolutional layer network

Figure 1: Training evaluation

1.5 Hyper-parameter tuning

Hyper-parameter tuning is required to obtain the best performance in terms of balancing the bias and variance. A detailed explanation of each of the hyper-parameters and their implications is given in 2.3. The following hyper-parameters were tuned :

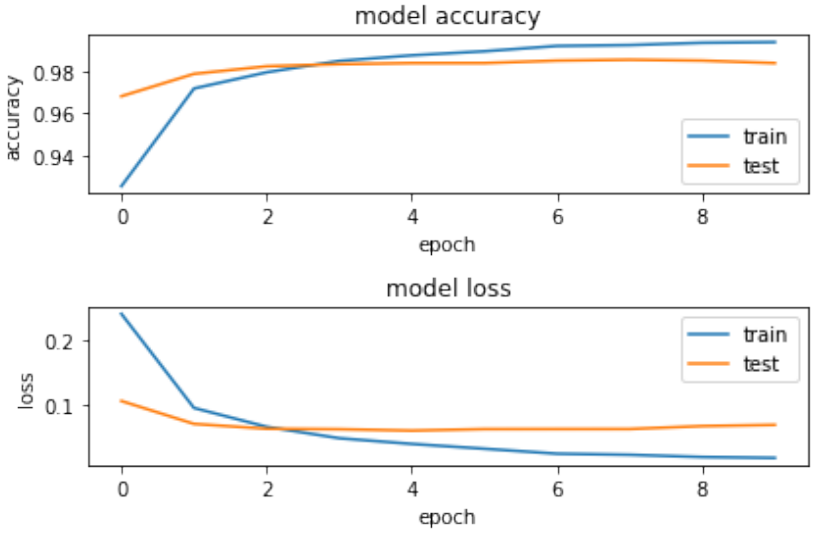
- **Activation function :** ReLU, Sigmoid and Tanh were the activation functions that were considered for tuning.
- **Optimizer :** The optimizer is the algorithm used by the model to update the weights. A disadvantage of SGD is that it cannot escape the saddle points. Thus, RMSProp and Adam were also considered.
- **Learning rate :** Choosing an appropriate learning rate is very important. If it is too high, we may be unable to converge to the minima and if it is too low, it can slow down the learning process.
- **Dropout :** The keep-probability of the Dropout layer is a hyper-parameter which could act as a regularizer to help us find the optimum bias-variance spot.

The hyper-parameter tuning was done using 'hyperas'. The tuned hyper-parameter values are :

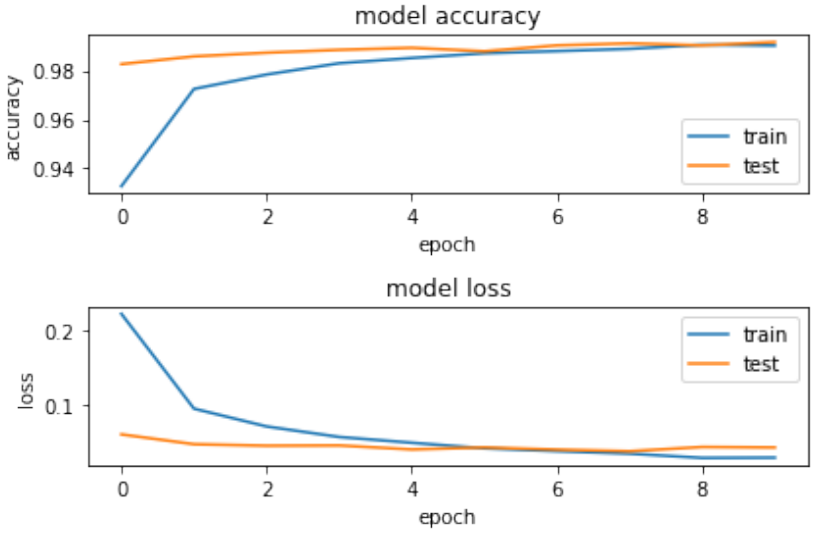
1. **Activation function** = ReLU
2. **Optimizer** = Adam
3. **Learning Rate** = 0.001
4. **Droupout probability** = 0.27 and 0.51

1.6 Results

An evaluation of the two networks is shown in Figure 6. The classification accuracy of the firt network is 98.479% and that of the second network is 99.119%.



