Image Recognition to classify recyclable products using Transfer Learning

Rimmi Dhammi

Submitted for the Degree of Master of Science in

MSc Artificial Intelligence

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

September 01, 2020

Declaration

This report has been prepared based on my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 8366

Student Name: Rimmi Dhammi

Date of Submission: 01 September 2020

Signature: Rimmi Dhammi

Acknowledgements

I would like to express my gratitude to Dr Chris Watkins who helped me refine my work throughout this experimental study. His suggestions helped me explore areas that were more relevant and fascinating, and ones that I would not have approached if left to my own devices.

Abstract

Image recognition domain in deep learning is not only mind bogglingly exciting, rapidly advancing but also very easy approachable. With multiple libraries and modules, and countless tutorials online, the task of devising practical solutions to real life problems that may or may not have commercial value can be given shape with resources available for free.

This statement should not be taken to undermine the significance of generating clean data and robust code and relentlessly working towards improving these building blocks to push model's accuracy towards acceptable limits , but to motivate implementation of current tools to figure out accessible solutions to mundane and time consuming tasks in human life.

Tasks like how much to water house plants and garden plants, what items to recycle and what to bin as general waste, calories in one's food, pests in your garden and how to control them, identifying your employee and which department he/she works in or even what kind of coffee and food your colleagues like [1]. All these tasks take effort to remember and some while do not cost anything if you forget the right details (like forgetting your co-worker's area of research), mistakes with others might be slightly more painful – economically (like killing a batch of Dracaena Marginata due to over watering, turbine failure due to delayed recognition of suffering blades [2] or worse, in medical sector – misidentification or delayed dignosis of metal illness [3].

It all boils down to human training time!

In any field.

The motivation of this project was to replace human training time with machine training time and weed out potential mistakes. The direct implication being – much time is saved and as in commercial sense time equates to money – the outcome of this undertaking can be considered as viable success.

Keeping true to my belief in previous sentence, initially the objective of my project was only training a model to identify a recyclable item from a non- recyclable commodity. But the lack of data, pushed the research into area of data augmentation and leveraging transfer learning for a model that can handle multiple uncorrelated tasks based on single input.

By the end of the project many assumptions like equal transfer of learning on different data set size or image size, low training time were discarded and effects of regularization, learning rates, normalization on dataset and on hidden layers, difference between feature extraction and fine tuning, image augmentation, validity of ImageNet weights for this task and overall extent of generalization of model trained using transfer learning became apparent.

This project with different data set can be used by any sector to achieve implementation worthy results.

Contents

1 Introduction

This decade, computer vision has seen remarkable and unrelenting progress, that many image recognition architectures have been set as benchmarks or state of the art for beginners in the field to look up to. Thanks to the widely available huge data sets and model generalization, someone else's hard work can be exploited for tasks that do not comply with original model's intentions. Achievable objectives include transfer learning, object detection [4], image segmentation.
This project to identify recyclable items also deals with sub task of identifying the object in the image. This can be viewed as two separate jobs on same input – an image. I experimented with multi-hot vectors that delivered acceptable results but bound the performance on one goal with the performance on another. In quest to achieve greater indpendence and control over two objectives led to architecture revision to single input, multi output using keras functional API.
I chose my inputs as Cifar100 dataset images [5] because many classes that it consitutes, can be deemed recyclable. But its biggest limitation with respect to image recognition task with deep neural networks is that it is a 'tiny image' data set and constitute only 60000 images of 100 classes, that restricts its generalization on unseen data. Another limitation is more generic in nature that plagues all image recognition tasks – correct detection under all visibility conditions – to be able to see image clearly, or identify an obscured image correctly.
But these were easily remedied by employing image upscaling, data augmentation [6] and transfer learning [7].
Once main architecture and dataset were in place, next task was to choose branch architecture (one for each task) and weights for transfer learning. The closest data set to Cifar100 genre is ImageNet [8] that constitute 15 million high resolution images belonging to roughly 22000 classes and thanks to the 'race to find benchmark on ImageNet', that there are 9 models (plus variants) pre trained on ImageNet that can be used for prediction, feature extraction or fine tuning.
This project slowly works towards its goal to separate recyclable items from non – recyclable ones, but intermediately explores pre-trained weights learning transferability of a model on Cifar100 dataset, transfer learning methods that work well, role of data augmentation, and hyperparameters for different transfer learning.
As a side note, I would like to mention as this was mostly implementation project, most help was soughted from keras documentation and tensorflow documentation.

