Introduction to Machine Learning: Project Report
Object Recognition on CIFAR-10

Akshat Praneet, Niwas Chadha, Tanmay Betrabet

Abstract: We attempt to achieve and beat the state of the art results on the CIFAR-10 dataset. We begin with employing simple classifiers and gradually scale up to experimenting with Convolutional Neural Networks. While we do not achieve the SOTA, our experiments lead us to 85% accuracy based on models completely designed by us.

1 Introduction

Humans have always had it easy with classifying images to objects. They are able to perceive shapes, edges, colours, etc in images and are able to subconsciously classify them into the encompassing object in no time. Classifying objects from a picture has not always been easy for computers. Most of the times, images show a two-dimensional representation of usually three-dimensional objects. Computers also find it difficult to pin point the objects in question when there is variation in terms of colour, brightness, angles, etc. With the recent advancements in computing and availability of large scale data sets, object recognition by computers has an ever growing interest.

With the advent of the digital age, image recognition and computer vision tasks are at every instance. Self-driving cars use object detection to drive smoothly with minimal risk to drivers and those on the streets. Autonomous drones are equipped with image recognition to facilitate easy GIS surveying, monitoring of forests, and for destructive purposes.

Large corporations like Google, Tesla, Amazon are leveraging these advances to bring more commercial uses. Google has been using image recognition to bring Google Lens and automatic translation from one's smartphone camera. Amazon is using this technology to make their warehouse logistics more streamlined. Even social media sites like Facebook and Twitter are leveraging this technology to identify images and objects in images to create accessibility descriptions of them.

Our project aims to understand this process better and try to build our own machine capable of identifying 10 object classes in the CIFAR-10 dataset. Our goal was to beat the state of the art model existing.

2 Data Set

2.1 CIFAR-10

CIFAR-10 (Canadian Institute For Advanced Research) a publicly available data set of collection of small images collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. It consists of 60,000, 32×32 coloured images belonging to 10 object classes. The classes are namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

There are 6000 images of each class and the data is divided into 50,000 training and 10,000 test images. The training batch is divided such that each class has exactly 5000 images.

The advantage of the low resolution images is that it allows us to try a variety of classifiers and tweak them to improve results.

A sample of CIFAR-10 images is as below:

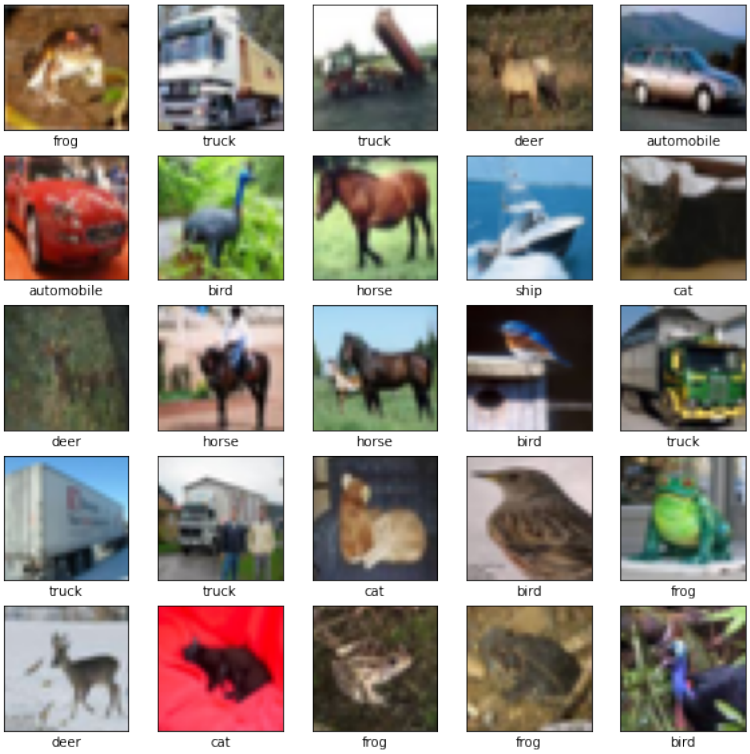


Figure 1: Sample Images of CIFAR-10 Data Set

2.2 Data Pre-Processing

Initially, the only pre-processing employed was scaling down the pixel values of training and testing images by a factor of 255, so each pixel holds values between 0 and 1.

We did not want to use any other preprocessing methods such as changing colour to greyscale as colours are an important attribute to these images. For example, a truck might have more grey colour than birds which can be more colourful. We also did not use PCA or ZCA scaling on the images since we believe it could lead to loss in features for the images.

