# Object Classification using Convolutional Neural Networks on CalTech-256 Dataset

Nasif Imtiaz, Saad Mohammad Abrar, Taufiq Islam Protick
simtiaz@ncsu.edu,sabrar@ncsu.edu,tprotic@ncsu.edu
Department of Computer Science, North Carolina State University
Raleigh, North Carolina

## KEYWORDS

datasets, neural networks, CNN, CalTech-256, computer vision, object classification, image classification

**ACM Reference Format:**
Nasif Imtiaz, Saad Mohammad Abrar, Taufiq Islam Protick. 2021. Object Classification using Convolutional Neural Networks on CalTech-256 Dataset. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

There are a theory which works quite well.

## 1 INTRODUCTION & BACKGROUND

### 1.1 Problem Statement

Intelligent systems equipped with the advances in artificial intelligence and computer vision that perform visual recognition have gained much attention recently[5][8]. In conjunction to that, advances in terms of mobile internet and social media, the volume of image data over the years has exploded unprecedentedly. In keeping up with the pace, there are significant improvements in the image processing techniques as well. For example, in the past several years, we have seen object detection performance shoot up from approximately 30 percent in mean average precision to more than 90 percent today, on the PASCAL VOC benchmark[6]. In addition to that, for image classification on the challenging ImageNet dataset, state-of-the-art algorithms now exceed human performance. These improvements in image understanding have begun to impact a wide range of high-value applications, including video surveillance[10], autonomous driving[7], and intelligent healthcare[11]. Deep Learning is the central reason behind the recent advances in image recognition. Complementing with the availability of vast computational resources, deep learning is emulating great success by the development of powerful models. For a variety of image recognition tasks, carefully designed deep neural networks have greatly surpassed previous methods that were based on hand-crafted image features.

Image classification can be defined as the supervised process of observing instances and labels of known object classes during training and learn association patterns that can be used during inference of type of objects. As an important aspect of image processing, image classification has become one of the international popular research fields. In recent years, the powerful ability with feature learning and transfer learning of **Convolutional Neural Network (CNN)** has received growing interest within the computer vision community, thus making a series of important breakthroughs in image classification.

**We introduce our problem as an *image classification* task over the CalTech-256 dataset that contains 256 object categories containing a total of 30607 images. Given the success of CNNs in the relevant fields, we implemented VGGNet, a commonly used CNN architecture in image classification and prepared a comparative study of its performances with a Fully Connected Network on the CalTech-256 dataset. In addition to that, we analyzed the effect of using data augmentation and transfer learning when applying CNN architecture to evaluate the change in performance.**

### 1.2 Related Work

There has been significant work done in the context of image classification based on object categories. Some of the state of the art approaches that has been employed on an smaller instance of CalTech-256 dataset, CalTech-101, has achieved superior performance. These approaches included improving scene representations such as the *spatial pyramid matching*[9] and improved hybrid classifiers such as *Support Vector Machines* and *K-Nearest Neighbours* [12]. [2] capitalized on the image representation idea of spatial pyramid matching and came up with an idea of identifying region of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*Conference'17, July 2017, Washington, DC, USA*
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

interest(ROI) in an image and use them in training classifiers. In addition to that, they employed *Random Forest & Fern* classifiers as a multiclass classifier. The ROI improved their performance by 5% and an overall improvement of 10%. Furthermore, works like [1] leverage the idea of using compact image descriptors in object detection to achieve state of the art results.

In close relation to the work we have done, there are also several open source projects that has used deep learning and artificial neural networks. For example, [3] has implemented fully connected layers and several architectures of convolutional neural networks using Keras on CalTech-256. They were able to verify a steep increase in the accuracy when using Convolutional Neural Networks instead of fully connected neural networks. They have also experimented using several activation functions like $ReLU$, $tanh$ and sigmoid and recorded their relative performances. Furthermore, they have trained the convolutional neural networks both from scratch and transfer learning, and experimentally verified that transfer learning results in significant improvement in performance both in terms of execution time and accuracy on a defined test case. As part of our project we want to emulate the results using different experimental setups.

## 2 METHODS

We first explain our approach and then in following section justify the rationale behind them.