(a) One convolutional layer network

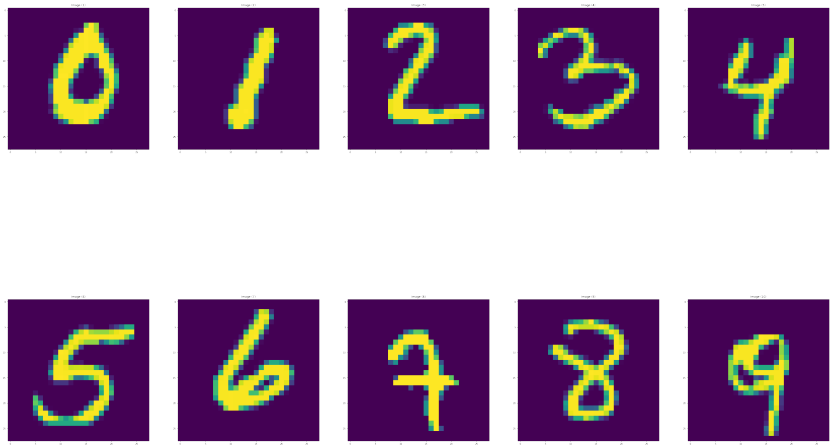


(b) Two convolutional layer network

Figure 2: Training evaluation

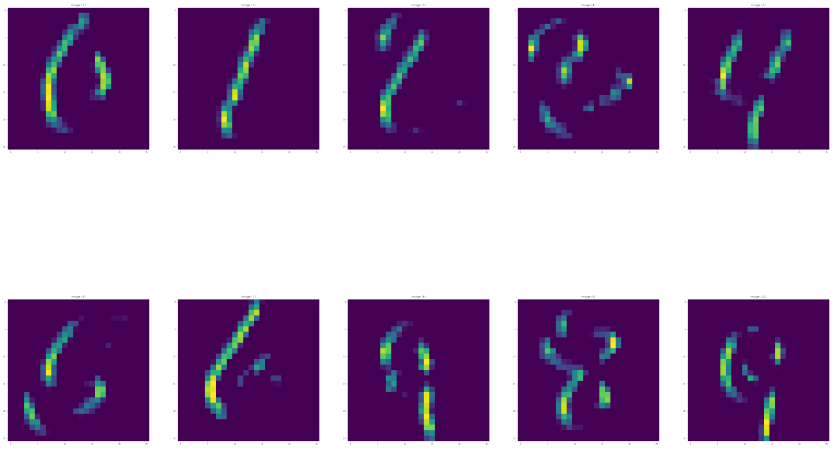
1.7 Activation Maps

By analysing the activation maps, we can get a sense of all the features that have been identified as important by our model as well as compare it to known methods of feature extraction. Shown below are the activation maps as we progress through the layers in the Neural Network.



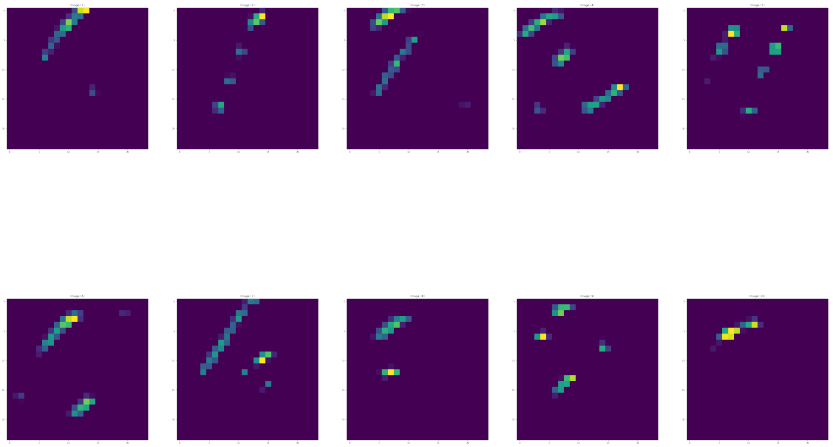
(a) Input MNIST Data for each digit

The above image data is passed through the first layer of the Neural Network, which is a Convolutional layer with kernel size 3x3, the following output is obtained as an activation map.



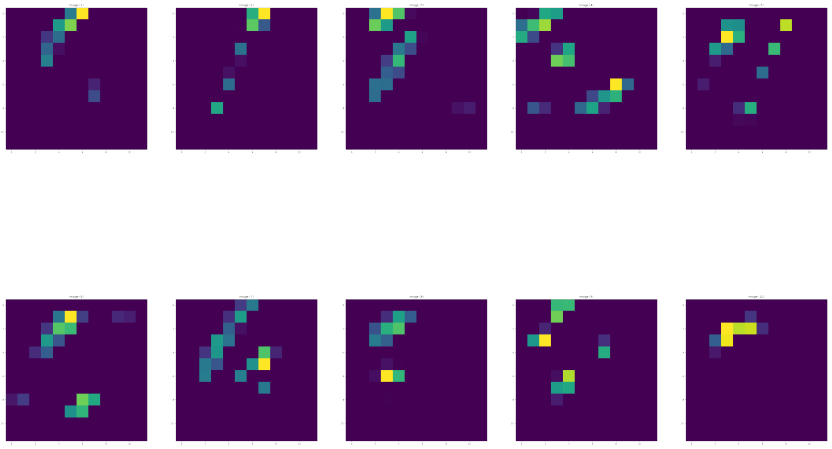
(a) Activation Map for Layer 1

The above output is passed through a second convolutional layer of the same kernel size and the output is shown as an activation map below. Now the output is passed through the max pooling layer and the results are as shown in



(a) Activation Map for Layer 2

Figure 6a.
It is these set of features that are fed to the fully connected layers for subsequent classification.