2 Background Study

2.1 Transfer Learning

Transfer Learning is method that employs pre-trained models. A pre trained models is a saved model that was trained on a large dataset and is considered to have generalised enough, that it could be applied to other tasks that are different from its primary learning.

The prerequisite for a pre-trained network is that the original data set should be large enough so that features learned by the model could represent most real-world objects and hence can be utilized in different computer-vision problems, that may have labels and classes very different from the original data.

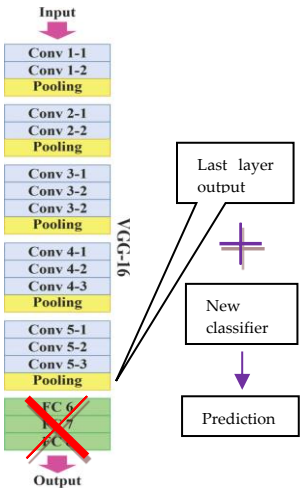
There are two approaches to transfer learning – feature selection and fine tuning.

2.1.1 Feature Extraction

Feature extraction method uses the features learned by the network from original data to extract relevant features from new data. These newly extracted features are then classified by a new classifier. This smaller and less complex classifier is trained from scratch.

This procedure is rather straightforward with CNN models, as the network usually comprises of conv, max-pooling, dropout, and normalization layers (a convolutional base) and finally has a head of FC layers whose output is given by a classification activation function.

All we do is take the output of the final layer of convolutional base, (just before the dense FC layer head), after running our new data through it. These collected features are then run through our smaller classifier to train it. The classifier could be logistic regression, SVM, another dense neural network or any other method in data analyst's kit.



For feature extraction, we do not need to retrain the entire model or run our data through it multiple time. The data is passed through the network only once – that renders this method very quick.

As convolutional base is by far the most expensive part of the pipeline, running each input through it only once makes this method very cheap.

But again, as we extract features in one run, this method does not support data augmentation.

2.1.2 Fine Tuning

The most important aspect of transfer learning is making learning possible on small dataset. Widely used technique for model reusability is called fine tuning. We first train our custom FC head by keeping the base model layers frozen and once

the model converges, we unfreeze a few of the top layers of a frozen model base and train both the newly added part of the model (in this case, the fully connected classifier) and these top layers.

The process runs end to end for the number of epochs specified. This does let us use data augmentation, that curbs overfitting problem tremendously. But for the same reason, fine tuning is very expensive, both in terms of time and memory.

This process fine tunes the higher order feature representations in the base model, making then more relevant to the task.

Short coming of transfer learning is that we cannot tune pre-trained network's parameters (such as the number of filters per convolution layer, or the number of layers in the network). Therefor we must rely on other regularization techniques, batch normalization and pre – processed inputs to achieve higher accuracy.

2.2 Data Augmentation

In this project I am tackling a small data set and training on models with huge capacity. Any pre-trained model with ImageNet weights has excess of 25,00,000 trainable parameters. And my training dataset size is 5000 (or 220 in case of self-generated data). It is like putting a size 3 foot in a size 8 wellingtons – overfitting is inevitable in the case when I would use some layers from conv block and try and update their weights using my small training data, a process called fine tuning.

As F. Chollet said, "Overfitting is caused by having too few samples to learn from, rendering you unable to train a model that can generalize to new data. Given infinite data, your model would be exposed to every possible aspect of the data distribution at hand: you would never overfit. Data augmentation takes the approach of generating more training data from existing training samples, by augmenting the samples via several random transformations that yield believable-looking images. The goal is that at training time, your model will never see the exact same picture twice. This helps expose the model to more aspects of the data and generalize better."

My best approach here was to use Keras ImageDataGenerator that can be used to perform random transformations on the dataset. Model does not see the original dataset anymore, but the transformed images generated by this instance.

.

3 Project Stage 1

The Stage 1 of the project focused on getting all hardware and software dependencies in line, so that execution runs smoothly.

To tap GPU power, CuDNN, GPU drivers and CUDA were installed with compatible versions of tensorflow-gpu. Project was set up in virtual environment to counter any clashes with version of other dependencies. The extensive guide to installation of CUDA can be found on

1. CUDA Toolkit documentation [9]
2. Tensorflow GPU support [10]

My system configuration					
CUDA enable GPU card	NVIDIA GPU driver	CUDA	CuDNN	Python	Tensorflow
GeForce GTX 1050 Ti	Game Ready – version 452.06	10.1	7.6	3.7.6	2.1.0

Note that there is latest version of each available but due to tensorflow-gpu requirements, most were rolled back to their previous releases.