### 2.1 Approach

*2.1.1 Data Pre-processing:* Our input data are raw images in *.jpg* format. To feed the data into our classification model, we perform following pre-processing:

(1) We resize the images to (128,128) square shape maintaining the original aspect ratio.
(2) If any image in the dataset are smaller than this chosen size, we fill the leftover area with uniform black pixels.
(3) We then convert the images to an floating-point array of (128,128,3) shape where the last axis contain the color information from three spectral bands as our input images are in RGB mode.
(4) We then divide all pixel values by 255 in order to normalize them within range [0,1]

*2.1.2 Classification Models:* We propose four steps of broad-level models below where each step will be used as a *baseline* for the following step.

(1) **ANN:** 2-Layer Artificial neural network with 2048 neurons in each hidden layer. We use *random normal* weight initialization and *relu* activation in the hidden layers. We use *softmax* activation in the last layer and *categorical crossentropy* as loss function as ours is a *single-label multi-classification task*. We use $laearning_rate = .1, dropout = .5, batch_size = 64, epochs = 10$.
(2) **CNN:** A popular CNN architecture: VGGNet
(3) **Data Augmentation:** We add augmented data to our models. Keras has a *ImageDataGenerator* module to augment dataset by generating new samples from existing samples.
(4) **Transfer Learning:** Using an already trained model as a base, and then adding new parameters (layers and neurons) to that model and train the new parameters on the target dataset is called 'Transfer Learning'. We use VGG16 as our base model as it is one of the famous image classification model trained on a large dataset. We then add one to three fully connected layers to evaluate accuracy on CalTech-256 DataSet.

*2.1.3 Tuning Hyper-parameters:* We plan to tune two hyper-parameters in our experiment:

(1) Batch size: Smaller batch size takes less time to train a network under same epochs. Therefore, we will try to find a batch size that is small yet achieves baseline performance for each model.
(2) Epochs: We will keep track of training and validation loss to find the right epoch count to avoid overfitting.

*2.1.4 Model Evaluation:*

(1) For all models we use 80% data as the test data and 20% of the data as the training data.
(2) We use accuracy as the sole metric to evaluate the models.

### 2.2 Rationale

For each steps from corresponding subsection in methods, we justify our rationale here:

*2.2.1 Data pre-processing:*

(1) We resize the images to a square size as ANNs and CNNs usually take square shaped image. CNNs have a fully connected layer at the end of the network structure where we need to flatten the data. The square shaped input format is thus a prerequisite before flattening the data. However, we maintain original aspect ratio in order to not lose any pattern in the image. We chose (128,128) as most of our images lie between the range 256-500 pixel shapes, therefore, we don't lose too much information while still keeping a relatively smaller parameters at 128.
(2) Filling leftover pixels with uniform black color essentially tells the model that it is part of the background without the presence of any pattern.
(3) Neural network only works with floating-point data.