However, in experiments on our baseline CNN, we augmented the training data by flipping, changing height and width, and zooming in on some images randomly. This would disturb the images slightly without compromising on the features that we expect the machine to detect before classification. The results improved our validation and test accuracy by 2-3% across the board for all our models.

3 Survey of Approaches

Models based on supervised or semi-supervised learning are effective approaches while dealing with image classification. Some of the common ones are briefly described below:

3.1 Supervised Learning

Given that CIFAR-10 is a labelled dataset with 60,000 labelled images, this approach is advisable. Methods such as k-Nearest neighbour (k-NN) classifier, random forests and Support Vector Machines(SVM) are simple and do not require heavy computational power to train.

The k-NN is the simplest of these which involves a single pass of all training patterns once to be trained. Random Forests yield interpretative results which make them desirable in any classification task. However, handling complex input vectors, such as a large vector of RGB pixels of an image, in the absence of meaningful 'features', these classifiers are unlikely to be effective. We still trained such models to act as a baseline.

More complex mechanisms such as Neural Networks are shown to have much better accuracy with image classification tasks and that is primarily what we tried to use to achieve SOTA comparable results. These are usually slow and computationally expensive to train given the vast number of model parameters. However, given their superior scaling properties, these are very desirable for such tasks.

3.2 Semi-Supervised Learning

By using a small sample of labelled data, a model is trained much like in supervised learning. This is then followed by the machine classifying the unlabelled patterns and learning through unsupervised mechanisms such as k-Means Clustering or Gaussian Mixture Models. These would be cost effective in the absence of readily available labelled data however we did not pursue these for the purposes of the current project.

4 Our Experiments With Code (Part 1): Baseline Models

While there is a plethora of approaches available to classify images, we tried to see if we could get a classifier using standard classifiers, keeping in mind the principle of Occam' Razor.

4.1 Random Forest Classifier

With the default parameters of 100 decision trees trained with bootstrapping of samples from the training set, we were able to achieve an accuracy of 46.33%.
While the model wasn't particularly over-predicting any particular class, the accuracy was fairly low. We hypothesise that this can be attributed to the fact that Random Forest being an entropy classifier, it is easier to randomly predict an image belonging to a certain class than not.

4.2 k-Nearest Neighbours Classifier

We also trained and tested a standard k-Nearest Neighbours classifier with the train-test split in the CIFAR-10 dataset. However, the results were less satisfactory compared to the Random Forest Classifier with an accuracy of close to 34%.

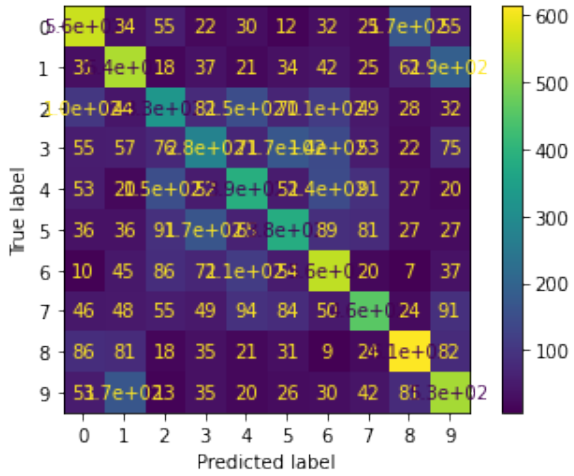


Figure 2: Random Forest Confusion Matrix

The low accuracy of the classifier could be attributed to the nature of the images in the training set being broadly either an animal or a vehicle. The images with animals tend to have a natural outdoors background with green grass or trees whereas the vehicles have more of a greyscale coloured background. The k-NN classifier might struggle to distinguish within the animal classes due to an issue like this.

Interestingly, the model did not take too long to train however the time to predict test images was quite time taking.