(a) Activation Map for Layer 3

Figure 6: Activation Maps

2 CIFAR10 Classification

2.1 About Dataset

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. The same can also be imported using keras backend in python. The classes that the CIFAR10 dataset contains is shown in Figure 7.

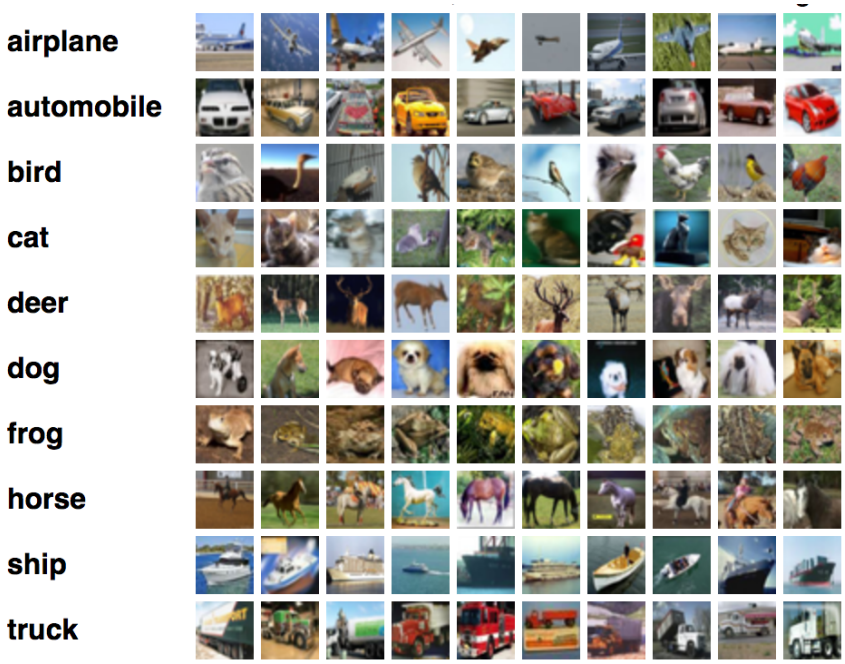


Figure 7: CIFAR10 Dataset

2.2 Convolution Neural Network

Previously we used computer vision approaches to solve the CIFAR10 classification problem. We used algorithms such as SIFT/SURF in order to get valuable keypoints and descriptors followed by bag of visual words (BoVW) or Vector of locally aggregated descriptor (VLAD) approach to generate a feature vector for an image.
After performing the above two approaches, we saw that VLAD showed significant improvement in the accuracy compared to BoVW. Stating results shown in Table 1, VLAD gave approximately 20% more accuracy than BoVW.

Table 1: Accuracy Table

Model	Accuracy
SURF + BoVW + SVM	25.5%
SURF + VLAD + SVM	44%

But using these ancient algorithms and approaches will not give accuracy better than what we got. Hence we need better approaches such as Deep Learning models in order to get good accuracy. A Deep learning approch will help in learning and extracting valuable features that will result in better classification
Therefore we are now using a convolution neural network in order to get better performance. For Building a convolution neural network we need to first set the hyper-parameters according to our model and dataset. Hence we need to fist explore and experiment with all sorts of hyper-parameters in-order to get the best set of hyper-parameter that will produce maximum accuracy in least amount of training time. This is called hyper-parameter tuning.

Here are the set of hyper-parameters that we can tune:

- Activation Function
- Batch Normalization
- Model Optimizer
- Momentium Learning Rates
- Number of Convolution and Polling layers
- Number of neurons per layer
- And many more ...

2.3 Experimentation

For tuning the hyper-parameters we need to understand them and experiment different set of parameters in order to know the best suited parameters because the value of these parameters varies with dataset and the problem trying to solve.
Hence in these section we try to experiment with as any hyper-parameters as possible. In order to understand them we created a simple CNN with 2 convolution layer, 2 pooling layers and fully connected component as shown in Figure 8. All the experiments were trained on this architecture using only 30 epochs (less epoch is used just for testing all the hyper-parameter).