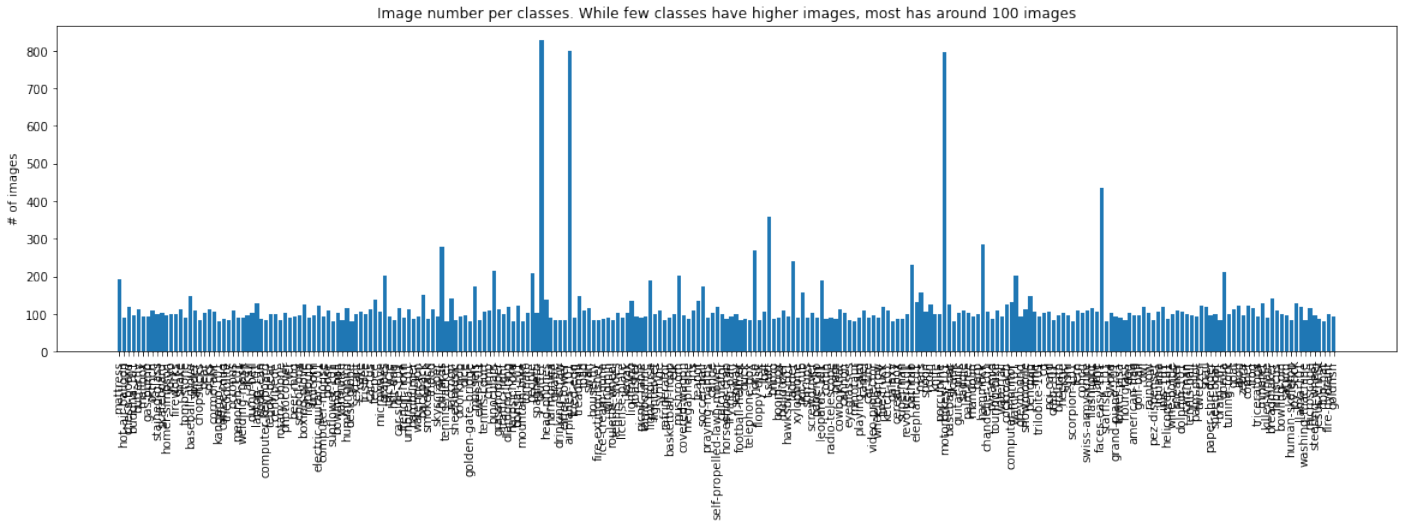


**Figure 1: Every category has at least 80 image samples**

(4) Normalizing the values between [0,1] makes the network computationally efficient without losing any information.

*2.2.2 Classification Models:*

(1) We first start with a 2-layer artificial neural network. It is known that even smaller capacity CNNs work much better than higher capacity ANNs. Therefore, we start with ANNs to use as a baseline before CNNs and evaluate the improvement. We define ANNs as commonly used hyperparameters.
(2) We perform commonly known CNNs architecture known to be well suited for image classification: VGGNet.
(3) Too few samples can cause overfitting in CNNs. Data augmentation is a commonly used technique to generate more data samples from existing data via some random transformations.
(4) A pre-trained network is a network whose initial weights come from a pre-trained network that was trained on a large dataset. If the original dataset was large and generic, then the *spatial hierarchy* of learned features boosts performance when dealing with a new image dataset. In this study, we also plan to select the best subset of object categories from within the CalTecch-256 dataset and use that trained model on the rest of the categories to evaluate transfer learning within a small dataset but with related objects. To the best of our knowledge, this experiment on Caltech-256 dataset has not been performed before.

*2.2.3 Tuning Hyper-parameters:* We chose to tune batch size and epoch as both will tell us how we can train the model in lesser time yet achieve good performance. We are unable to tune other hyperparameters as training CNNs on image dataset requires extensive computing power. Therefore, we chose commonly used values for other hyperparameters.

*2.2.4 Model Evaluation:* As our dataset is well-distributed in all categories, accuracy is the right metric for such single-label multi-class object classification.

## 3 EXPERIMENT

Here we explain our dataset, the hypotheses we will evaluate, and design of our experiment.

### 3.1 Datasets

We use Caltech-256 dataset [4] curated by Griffin et. al. The dataset has 256 object categories while the rest of the images fall into a 'clutter' category. Therefore, the dataset has 257 categories and 30,607 samples in total. Figure 1 shows the sample image distribution per categories. While some category has higher number of images, all categories have at least 80 images.

### 3.2 Hypotheses

(1) CNNs performs better in image classification than ANN on the CalTech-256 dataset.
(2) Data augmentation can improve the accuracy on the CalTech-256 dataset.
(3) Transfer Learning using VGG16 model trained on ImageNet dataset can improve performance with smaller epochs on CalTech-256 dataset..

## 3.3 Experimental Design

We are developing our code in following github repository: **https://github.com/nasifimtiazohi/Caltech256-project**

We have conducted our experiment using NVIDIA GEFORCE RTX 2060 GPU which has a Turing Architecture, Boost Clock of 1680 MHz to train our model given the extensive computational requirements. Firstly, after the 80-20 split of the dataset into training and test data instances, we trained an ANN model, VGGNet, a variant of a CNN architecture. Hence, we applied data augmentation and transfer learning to VGGNet to observe the changes in performance and accuracy.

*3.3.1 ANN:.* As an ANN, we built a fully connected (i.e. dense) network. Fig 2 shows the architecture.