Installing CuDNN and CUDA would frequently throw an Error:
'Failed to get convolution algorithm. This is probably because cuDNN failed to initialize, so try looking to see if a warning log message was printed above.'

This error forced to shut down each running sequence and clearing RAM – primarily by restarting computer, which was a very time consuming and wasteful ordeal. The error did not rectify itself even when I started using tenth of the data available or using batches.

As all my versions were mutually compatible, and I switched to using very low data, the error was finally corrected by adding:

```
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
```

at the beginning of the worksheet. This allowed for effortless execution of CNN architecture and model training.

This issue has been discussed in greater detail on page:
https://stackoverflow.com/questions/53698035/failed-to-get-convolution-algorithm-this-is-probably-because-cudnn-failed-to-in

4 Project Stage 2

The Stage 2 of the project focused on:

- Data selection
- Data generation
- Model architecture selection

4.1 Data selection

4.1.1 Get data relevant to task

Data selection aim was to get relevant data and largely depended on the task at hand, that is, sorting items as either recyclable or non-recyclable. Looking for a data set with images of such items, lead to Cifar100 – that comprises of classes like bicycle, ray, plate, cup, bowl, willow tree – general mix of worldly items without extensive category overlap – this averted the data set category bias. This is also a balanced data set with respect to native classes, but not based on recycling or not-recycling and is fetched from Keras dataset library.

Characteristics of Cifar100 dataset					
	Classes	Training images	Test images	Resolution	Channels
fine labelled	100	500 per class	100 per class	32 x 32	3
coarse labelled	10	5000 per class	1000 per class	32 x 32	3

4.1.2 Class overlap with pre-trained network dataset

The second criteria for data selection was class overlap with the dataset that was primarily used to train the network – whose weights we are going to employ in our model training. ImageNet and Cifar100 have 54 categories in common, with some belonging to recyclable and others to non-recyclable category.

Overlapping categories are, 'apple', 'aquarium fish', 'bear', 'bee', 'beetle', 'bicycle', 'bottle', 'bowl', 'bridge', 'bus', 'butterfly', 'camel', 'clock', 'couch', 'crab', 'crocodile', 'cup', 'elephant', 'fox', 'keyboard', 'lamp', 'leopard', 'lion', 'lizard', 'lobster', 'mushroom', 'orange', 'otter', 'pickup truck', 'plate', 'poppy', 'porcupine', 'rabbit', 'ray', 'shark', 'skunk', 'snail', 'snake', 'spider', 'squirrel', 'streetcar', 'sweet pepper', 'table', 'tank', 'telephone', 'television', 'tiger', 'tractor', 'train', 'turtle', 'wardrobe', 'whale', 'wolf', 'worm'

		ImageNet	Cifar100
Class	ImageNet	1000	54
Overlap	Cifar100	54	100

4.1.3 Amount of data for training and testing

Of the total dataset available, only small portion was used for actual training. The smaller data extracted was balanced with respect to the native classes.

Data Usage – tenth of original data set			
	Train data	Validation data	Test data
Feature Extraction	5000	1000	1000
Transfer learning	5000	1000	1000

4.2 Data gathering

Apart from cosmetic data, some real-world data was gathered to test the robustness of the model. This chore was mainly manually taking photos of a recyclable item of a class that was already there in Cifar100.

This data set was generated from my directory using Keras ImageDataGenerator and then processed to same data type as Cifar100 data. This data set is a balanced dataset too.

Characteristics of real-world data set				
Total Images	Classes	Image dtype	Label dtype	Resolution
316	11	'uint8'	'int32'	Model dependent