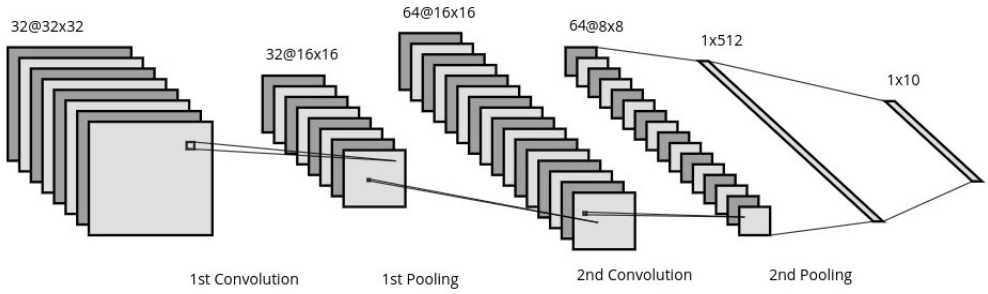
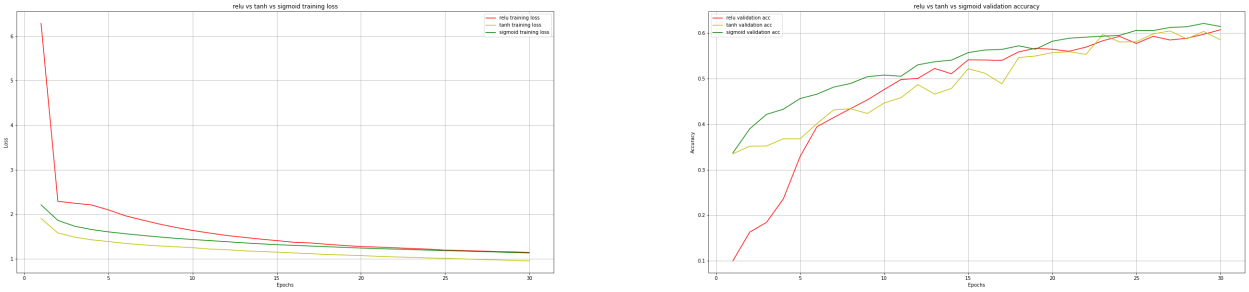


Figure 8: CNN structure

2.3.1 Activation Functions (ReLu, Tanh, Sigmoid)

It's a function that you use to get the output of node. Simply put, it calculates a "weighted sum" of its input, adds a bias and then decides whether it should be "fired" or not.
There are three commonly known activation functions called ReLu, Tanh and Sigmoid.
We trained the CNN thrice with different activation functions.



(a) Training Loss Comparison

(b) Validation Accuracy Comparison

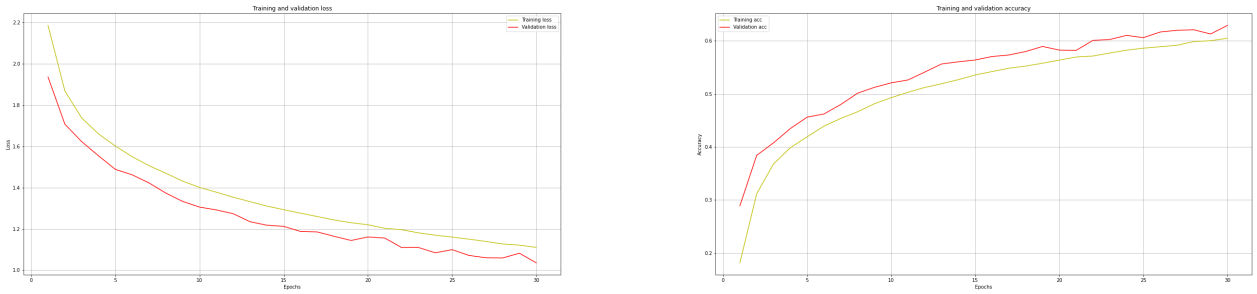
Figure 9: A figure with two subfigures

From the Figure 9a we can see that loss function for tanh and sigmoid dropped faster than relu but as the number of epochs increases the loss in tanh and sigmoid saturated with very less decrement whereas loss for relu continued to drop. The reason why loss for tanh and sigmoid was saturated after first few epochs is that their gradient started to vanish with epoch, due to which weights updated very slowly. Hence activation functions like tanh and sigmoid need

way more epochs to train compared to relu.
Hence it is recommended to use ReLu as the activation function.
Note: Time taken by for one epoch was same for all three functions.

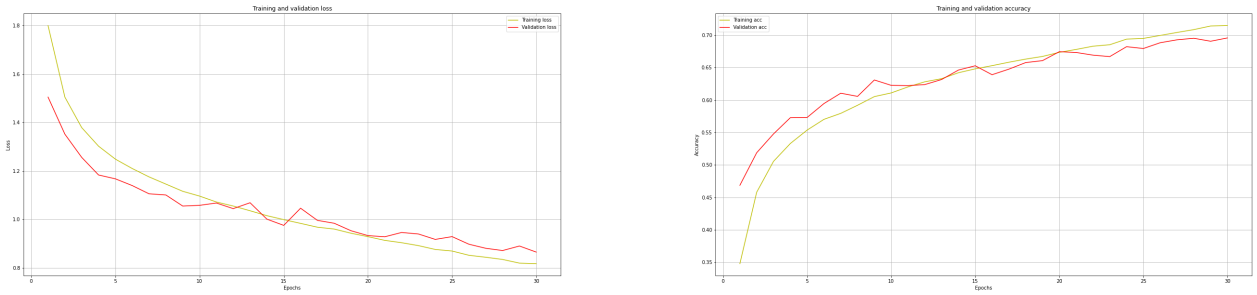
2.3.2 Batch Normalization

Batch normalization is used to stabilize and perhaps accelerate the learning process. It does so by applying a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.
In order to understand the effects of batch normalization we train the CNN model twice, once with batch normalization and once without batch normalization. Both the times the activation function used was sigmoid.