*3.3.2 CNN:.* We are using VGG as the variant of a CNN model to train and test the data. This neural network architecture is known for its $3 \times 3$ filters. In our design, its architecture is a series of Convolutional Layers and Relu Layer that are followed by a pooling layer whose padding is valid. Batch normalization is included between each convolution and ReLU layer of the network so that weights in the network don't become imbalanced with extremely high or low values. Inclusion of batch normalization increases the speed in which training occurs and reduce the influence of the outlying large weights. Four iterations of conv2D, batch normalization, ReLU, and max pooling 2D is followed by a layer that flattens the 2D data into a numpy array. Later on, this flattened 1D data goes through a fully connected hidden layer with 512 nodes, followed by batch normalization, and ReLU. Finally the output layer is a fully connected layer with 10 nodes with softmax activation function, that classifies each of the 10 image classes that we want classified.

*3.3.3 Data Augmentation:* The Caltech-256 dataset has a very limited number of images per class. Although there are 3 classes that have over 800 images, the other classes have on an average 100 images per class. Two work with such a small number of training data for almost all classes, a model may face two problems: it will have less data to learn from, and it will overfit to the existing data. Consequently it will fail to be a robust learner. In this experimental design we will discuss how Data Augmentation technique has been implemented in the training data to solve these two issues.

*Library used:* We are using `ImageDataGenerator` class of `Keras` library to do this technique. Detail implementation of this is included in the Github Repository named `dataAugmentation.ipynb`. The implementation is included in `experimentVGGwithDA.ipynb`.

*Implementation:* An object of ImageDataGenerator created with a default constructor means that no data augmentation has been done. To implement data augmentation, an object

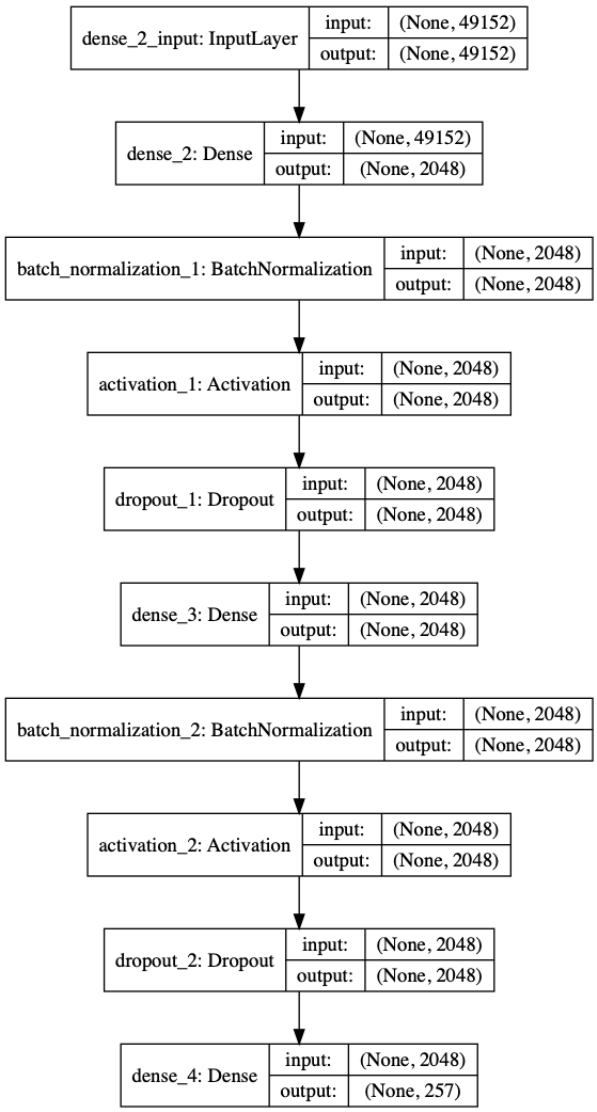


**Figure 2: Dense network architecture**

of this class has to be created specifying which modifications will be done to the existing data by putting arguments in the constructor. After preprocessing the data and splitting up the train and test data into 80-20 split, we designed a CNN model whose description is given in the subsection above. Next we added the modifications shown in Figure 3 in an ImageDataGenerator class.