4.2.1 Class overlap with pre-trained network dataset and training dataset

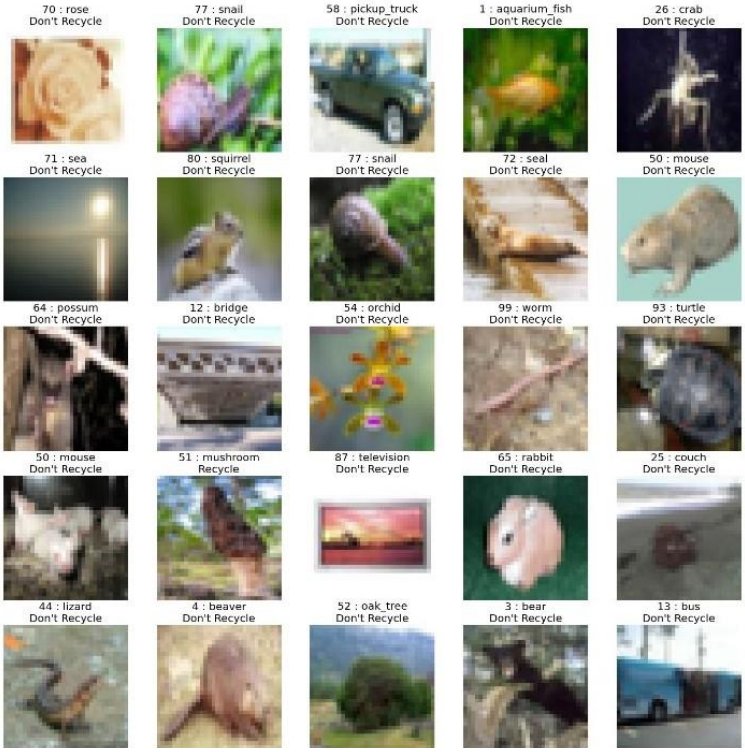
Overlapping classes			
	Test data	ImageNet	Cifar100
Test data	11	6	11

Six of the gathered classes are present in both ImageNet and Cifar100– 'apple', 'bowl', 'clock', 'cup', 'orange' and 'plate'.

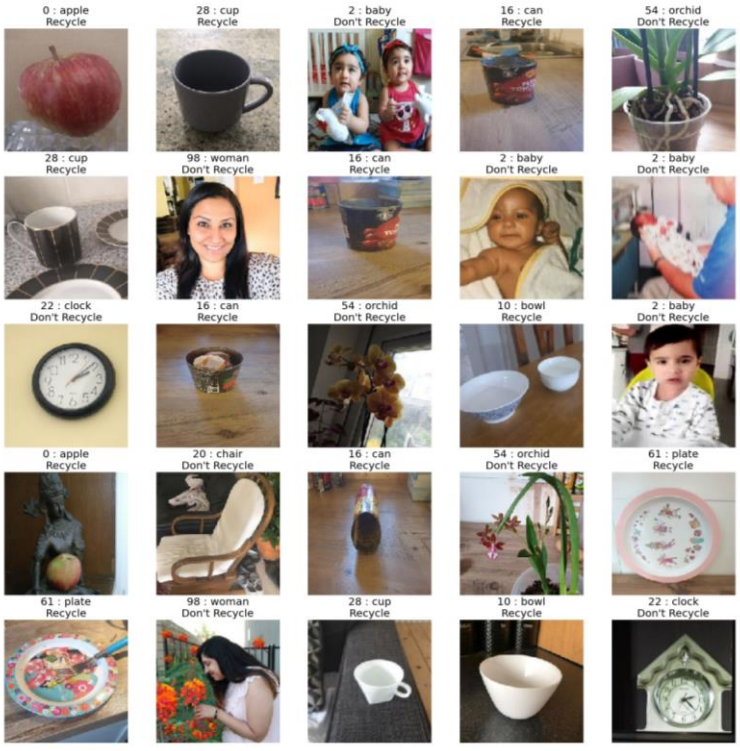
Other five classes are present only in Cifar100 – 'baby', 'can', 'chair', 'orchid' and 'woman'.

This was done intentionally to check the pre -trained model's confidence on classes that it is familiar with already vs new classes that it has learnt to predict after fine tuning.