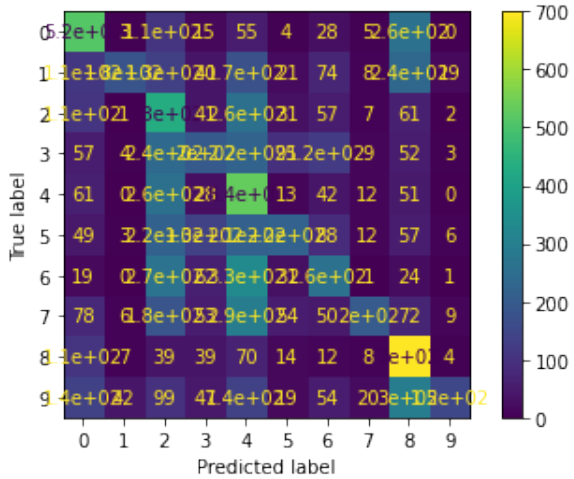


Figure 3: k-NN Confusion Matrix

4.3 Support Vector Machine Classifier

We tried to train an SVM classifier on the training data on the linear setting as this too was a standard classifier used for image recognition. The training process was immensely time taking therefore we restricted ourselves to a maximum of 500 iterations. The best accuracy on the test data which we could get from this classifier was an abysmal 35%.

We hypothesise that the low accuracy can be attributed possibly to the functioning of the classifier where it tried to linearly partition the data which was quite difficult to do given the limited itera-

tions.

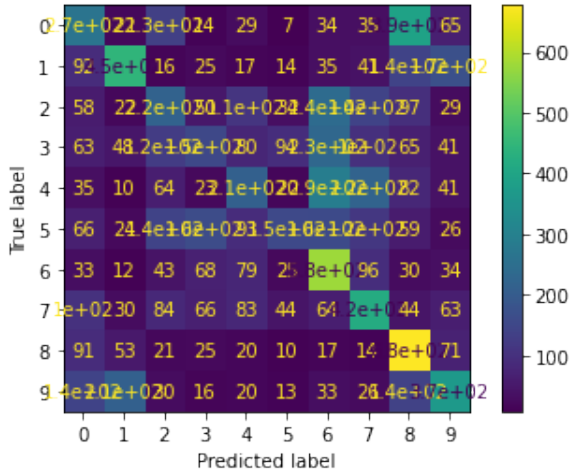


Figure 4: SVM Confusion Matrix

5 Our Experiments With Code (Part 2): Convolutional Neural Network

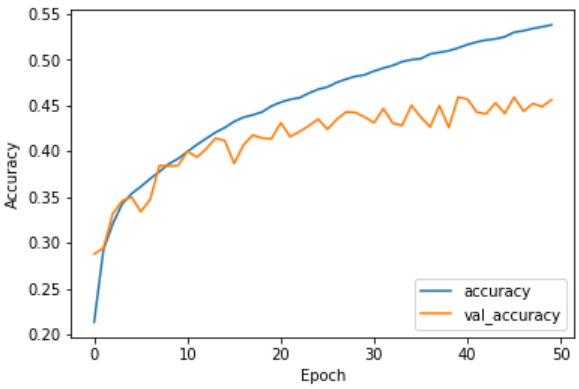
Since the simple approaches did not yield the accuracy levels we were hoping for, our next approach was to train Neural Networks and gauge their accuracy. Our approach was to first test the basic models and then incrementally add complexities in an effort to increase accuracy.