The impact of such transformations to any image is demonstrated in Figure 4. If we assume that the original dataset had only one image of a dog before, now it will have eleven images. Although they are of the same dog, but it will help make the model robust. If the image at the left of Figure 4 was the only image with which the model was trained, the model

```
In [78]: generator = ImageDataGenerator(
             rotation_range=40,
             width_shift_range=0.2,
             height_shift_range=0.2,
             shear_range=0.2,
             zoom_range=0.2,
             horizontal_flip=True,
             fill_mode='nearest')
```

**Figure 3: The `generator` object, an instance of the `DatatImageGenerator` class is instantiated with the arguments mentioned in this Figure. The output of such operations to a particular image is shown in Figure 4.**

would have failed to predict dogs in any test images where they are facing left. Adding horizontal flip makes the model robust against such bias. Moreover, inclusion of rotation of the original image makes the model robust against test data that are not parallel to the horizon. This technique is also

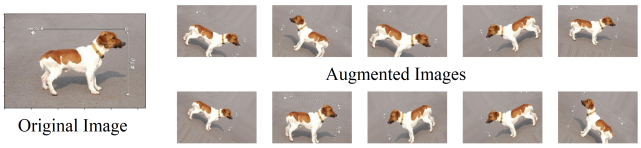


**Figure 4: In the left, we have an original image from the dataset. In the right, the implemented augmentation technique added reasonable modifications to the data that includes rotation, width and height shift, shearing, zooming and horizontal flip. We did not consider vertical flip as an upside down image is not the normal orientation even in unseen data.**

important because it prevents the model from overfitting. With having so many images to train with, the CNN's filters change in a different way, detect edges at different parts of an image (due to shifting an image), and also at different orientations (due to rotation). Adding these synthesized images thus helps make the learner deal with limited dataset per class and also with overfitting the data.

*3.3.4 Transfer Learning:* A neural network starts with randomly initialized weights. However, we can start with an already trained model with a different dataset which may give us a *headstart*. This technique is referred to as 'Transfer Learning'. In image classification, trained model in any dataset may help to extract the low level image features (e.g. edges, contrast, curves, highlights) which can be a starting point for a model on the target dataset. We can add new layers to this model which will help us to further identify the target high-level features of our dataset. Through this technique, we can achieve good accuracy with fewer parameters and hence, less training.

In this experiment, we use VGG16 trained on ImageNet dataset as a base model. We flatten the output layer of te base model then construct models in three different ways:

(1) Add one layer of 512 neurons fully connected with the base model.
(2) Add two layers of 1024 and 512 neurons fully connected with the base model.
(3) Add three layers of 2048, 1024 and 512 neurons fully connected with the base model.

For all the models, we use Data Augmentation technique as well. We then evaluate the accuracy to identify the correct level extension need to VGG16 model in order to classify objects in CalTech-256 dataset. We run the one layer extension with 80 epochs to find loss and accuracy to stabilize after 10 epochs. Therefore, for the 2 and 3 layer extension, we only train the network for 10 epochs as computation is expensive.

## 4 RESULTS

We ran four variants of neural networks (ANN, plain CNN, CNN with data augmentation, and CNN with transfer learning and data augmentation) for 257 classes in a GPU of the ARC cluster. Notice that the partial results presented in the midway report presented data for 10 classes and models were run on a CPU. The CNN has the same architecture mentioned in Section 3.3.2. The training (24,485 images) and validation set (6,122 images) had an 80-20 split. All the models including the ANN ran for 80 epochs, had a learning rate of 0.001 and batch size was 32. Below we demonstrate the results and how they support/refute out hypotheses. ANN: We ran the dense network for 80 epochs with a batch size = 32. Figure 5 shows the loss and accuracy for both training and test set up to 80 epochs. We see that both accuracy and loss stabilizes after 10 epochs and remains so except a temporary spike near 70th epochs. The accuracy remains stabilized at around 26%.

### 4.1 CNN without any enhancement techniques

The code for this model can be found in the GitHub repository at `scripts/CNN.py`. Figure 6 shows the results. We see that just after 10 epochs, the training accuracy gets closer to 1. Thus the training loss also becomes minimum after 10 epochs. However, we see that the accuracy of the validation dataset does not improve over the epochs. It reaches at its peak accuracy (36%) on the fifth epoch and remains fluctuating in the range 32-36% for the rest of the epochs. The best training accuracy achieved by the model (0.9997) was on the 74th epoch with loss 0.360993. At this time the