Data Visualization



Cifar100 dataset

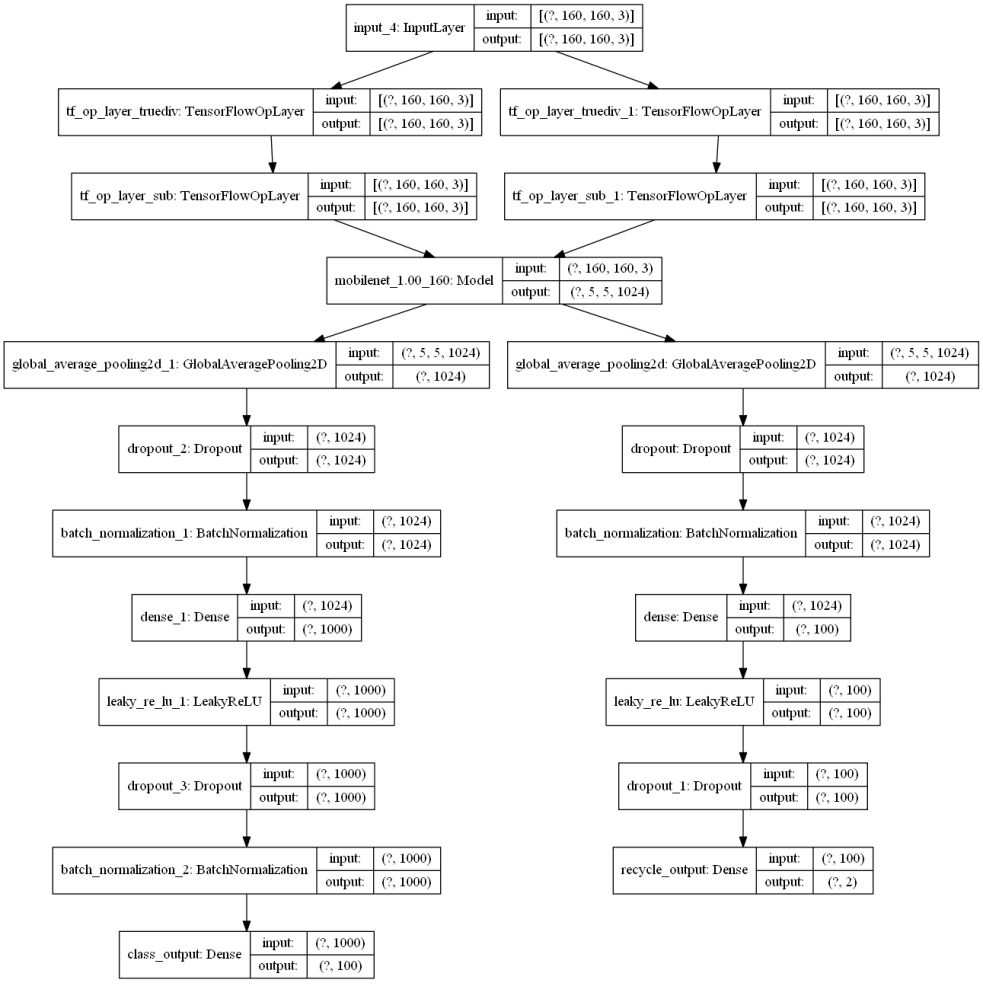


Generated dataset

4.3 Model architecture selection

Initially the experiment started with single input, single output architecture, using multi-hot vectors, where a binary vector representing recycling set was added column wise to categorical vector of size (, 100) of Cifar100, so that now it became (, 101).

Very early in the test, it was discarded for single-input, multi-output architecture using Keras Model layers. This architecture mother branch takes in single input, that is our images and branches out to multiple sub- architectures, wherein each can deal with a subtask of identifying different aspects of the object. In our case, it will be binary classification on one branch to identify recyclable products and categorical classification on the other branch to identify native classes. The sub-branch can incorporate pre-trained networks and can have individual FC heads and outputs.



4.3.1 Selection of sub – architecture

Sub architecture for this project includes a pre- trained model and custom fully connected layers with output.

There are 9 main models available in Keras applications module that have been pre-trained on ImageNet images. Selecting a pre-trained model involved these considerations:

4.3.1.1 Minimum supported resolution

Of 9 models and their versions, these models supported the resolution of Cifar100 images (that is, 32 x 32) so that training and fine tuning could be successfully employed, if using the original size.

Model	Optimal image resolution	Model	Optimal image resolution
VGG16, VGG19 [11]	224 x 224	ResNet50	224 x 224
EfficientNetB0 – EfficientNetB7 [12]	224 x 224	ResNet101	224 x 224
ResNet152	224 x 224	ResNet152V2	224 x 224
ResNet50V2	224 x 224	MobileNet [13]	224 x 224
MobileNetV2 [14]	224 x 224	DenseNet169 [15]	224 x 224
DenseNet121	224 x 224	DenseNet201	224 x 224
NASNetMobile	224 x 224	NASNetLarge	224 x 224

Eventually during the training, upscaling of Cifar100 images became necessary to achieve higher validation accuracy, hence Inception model [16] can be included in the list too.

4.3.1.2 Trainable Parameters

Trainable parameters became one of the criteria keeping in mind

- Training data set size – as more parameters on smaller data set would lead to overfitting
- Resource constraints – memory available vs computations required
- Depth of the model

4.3.1.3 Model simplicity

A simple model is the one with basic layers like convolution layers and dense layers making the most of the networks' structure. A simple model rendered the experiment more confidence and gave datum to measure results on a more complex model with batch normalization layers and depth wise separable convolutions.