(a) Training and Validation Loss (b) Training and Validation Accuracy

Figure 10: Without Batch Normalization



(a) Training and Validation Loss (b) Training and Validation Accuracy

Figure 11: With Batch Normalization

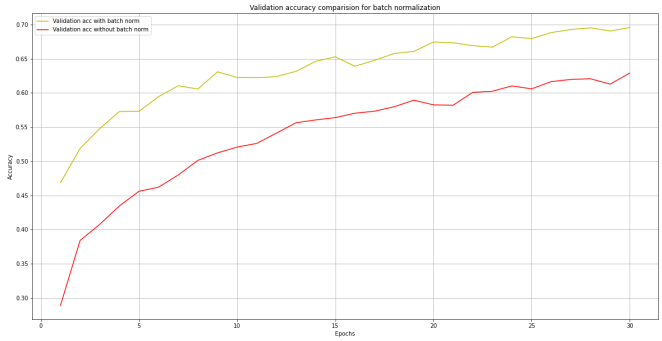


Figure 12: Validation Accuracy comparison with and without Batch normalization

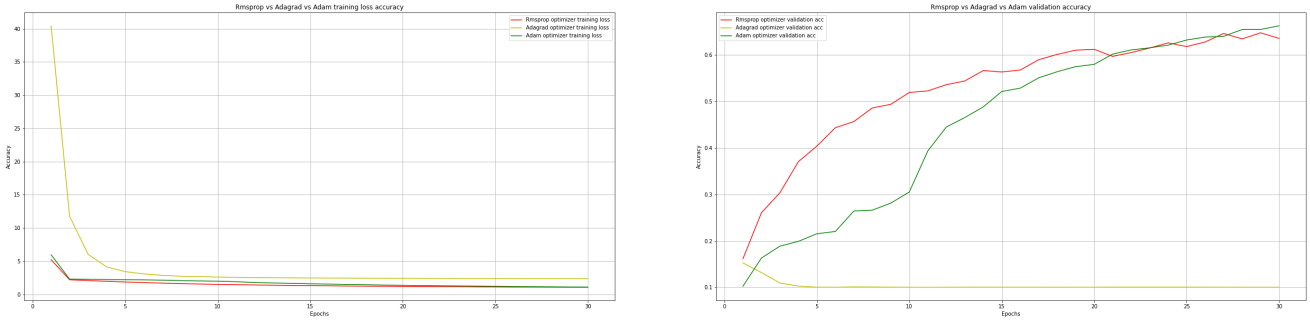
After training the CNN with and without batch normalization, the results were quite good. From the Figure 12 we can clearly validation accuracy for CNN with batch normalization was way always greater than the one without batch normalization. As batch normalization itself does not help in producing better accuracy but help in faster convergence of the back-prop algorithm, which will help in lowering the loss faster and hence helps in producing greater accuracy at same number of epochs. That is, the one with batch normalization will take less number of epochs compared to one without batch normalization.
As stated in the activation function section about the vanishing gradient, batch normalization helps in mitigate the issue by normalizing the input.
This can be seen Figure 10 (a) and Figure 11 (a), the loss in CNN with batch normalization is way lesser than the one without.
Hence it is better to use batch normalization for faster convergence. Note: Due to batch normalization, each epoch time in increased by 30s but this increase is nothing compared to the convergence time taken by one with no batch normalization.

2.3.3 Optimizers and Learning Rates

Optimization is a process of searching for parameters that minimize or maximize our functions. Hence in machine learning we use optimizers to update the weight parameters to minimize the loss function. Loss function acts as guides to the terrain telling optimizer if it is moving in the right direction to reach the bottom of the valley, the global minimum. There are four popular optimizers used for CNN:

- SGD
- RmsProp
- AdaGrad
- Adam

Which optimizer to use depends on your dataset, hence we experimented RmsProp, AdaGrad and Adam optimizers on the CNN. We trained the CNN thrice using differnet optimizers each time to understand it's performance for CIFAR10.



(a) Training Loss for different optimizer

(b) Validation Accuracy for different optimizer

Figure 13: Optimizer Comparison

From Figure 13 (a) we can see that loss for AdaGrad is greater in comaprision with RmsProp and Adam. And from Figure 13 (b) we can clearly stat that AdaGrad optimizer for our dataset is not a good fit. RmsProp and Adam can be used as optimizers, and seeing the graph it makes more sense to use RmaProp as the optimizer.
There is one more parameter in all the three optimizer is the momentum. Momentum is like a ball rolling downhill. The ball will gain momentum as it rolls down the hill. So in momentum, instead of using only the gradient of the current step to guide the search, momentum also accumulates the gradient of the past steps to determine the direction to go. Momentum helps accelerate gradients in the right direction.