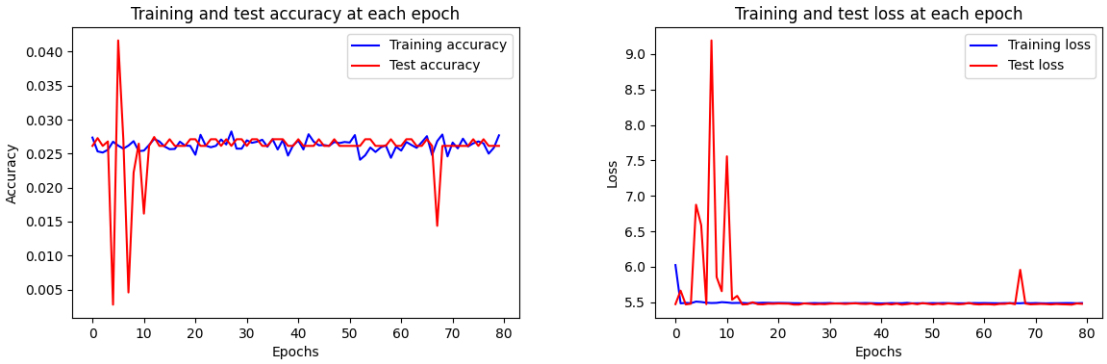


**Figure 5: Accuracy and Loss for Dense Network**

validation accuracy was 0.3609. The best validation accuracy was achieved by the model already at 5th epoch (0.367). The model evidently overfits.

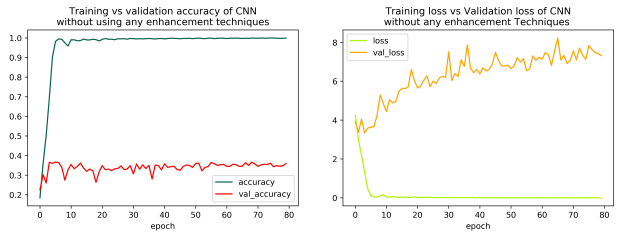


**Figure 6: The x-axis denotes epoch (total epochs = 80), the y-axis in the left plot denotes accuracy, and in the right plot denotes loss. (Left)Training vs validation accuracy and (Right) Training vs validation loss of the CNN model without any enhancement techniques.**

## 4.2 CNN with Data Augmentation

The code for this model can be found in the GitHub repository at
`scripts/data_augmentation.py`. Figure 7 shows the results. Due to addition of data augmentation techniques, we see significant changes in the model's accuracy and loss compared to the model with no data augmentation incorporated. The training accuracy reaches at its peak at a later time step (78th epoch).The training loss follows a reciprocal trend as well. The model reaches at its peak validation accuracy (45%) on the 68th epoch and never goes down from 41% after 28th epoch. The best training accuracy achieved by the model (0.8422) was on the 78th epoch with loss 0.5448. At this time the validation accuracy was 0.4414. More discussions on hypothesis is at the Discussion section.

### *4.2.1 Discussions.*


**Figure 7: The x-axis denotes epoch (total epochs = 80), the y-axis in the left plot denotes accuracy, and in the right plot denotes loss. (Left)Training vs validation accuracy and (Right) Training vs validation loss of the CNN model without any enhancement techniques.**

***Comparisons between CNN with no enhancement techniques and CNN with data augmentation:*** Our rationales behind using data augmentation were that it increases training data and reduces overfitting. If we put Figure 6 and 7 in contrast, we see that our rationales were justified. In figure 7, the parabolic green and lemon green curves for accuracy and loss respectively shows that the model with data augmentation reduces overfitting compared to the sharp rise of training accuracy accuracy in Figure 6 already at epoch 5.

Moreover, one of our hypotheses were that Data augmentation can improve the accuracy on the CalTech-256 dataset. The comparative rise of validation accuracy from 36% at the CNN with no data augmentation to 45% at the CNN with data augmentation reflects that.

## 4.3 Transfer Learning

Figure 8, 9, 10 shows the accuracy and loss for models with 1,2, and 3 layer extension after VGG16 base. We see that they all converge to accuracy around 46% at 10 epochs. Based on the results we have following observations:

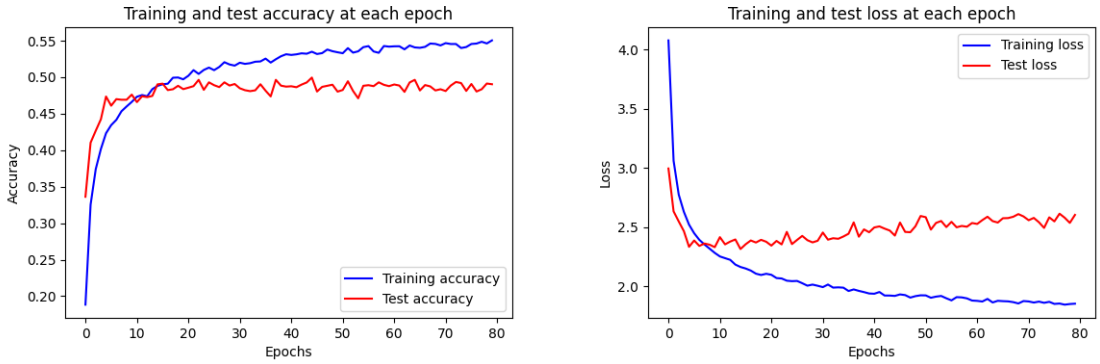


**Figure 8: Accuracy and Loss for Transfer Learning from VGG16 base with 1 Layer extension**

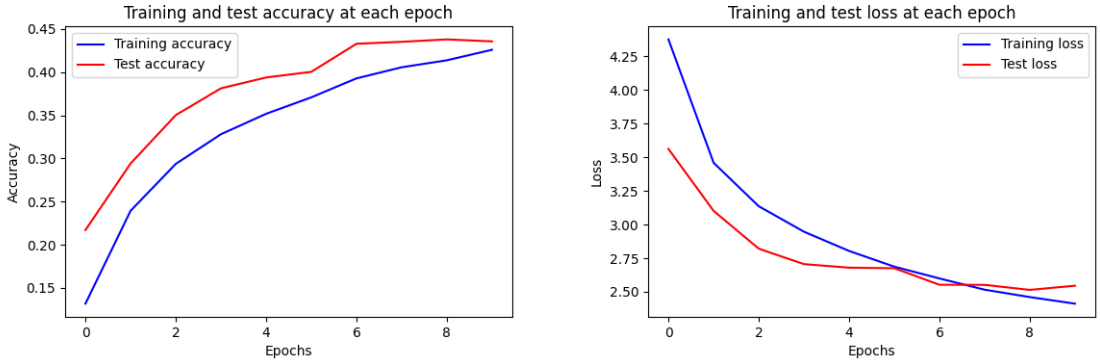


**Figure 9: Accuracy and Loss for Transfer Learning from VGG16 base with 2 Layer extension**

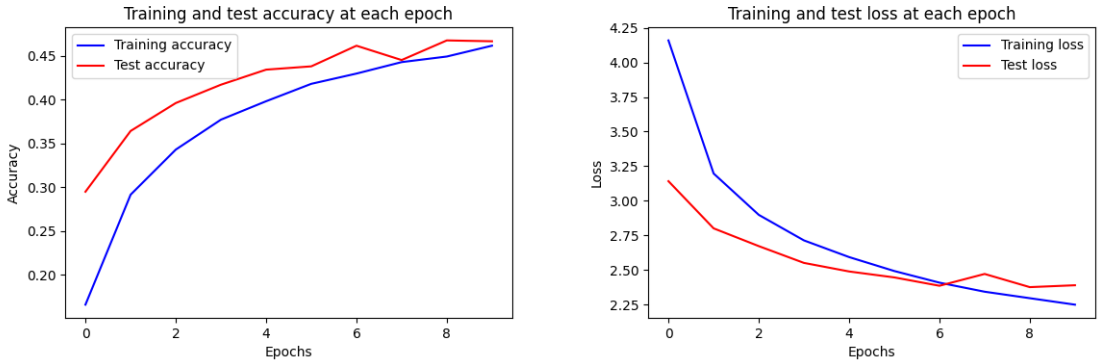


**Figure 10: Accuracy and Loss for Transfer Learning from VGG16 base with 3 Layer extension**

(1) One-layer extension provides similar results to two or three layer extension.
(2) Transfer learning can help achieving equal or higher accuracy to CNN with smaller parameters (One fully connected layer of 512 neurons) and smaller epochs, therefore, significantly lesser training time computational power.
(3) Prior work [3] has achieved higher accuracy on this dataset (57%) using VGG19 as a base model. Therefore, our results indicate that choice of base model influences performance.