4.3.1.4 Modal's performance on ImageNet dataset

How well the model did on ImageNet dataset. This data was gathered from Keras applications documentation – where top 1 and top5 accuracy of models is listed when tested on ImageNet dataset.

Keeping in mind the top four reasons, MobileNet was chosen to carry out this experiment.

Model statistics			
	Cifar100 resolution support	Total Parameters	Depth of the model
MobileNet	Yes	4,253,864	93

Model statistics			
	Total Parameters	Model's performance on ImageNet, Top 1	Model's performance on ImageNet, Top 5
MobileNet	4,253,864	0.704	0.895

4.4 Evaluating pre-trained weights before training

As we were using MobileNet pre-trained on ImageNet, it was interesting to observe how it performed on Cifar100 model, even before we trained it. Given that there are 54 overlapping classes, my initial assumption was that it would be able to predict those classes accurately.

MobileNet predictions on Cifar100 before training	
Accuracy	0.1548

5 Project Stage 3

This stage includes

- Training
- Test results
- Learnings

5.1 Training

Training the model involved these steps

- Preparing the data
- Building input pipeline
- Building model and loading pre trained weights
- Feature Extraction
- Fine tuning

5.1.1 Preparing the data

Primary step was data visualization (data type, data shape, resolution, data size, distribution, if its stored as array or tensor) and ascertaining if it was fit to be trained with. Understanding what the pre-trained model was trained with, helped pre-process the data in the right direction. Of 100 labels in Cifar100, 54 are present in ImageNet, making it highly likely to transfer 'the learning'.

Cifar100 comes with train, test split in size 32 x32, so no further set was needed. Images used for training and testing were upscaled to either 160 x 160 or 224 x 224 resolution. To accommodate expensive training and pre-processing, only 5000 of total 50000 training images were used and 1000 of 10000 test images were used.

For self-generated data, of 316 images in 11 classes, 20 images from each class were used for training set and rest were used for validation set. The target size of self -generated data was kept 224 x 224.

In both cases, as data set was not large enough, data augmentation was used to artificially introduce diverse samples. This was implemented using data generator.

Train dataset features				
	Training set	Validation set	Resolution	Upscaling
Cifar100	5000	1000	32x 32	5 times; 7 times
Self- generated	220	96	224 x 224	None

Train dataset features			
	Data Aug training samples	Data Aug validation samples	Test data
Cifar100	~150,000	~150,000	generated set – 220 samples

The result of using generator for data augmentation was, that I was able to feed approximately 150,000 training images to my model. Same goes for the validation set, though no augmentation method was assigned to it.

Apart from upscaling and augmentation, data was subjected to rescaling as well. This was achieved by using the pre-process method of the pre-trained model, that rescales pixel values from [0-225 to [-1,1] or [0, 1], in line with what model expects.

5.1.2 Building input pipeline

Cifar100 dataset was fetched from Keras library and Imagedatagenerator function was used to generate batches of training data to which augmentation was applied before feeding it to the network.

For self-generated dataset, a test and validation directory were made that were structured like this:

/root directory
    Class a
        a_1
        a_2
        …..
    Class b
        b_1
        b_2
        …...

Upside of maintaining the data tree structure like this was that I was able to get generator to infer labels too. The data is generated as tuple of NumPy array and labels with data type float 32. As original Cifar100 datatype is uint8 and int32, therefore the generated data type was converted to the right type and shape, before utilizing it for training our model.

5.1.3 Building model and loading pre-trained weights

To build our model, few parameters were inspected, to ensure that data flow from one layer to another was correct and model was connected throughout. If the model was disconnected, no gain in training was observed. The parameters were:

- Input shape to pre trained model – this can be chosen during model initialization. Generated data must be the correct shape that was selected when model was instantiated.
- Output shape of pre trained model – this factor decides the input shape to our FC head, therefore output of chosen convolution block

layer must be considered while building FC block. Output shape depends on the input shape and the chosen output layer.

	Input shape	Chosen output layer	Output shape
MobileNet	Batch size, 160, 160,3	Layer 86: conv pw 13 relu	None, 5, 5, 1024
MobileNet	None, None, None,3	Layer 86: conv pw 13 relu	None, None, None, 1024

The extracted features were fed to a densely connected classifier, by flattening them (Global average pool used in case of MobileNet). The densely connected head was defined using batch normalization layer, dropout and dense layers. MobileNet architecture was primed using Model API where functions can be passed as layers, layers are callable, and it allows for multi input and multi output structure. That is why upscaling and data pre-processing layers were added to the model and compiled together.