2.4 CNN architecture

The architecture of a CNN is a key factor in determining its performance and efficiency. The way in which the layers are structured, which elements are used in each layer and how they are designed will often affect the speed and accuracy with which it can perform various tasks.
After getting a good idea about the hyper-parameters we start training the CNN with the better suited hyper-parameters. But before that we need to first decide the architecture of out CNN, i.e number of convolution layers, number of pooling layers, dropout layer, number of neurons in fully connected layer, etc.

2.4.1 First Architecture

We first used the same CNN architecture as showed in Figure 8, i.e. 2 convolution layers, 2 pooling layers and a fully connected layer, with the hyper-parameters values as follows:

- 100 epochs
- ReLU as Activation function
- With Batch Normalization
- RmsProp as the Optimizer
- Learning rate of 0.0001
- Decay of $1e^{-6}$

Dropout regularization technique is utilized to avoid overfitting.
Details about each of the layers in the architecture:

1. For the first convolution layer we used 32 filters with the filter size as 3×3 with zero padding so that the output size of the convoluted image is the same as the input image.
2. Followed by this comes the first pooling layer on pooling size 2×2 halvening the image dimensions
3. Followed by this comes second convolution layer we used 64 filters with the filter size as 3×3 with zero padding so that the output size of the convoluted image is the same as the input image.
4. Followed by this comes the second pooling layer same as the first one.
5. Output of this pooling layer is flattened and feeded to a fully connected neural network

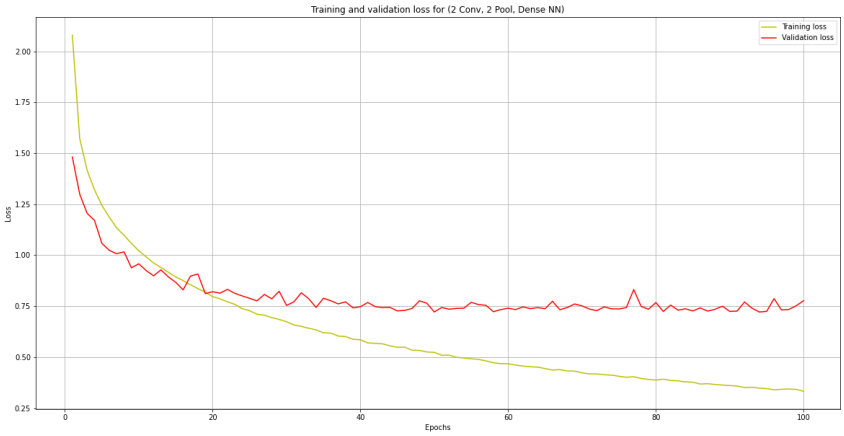


Figure 14: Training and validation loss for architecture 1

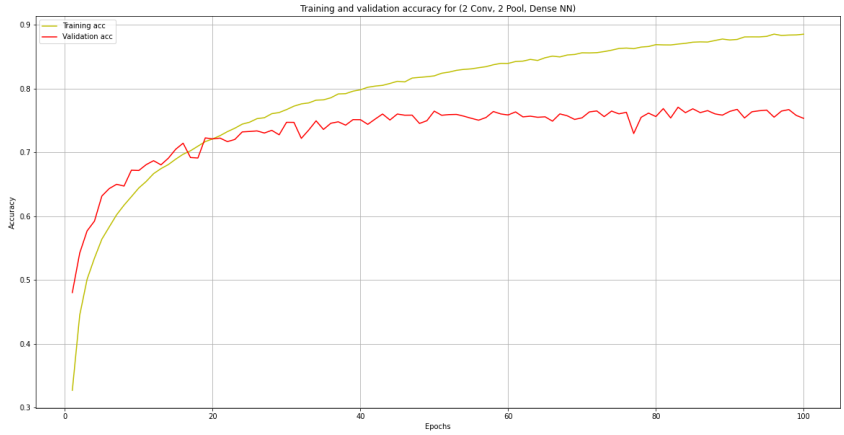


Figure 15: Training and validation accuracy for architecture 1

It can be seen from the Figure 14 that the training loss kept on decreasing but the validation loss started to saturate after 50th epoch.
We can conclude that the network might have started to overfit the data due to which the training loss was decreasing continuously while validation loss was saturating. We were able to achieve maximum validation accuracy of **77.11%** at 84th epoch with corresponding training accuracy of **87.12%**.

Hence we concluded to make our CNN network even more dense by adding more convolution layers to it to make our network a bit more complex.