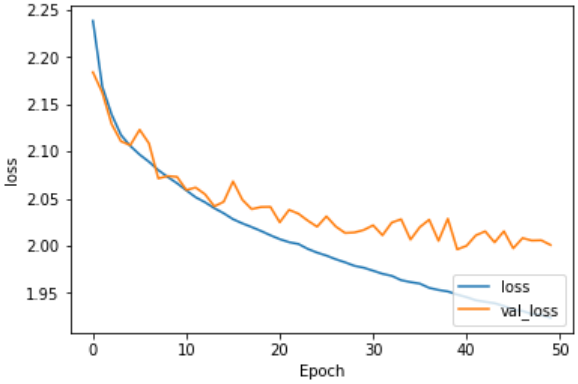
5.1 Multilayerd Neural Network

Our first approach to this problem was to train a Multi-Layered Neural Network. This simple model contained four Hidden layers of 1024, 256, 64 and 32 neurons respectively. This model had no other layers except the ones mentioned. Due to a large number of data points and the subsequent need of multiple layers with large number of neurons, the number of parameters exceeded 3 million.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_3 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_14 (Dense)             (None, 1024)              3146752
_________________________________________________________________
dense_15 (Dense)             (None, 256)               262400
_________________________________________________________________
dense_16 (Dense)             (None, 64)                16448
_________________________________________________________________
dense_17 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_18 (Dense)             (None, 10)                330
=================================================================
Total params: 3,428,010
Trainable params: 3,428,010
Non-trainable params: 0
_________________________________________________________________
```

This led to over fitting of data with training accuracy of approximately 53% and validation and testing accuracy less than 45%.



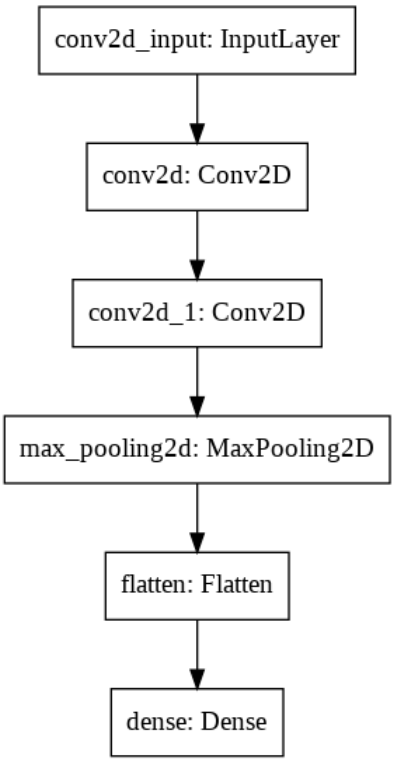


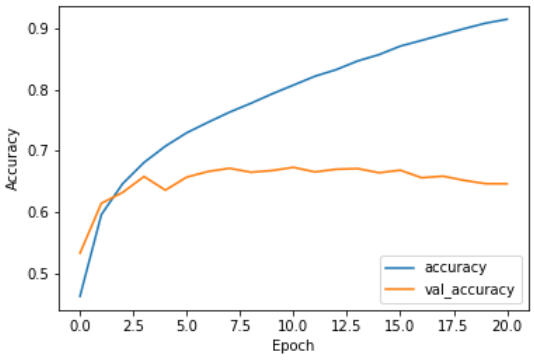
5.2 Simple CNN

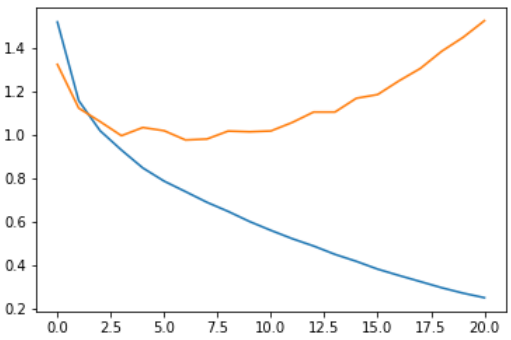
The first model of the CNN that we built had a simple architecture involving 2 convolutional layers, followed by a pooling layer. This was flattened and connected to a dense final layer. Learning occurred using the Adam optimiser with batches of 128 images at a time with 20% of the training images reserved for validation.

The model yielded an accuracy level of 63.37% on the testing data which was an improvement on the initial simple models. However, the model's training and validation accuracy plot over epochs, suggests that it begins to overfit the training data beyond a certain point. The model's generalisability could be improved upon without significantly altering its initial architecture.