For the experiment where the model is trained on generated data and tested on Cifar100 test set, the input size is kept None, so that the model could be tested on both 224 x224 images (upscaled Cifar100) and 32x 32 images.

The model parameters differ for each experiment, as the input data size changed (to counter overfitting)

5.1.4 Feature Extraction

As mentioned before, feature extraction [7] consists of using the representations learned by a pre-trained network to extract interesting features from new data. These features are then run through a new classifier, which is trained from scratch.

5.1.4.1 Instantiate

Pre-trained model is first instantiated by specifying weights (in our case – 'ImageNet') , without the FC head (so, include_top = False) and input shape of the image array (or tensor)(160 x 160 for MobileNet).

5.1.4.2 Extract features

Features were stored in a variable by fitting our pre-defined model on training data. Using model.predict, gave the array output of size ?,7,7,512 for VGG and ?,5,5,1024 for MobileNet (here '?' represents the data size). Extraction of features on 5000 samples took 0.18 minutes

5.1.4.3 Create new FC head

New FC head was created using a flatten layer and a dense layer with 512 neurons. It was tweaked and dropout and batch normalization layers were added by monitoring the history of the model, where validation accuracy and validation

loss were plotted against accuracy and loss and the model was adjusted until overfitting was slightly controlled.

Adam was the optimizer chosen for the entire experiment due to its adjustable learning rate.

5.1.4.1 Fit new FC head to extracted features

The features extracted from training and validation data were fitted to the dense network, using 100 epochs with sparse categorical cross-entropy as loss function. The 100 epochs were monitored on basis of validation loss and early stop.

	Trainable parameters	Epochs	Early-stop epoch	Val_loss at training end	Time
MobileNet	0	100	21	1.59091	0.80 m

Note: Feature extractions was brilliantly quick, but the overfitting was very severe. Therefore, this trial was dropped from further experimentation, as without data augmentation, I was not able to control over-fitting.

5.1.5 Fine Tuning

Fine tuning [7] involves running the pre-trained model end to end on the inputs and leverages data augmentation. There is fair warning about this technique being very expensive. Therefore, utilizing GPU for this project was a big boon. Following are the steps followed to achieve fine tuning on our models

5.1.5.1 Freeze layers in conv base

After instantiating the model and adding a new classifier, it is important to freeze convolution base, as this prevents the model's weights from being overwritten by new updates. This step is crucial as not freezing would lead to the pre-trained model forgetting its own learning (the representations it leant on the big dataset will all be lost).

Freezing the model conv base is as simply done as model.trainable = False. When reading model summary in, the trainable parameters for the pre-trained model should be zero.

5.1.5.2 Set up generator for data augmentation

A generator that is used to fetch data from directory or from keras library can be used here again – the only difference is while initializing it we can pass data augmentation parameters to it so that when it is collecting batch , augmentation is applied to them before the batch is fed to the network. Validation data should not be augmented.

5.1.5.3 Fit generator to pre-trained model + FC head

When fitting generator, steps per epoch is a necessary parameter that must be assigned as otherwise the generator would run forever. Again, training was carried out with Adam optimization, over n epochs using early stopping the was based on validation loss. This training run time is noted in minutes (on GPU).

	Trainable / Non trainable parameters	Epochs (n)	Early-stop epoch	Val_loss at training end	Time
MobileNet cifar	578,148/ 3,230,912	70	19	1.99132	4.07 m
MobileNet gene	5,523/ 3,230,912	120	102	1.00662	5.63 m

5.1.5.4 Unfreeze layers in base model

After training of the classifier head, chosen layers in conv base were unfrozen.

	Number of layers unfrozen	Number of total trainable parameters in pre-trained model
MobileNet cifar	24	2,126,848
MobileNet gene	12	1,593,856

Less number of layers were unfrozen in case of generated data, to control trainable parameters numbers, as, the generated data quantity was too low and large model capacity could have led to overfitting.

In case of MobileNet that has batch-normalization layer in its conv base, a parameter 'training' was set as False so that no update would take place on batch norm layers during fine tuning.

5.1.5.5 Fine tune all unfrozen parameters

We fitted our generator to the model again, in its new state but with very low learning rate to control the magnitude of updates that would propagate through the network. Adam was used as optimizer with learning rate of 1e-5 and the model was compiled again before training to set the changes made in the architecture.