2.4.2 Second Architecture

The second architecture that we thought of was the similar to the first one but with 2 more convolution layers as shown in the Figure 16

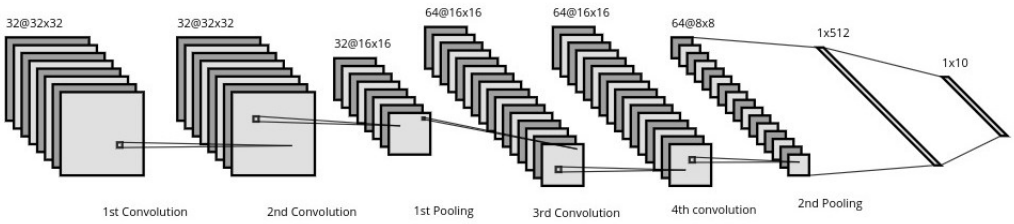


Figure 16: CNN structure

We used 4 convolution layers, 2 pooling layers and a fully connected layer, with all the hyper-parameters values same as the first architecture. Dropout regularization technique is utilized to avoid overfitting.
Details about each of the layers in the architecture:

1. For the first convolution layer we used 32 filters with the filter size as 3×3 with zero padding so that the output size of the convoluted image is the same as the input image.
2. Followed by this comes second convolution layer same as the first
3. Followed by this comes the first pooling layer on pooling size 2×2 halvening the image dimensions
4. Followed by this comes third convolution layer we used 64 filters with the filter size as 3×3 with zero padding so that the output size of the convoluted image is the same as the input image.
5. Followed by this comes second convolution layer same as the third
6. Followed by this comes the second pooling layer same as the first one.
7. Output of this pooling layer is flattened and feeded to a fully connected neural network

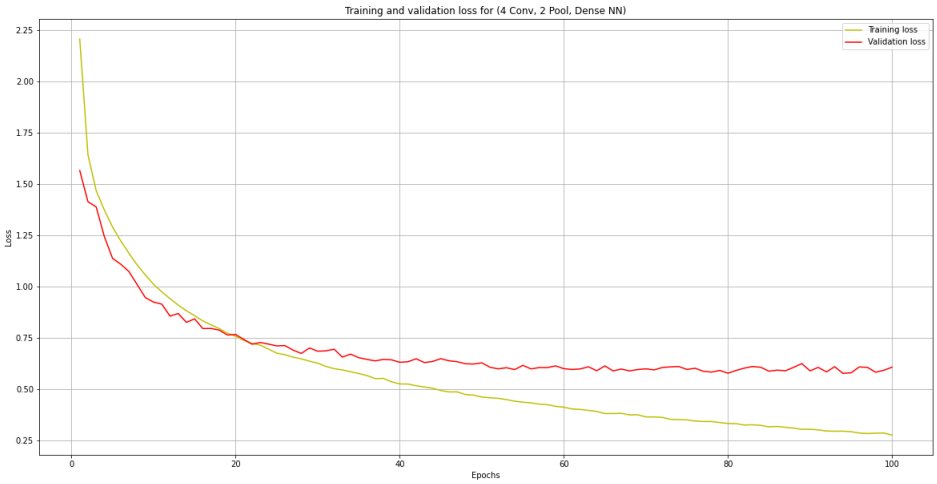


Figure 17: Training and validation loss for architecture 2

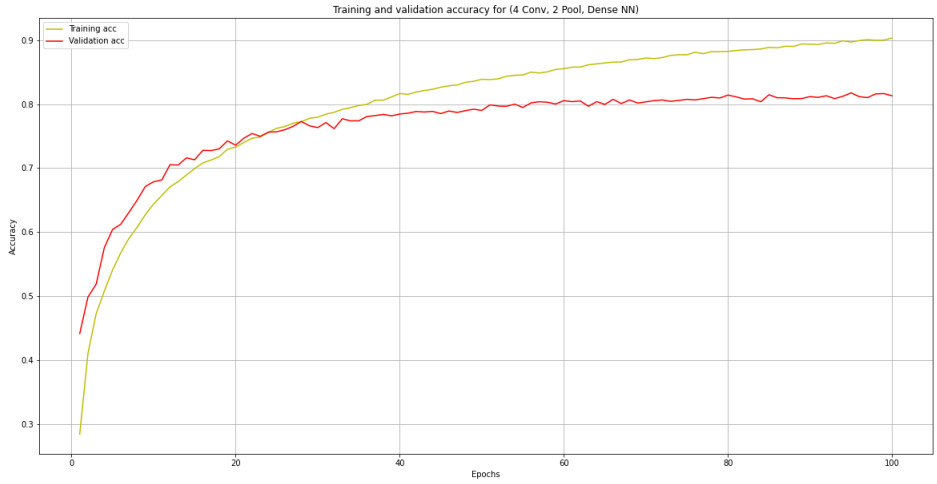


Figure 18: Training and validation accuracy for architecture 2

We can see from the Figure 17 and Figure 18 that unlike the first architecture, this architecture did not saturate. The validation loss keeps on decreasing, maybe not as fast as the training loss but it still kept on decreasing due to which validation accuracy kept on increasing.
We were able to achieve maximum validation accuracy of **81.66%** at 99th epoch with corresponding training accuracy of **90.02%**, which a significant improvement from first architecture.

2.5 Problems Faced

One of the major problem faced was the lack of computation power. Training the neural network even with 30 epochs was very computationally intensive for a personal computer or google colab (considering the size of the dataset). Processing took lot of hours in one stretch. With RAM limitations and low processing power it was very difficult to gather results. For get results for each experiment we needed to re-train the entire network which was very computationally heavy. Hence we performed as many experiments as possible.