We introduced data augmentation methods to some of the images before training and found that the testing accuracy increased slightly to 65.32%.





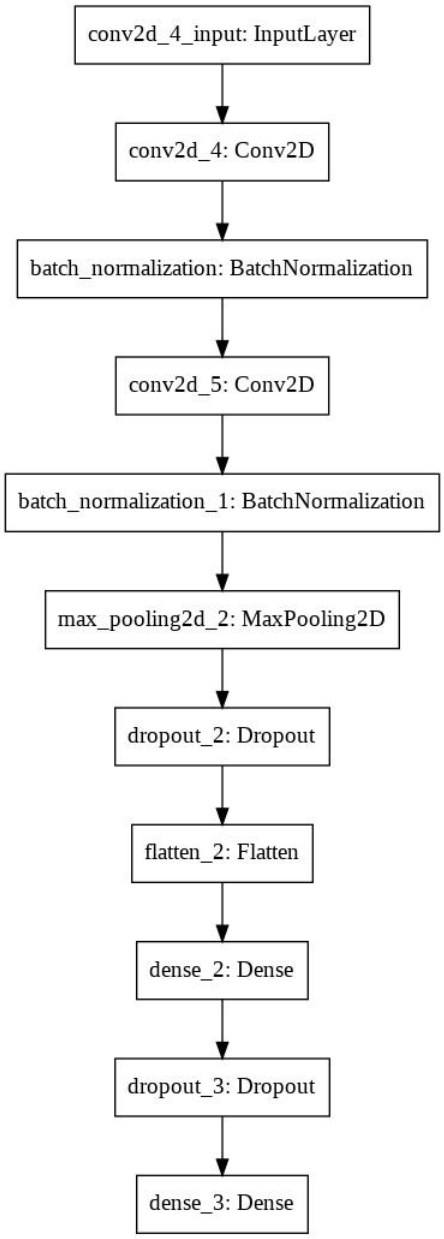


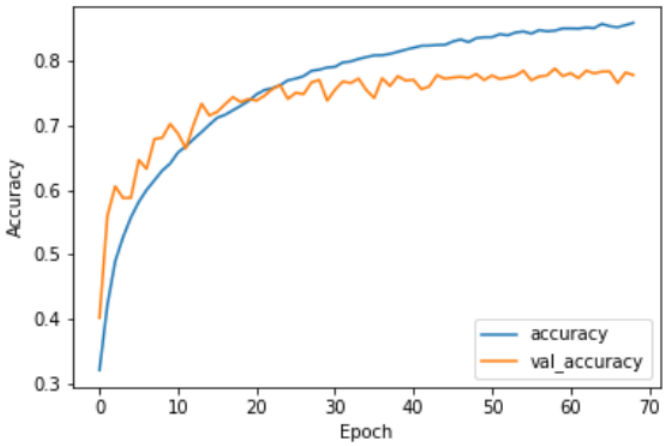
5.3 Improvements To Simple CNN: 'Not-So-Simple' CNN

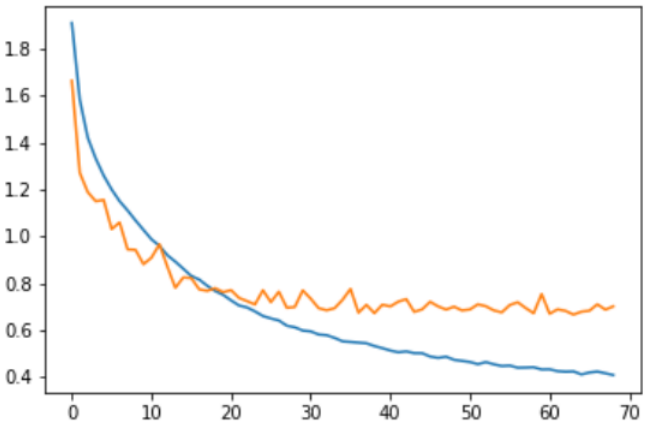
To further tackle the problem of overfitting, we introduced batch normalisation layers after each convolutional layer along with a dropout layer before flattening. This lowered training accuracy but increased validation accuracy, suggesting that the problem of overfitting had somewhat been mitigated. The machine now gave a validation accuracy of 71.42% and testing accuracy of 70.8%.

The third measure we introduced was adding a hidden dense layer before the final output layer along with a dropout layer. The dropout layers ensured that some attributes got dropped before moving on to the next layer so as to prevent overfitting.

Upon lending properties of an MLP to this final section, we saw improvements with the validation accuracy at 77.02% and testing accuracy up to 76.72%. By integrating all these improvements, the model definitely appeared to be on the right track.



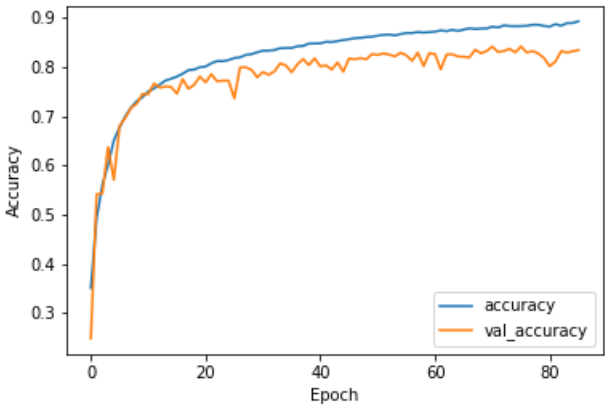


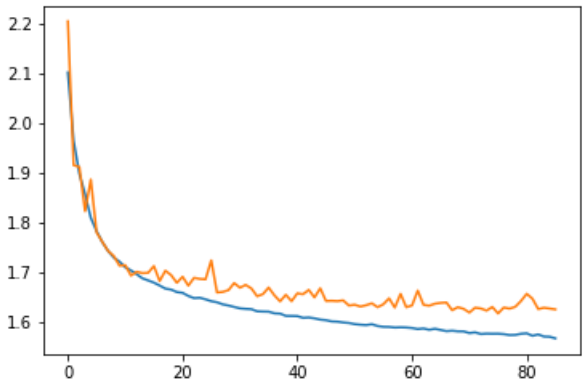


5.4 Approaching Final Model: Evolved CNN

We felt that the current model was maxed out in terms of optimising and would require additional complexity to improve its accuracy significantly.

We proceeded to add more convolutional layers followed by batch normalisation, coupled with pooling and dropouts in the same fashion as before. The results were encouraging and we improved our validation accuracy to 83.51% and testing accuracy to 83.1% without the hidden layer at the end. The model for our Evolved CNN is given in the Appendix.



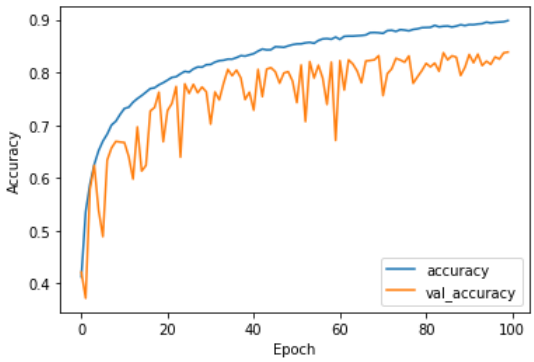


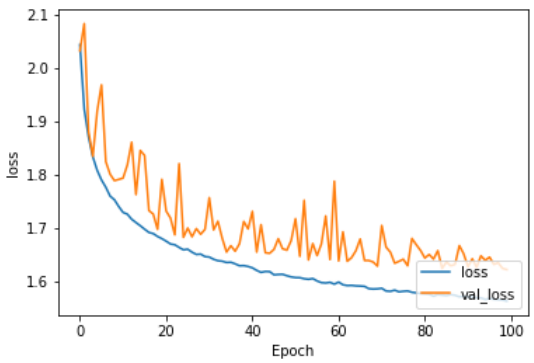
6 Our Experiments With Code (Part 3): VGG16 and VGG19

Our next approach was to look up already published models which have achieved state-of-the-art accuracy on image classifications

6.1 VGG16