| Model | Epoch at peak validation accuracy | Accuracy | Loss | **Validation accuracy** | Validation loss |
|---|---|---|---|---|---|
| ANN | 5 | 0.026179 | 5.503801 | **0.041653** | 6.588444 |
| CNN with no enhancements | 5 | 0.982112 | 0.119893 | 0.**367037** | 3.64355 |
| CNN with data augmentation | 68 | 0.819563 | 0.635642 | **0.453937** | 3.645688 |
| CNN with data augmentation and TransferLearning | 44 | 0.53494 | 1.919411 | **0.49951** | 2.428604 |

**Table 1: The table summarizes the peak validation accuracy (the boldfaced column) across all four models.**

### 4.4 Overall review of Top validation accuracy across all models

Table 1 summarizes the peak validation accuracy across all models. If we observe the boldfaced column in the table, we will notice that, when a more sophisticated (CNN in place of ANN) or enhanced (Data augmentation or Transfer learning in place of no such techniques) models are designed the validation accuracy gets better. The model with the lowest score is ANN, any other CNN model performed better than the ANN designed. This supports our first hypothesis. Moreover, we see that due to data augmentation, the model at the third row has less training accuracy compared to the one that used no data augmentation. This also supports our second hypothesis. The last model, built with transfer learning gives the highest validation accuracy with a smaller number of epoch (44th epoch compared to 68th epoch at data augmentation). This supports our third hypothesis.

## 5 CONCLUSION

In this report, we have shown that, model enhancements such as transfer learning and data augmentation has positive implications on the results of performance and accuracy. As seen in the table 1 above using a Convolutional Neural Network, there is a sharp rise in accuracy from ANN . In addition to that, the experiment empirically proves that transfer learning enhances the performance in terms of accuracy. Therefore, given the advantages of pre-trained model, we have seen that we reach convergence at a much higher epoch, saving us precious computational time. Therefore, it opens for us an avenue for further experiments we could try using the subset of the training data or even semi-supervised approach to see if we actually can achieve the same accuracy or even exceed it.

## 6 MEETINGS

The meeting times, agenda and attendance are placed in Table 2.

## REFERENCES


[1] Alessandro Bergamo, Lorenzo Torresani, and Andrew W. Fitzgibbon. 2011. PiCoDes: Learning a Compact Code for Novel-Category Recognition. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2088–2096. http://papers.nips.cc/paper/4319-picodes-learning-a-compact-code-for-novel-category-recognition.pdf

[2] Anna Bosch, Andrew Zisserman, and Xavier Munoz. 2007. Image classification using random forests and ferns. In *2007 IEEE 11th international conference on computer vision*. Ieee, 1–8.

[3] Inc. GitHub. 2018. Open Source Survey. https://github.com/nickbiso/Keras-Caltech-256.

[4] Gregory Griffin, Alex Holub, and Pietro Perona. 2007. Caltech-256 object category dataset. (2007).

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[6] Derek Hoiem, Santosh K Divvala, and James H Hays. 2009. Pascal VOC 2008 challenge. In *PASCAL challenge workshop in ECCV*. Citeseer.

[7] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. 2015. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716* (2015).

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[9] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. 2006. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, Vol. 2. IEEE, 2169–2178.

[10] Xinchen Liu, Wu Liu, Tao Mei, and Huadong Ma. 2016. A deep learning-based approach to progressive vehicle re-identification for urban surveillance. In *European conference on computer vision*. Springer, 869–884.

[11] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. 2018. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics* 19, 6 (2018), 1236–1246.

[12] Hao Zhang, Alexander C Berg, Michael Maire, and Jitendra Malik. 2006. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, Vol. 2. IEEE, 2126–2136.

| Dates | Agenda | Attendance | Time |
|---|---|---|---|
| 04/06/2020 | ANN, VGGNet and Overall project plan | All members were present | 9:00AM-11:00AM |
| 04/09/2020 | Data Augmentation | All members were present | 6:00PM-7:00PM |
| 04/11/2020 | Transfer Learning | All members were present | 6:00PM-7:00PM |
| 04/13/2020 | Experiments and GPU setup | All members were present | 4:00PM-5:00PM |
| 04/19/2020 | Remaining Issues | All members were present | 1:00PM-2:00PM |

**Table 2: Meeting times, Agenda and Attendance**