2.6 Conclusion

From this we conclude that hyper-parameter tuning is one of the most important part of neural network. There are no default set of hyper-parameters that will work for all the networks. Values for hyper-parameters varies with the dataset and type of problem. Experimentation is a major way for understanding your dataset and choosing appropriate set of hyper-parameters.

After performing all possible experimentation (keeping in mind the lack of computational power) we concluded that the second architecture will be our recommended architecture.
We believe (if less limitations on power) we can get better performance by add few more convolution layers with different set of hyper-parameters.

References

[1] https://keras.io/

[2] CNN Drawing Tool: http://alexlenail.me/NN-SVG/LeNet.html and https://vectr.com/

[3] https://keras.io/examples/cifar10_cnn/

3 Object Recognition

Object recognition is a computer vision technique for identifying objects in images or videos. It is a key technology behind driverless cars, enabling them to recognize a stop sign or to distinguish a pedestrian from a lamppost. It is also useful in a variety of applications such as disease identification in bio-imaging, industrial inspection, and robotic vision. Object recognition can be achieved using machine learning as well as deep learning techniques. As explained and evaluated in 1, the deep learning method learns the features and is more effective and it has thus been adapted to perform object recognition. A simple pipeline of an object recognition system is shown in Figure 19. It consists of a Convolutional Neural Network which learns the features followed by fully connected layers for classification. The final output is the probability for each class and the highest can be chosen as the prediction (Top 1). In addtition, a classifier such as a Support Vector Machine can be used to classify the image using the features that are learned by the network.

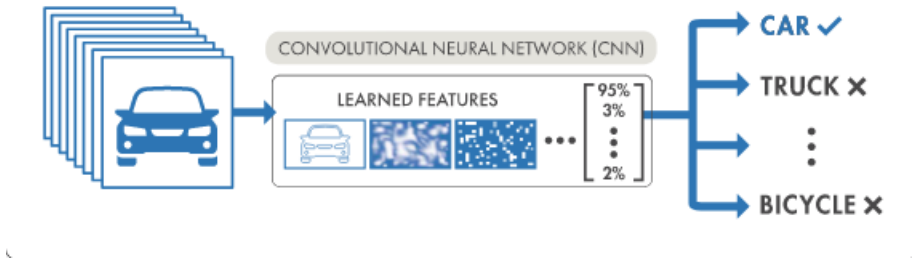


Figure 19: An object recognition deep learning system
Source: mathworks

There are generally two approaches to perform object recognition :

- **Training a model from scratch :** This requires a huge amount of training data and computation power.
- **Using a pre-trained deep learning network :** A deep learning network that has been trained on a huge dataset can be used in two ways - transfer learning or as a feature extractor. In this assignment, we use a pre-trained network as a feature extractor.

3.1 Dataset

The Caltech 101 dataset which is available at [1] was used for this task. The dataset contains images of objects belonging to 102 categories. A few categories of the dataset are shown in Figure 20. An object recognition dataset contains one category per image.

3.2 Preprocessing

- The number of images per category was uneven. Thus, 31 images were chosen at random for each object category to balance the dataset for training and testing. The number 31 was chosen because it is the minimum number of images in a category.
- The images were resized to the required input sizes for the respective networks that were used.

3.3 Network as a feature extractor

The pre-trained AlexNet, which has been trained on the ImageNet dataset was used as a feature extractor. The first few layers extract lower level features. These "primitive" features are then processed by deeper network layers, which combine the early features to form higher level image features. These higher level features are better suited for recognition tasks because they combine all the primitive features into a richer image representation. Thus, the layer right before the classification layer was used for feature extraction. A SVM was then used to classify the images using the extracted features. This method was implemented in MATLAB since keras and tensorflow do not readily contain the pre-trained AlexNet.

3.4 Additional experimentation

The above mentioned method was implemented using different well known CNNs such as VGG19, Resnet and Mobilenetv2. The results are shown in 3.5.

(a) Airplane

(b) Octopus

(c) Sunflower

(d) Watch

Figure 20: Few categories of the dataset

3.5 Results

The results of both the methods are shown in Table 2. It can be observed that the order of performance of the different networks is roughly similar to their order of performance on the ImageNet dataset. This is obvious, since a better performance on the ImageNet dataset would mean better feature extraction capabilities. Thus, their performance is better on this dataset as well.

Feature Extraction network	**Accuracy**
AlexNet	83.67%
VGG19	84.96%
Resnet18	85.51%
Mobilenetv2	88.56%
Resnet50	89.12%
Resnet101	93.03%

Table 2: results

These results show that a sufficiently good performance can be obtained by using a pre-trained network as a feature extractor. It is also computationally less expensive and does not require a huge dataset.

References

[1] Caltech 101 dataset - http://www.vision.caltech.edu/Image_Datasets/Caltech101/