One of the models we researched was VGG16, a convolutional neural network model proposed by K. Simonyan and A. Zisserman. VGG16 achieved 92.7%, top-5 image accuracy on ImageNet data set. It is to be noted that VGG16 trained on a much bigger data set 14 million+ images of 256 $\times$ 256 RGB images, with more than 1000 classes. We followed the architecture of the VGG16 network but scaled it down to adapt it to our model.
We decreased the number of trainable parameters by having more pooling layers to reduce over fitting while simultaneously increasing feature detectors that gave validation accuracy of 83% while the test accuracy was at 82.7%. The model for our adaptation of VGG16 is given in the Appendix.





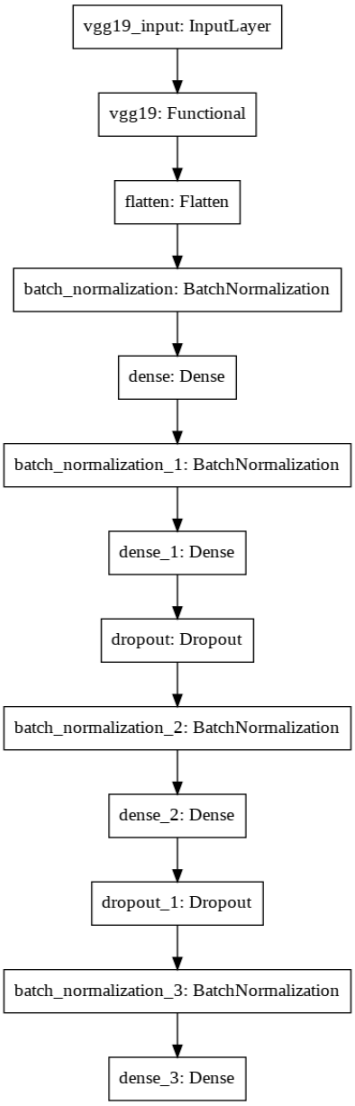
6.2 Transfer Learning using Pre-Trained VGG19

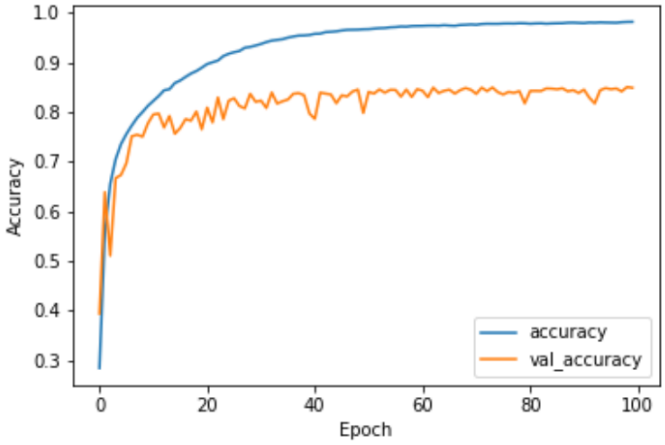
As a part of project, we decided to experiment using pre-trained models and build upon them. Using pre-trained models allows us to save on time and computational resources. We chose the VGG19 model by Simonyan and Zeisserman, as it was an advancement from the VGG16 model which we tried to emulate. The VGG19 model has 16 convolutional layers and 3 fully connected layers. It was trained on the ImageNet data set and achieved a top-5 accuracy of 90%.

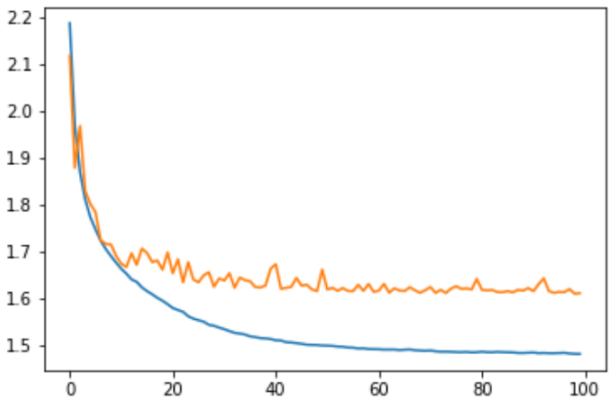
While importing the model, we do not include the 3 fully connected layers and changing the input shape it takes to suit our 32×32 sized inputs. More so, we build on it by adding a flattening layer right after the images go through the VGG19 model. We also add batch normalisation and dropout layers in the following dense layers of 256, 128, and 64 units. While the dense layers before the final output layer use a relu activation, the output layer utilises a SoftMax activation. We use the Stochastic Gradient Descent optimiser after a lot of trial-and-error with other optimisers.

Upon this stepping-on-the-shoulders-of-giants model, our machine reaches an 85.08% validation accuracy and 84.9% testing accuracy.

Due to the paucity of time and limitations in computing resources this was the maximum accuracy which we could reach.







7 Results

7.1 Summary of Results

The results achieved on the baseline model is as follows:

Model	Test Accuracy (in %)
k-Nearest Neighbour	34
Support Vector Machine	35
Random Forest	46.33

The results obtained on our progressive developments on CNN is as follows:

Model	Validation Accuracy (in %)	Test Accuracy (in %)
Multilayered Neural Network	53	45
Simple CNN	65.32	64.8
'Not-So-Simple' CNN-1	71.42	70.8
'Not-So-Simple' CNN-2	77.02	76.72
Evolved CNN	83.51	83.1
VGG16 Based CNN	83	82.7
VGG19 Transfered CNN	85.08	84.9

The state-of-the-art on the CIFAR-10 data set has been achieved by the model LaNet. It achieved 99.03% accuracy on the CIFAR-10 data. Other SOTA models have been the GPipe (99.0% accuracy), DenseNet(96.54%).

While our models do not reach the SOTA level, we believe if we run the model for suitable number of epochs with plenty compute resources available, we could have approached them.

8 Discussion

While working on the goal to increase accuracy we made some interesting observations and hypothesis on why some things happened.

Low Accuracy in Base Models

One of the reasons why our base models (k-NN, Random Forests and SVM classifier) failed was because they relied on unprocessed values of data points. Our data points , 32×32, RGB image when flattened becomes a vector of length 3072. These models relied on the values encapsulated in the vector rather than the values relative to its neighbour. In other words, these models failed to take advantage of values as a part of the local.

One easy way to understand is, by considering an image of a frog in grass and a deer in forest. Both these images would have high values for the green layer and relatively lower values red and blue. As such when we run k-NN model, the frobenius norm between the two images is very less. Now if we expand to include all data points with more classes, it is not convoluted to realise that the model will face this problem on a larger scale, and thus our models start evaluating on the basis of just colour and no other feature.

Does Data Normalisation Help?

Training our models on both unprocessed and processed data, we see a stark difference in the performance of our models. After standardising our inputs to the range [0,1], we see remarkable efficiency in training of the model which can be ascribed to easier handling of smaller inputs. After realising this we added Batch Normalisation layers before any activation layers to scale down our inputs for faster processing.

Impact of Data Augmentation

In an attempt to increase accuracy, we implemented data augmentation methods such as flipping, changing height and width, and zooming in on some images randomly. As mentioned above this boosted accuracy by 2-3%. This forced our model to recognise the same features in different orientations and sizes and thus widening the model's 'understanding' of the features that comprise a particular image.

The Appropriate Optimiser

Since this was a group endeavour, many models were generated independent of each other, by different members. As such, one noteworthy point we realised was the importance of our optimiser used for training process. Since different models were parallely being trained, different models gave similar results using different optimiser. Yet if the same model were again trained using a different optimiser we noticed a significant drop in learning speed and accuracy. As such our models were developed that seem to optimise the particular optimiser the best.

9 Conclusion

Over the course of this project, we have developed a deeper understanding of the methods of image recognition which has become an increasingly important tool in the modern world with vast consumer applications.

While the SOTA levels remained well beyond our reach it was interesting to note the incremental development starting from the simplest of classifiers to the much more advanced, convolutional neural networks. The biggest learning however not to be undermined has been learning how to deal with bugs and patience as machines train.

Given adequate compute power and further learning of machine learning our group believes we can definitely achieve the state of the art in classification of CIFAR-10 data.

10 References

https://www.cs.toronto.edu/ kriz/cifar.html
Wang, Linnan, et al. *"Sample-efficient neural architecture search by learning action space."* arXiv preprint arXiv:1906.06832 (2019).
Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." Advances in neural information processing systems. 2019.
Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

Appendix

The architectures for the Evolved CNN and the VGG16 based CNNs are included here.

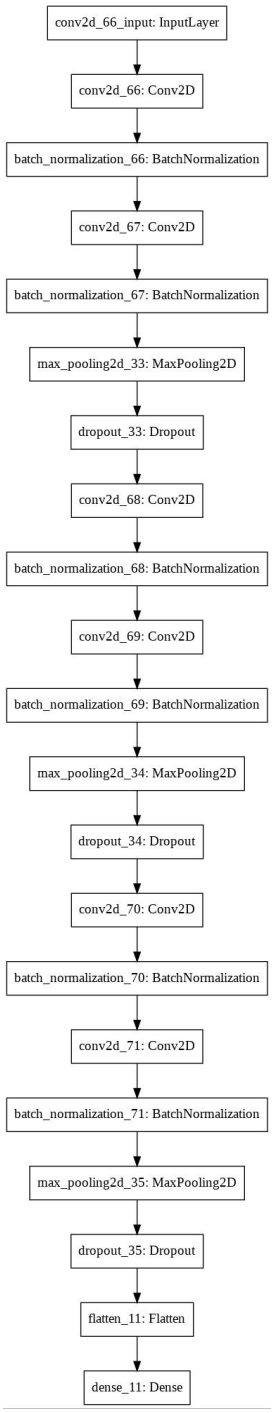


Figure 5: Evolved CNN

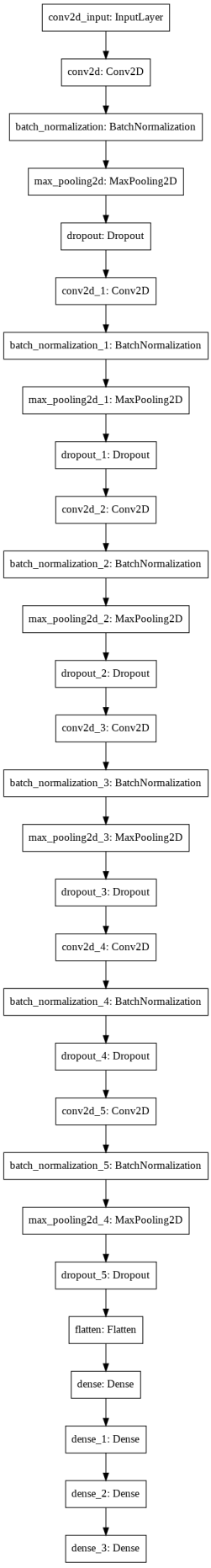


Figure 6: VGG16 Based CNN