	Trainable / Non trainable parameters	Epochs	Early-stop epoch	Val_loss at training end	Time
MobileNet - cifar	2,704,996/ 1,104,064	50	39	1.90328	9.68 m
MobileNet - gene	1,599,379/ 1,637,056	100	100	0.88622	4.92 m

We can notice that fine tuning rendered some tiny precious gain in validation loss and accuracy.

5.2 Test results

Test data was prepared for two experimentation set-ups, when training model with

- Cifar100 training data (so, test data – generated data and cifar test)
- Generated data (test data – cifar test)

Test standards include:

- **Accuracy** on test data – measure of how many correct answers the model was able to predict. It can be simply represented as

  = Number of correct prediction/Total sample size

- **Top 5 accuracy** on test data – it does measure correct answers but in five 5 predictions. More than accuracy, this parameter reflects hoe close the model is getting to the correct answers.
- **Top 10 accuracy** on test data – though unnecessary, top 10 was included to observe if the model's is even slightly closer to its top 5 mark, that is if it considered the features of predicted class closer to the features of actual class. It is just another measure of how far the model threw the dart from bull's eye.
- **Entropy** – It is a measure of uncertainty or surprise in the prediction of variables outcome.

$$H(x) = -\sum p_i \log_2 (p_i)$$

  When the probability is 1 or 0, the surprise in the prediction is nil. There is no uncertainty and we can confidently give the variable's value. As probability gets smaller the surprise gets bigger. In case of a categorical classifier, entropy of a certain class can reflect, classifiers confidence in its prediction. If its class entropy is low, that can be interpreted as classifiers orientation to predict that class as variable's/ event's outcome. Whereas vis-versa is true if the entropy is high (that is classifier favours all classes equally)
- Visualizing incorrect predictions

5.2.1 Performance of sub- architecture – for categorical classification of images

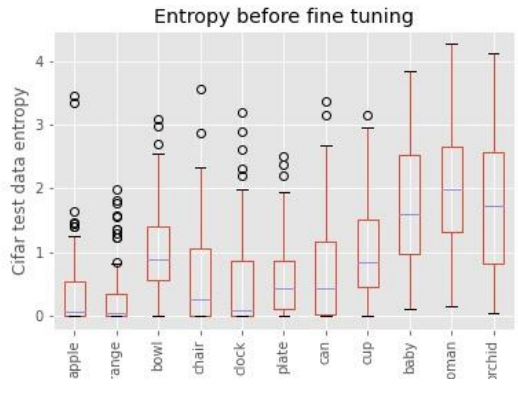
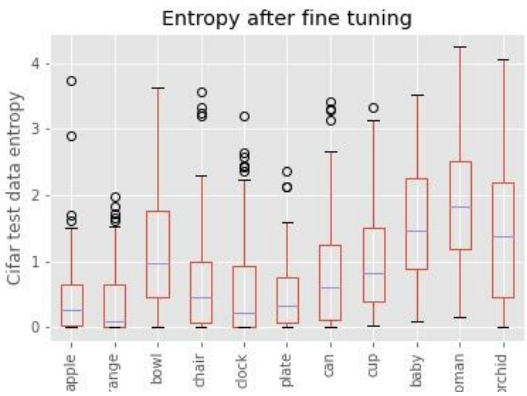
- Accuracy

Feature Extraction	Test data = Generated data			Test data = Cifar100 test set		
	Top 1	Top 5	Top 10	Top 1	Top 5	Top 10
MobileNet-cifar	0.432	0.814	0.89	0.542	0.846	0.916

*Feature extraction was discarded as valid approach due to severe overfitting

Fine Tuning	Test data = Generated data			Test data = Cifar100 test set		
	Top 1	Top 5	Top 10	Top 1	Top 5	Top 10
MobileNet-cifar	0.2590	0.5090	0.6363	0.486	0.799	0.884
ModelNet - gene	-	-	-	0.364	0.84	0.98

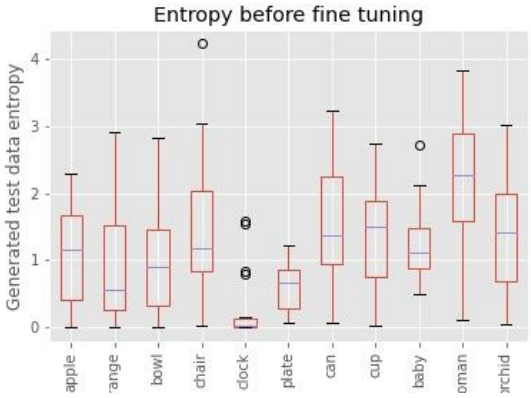
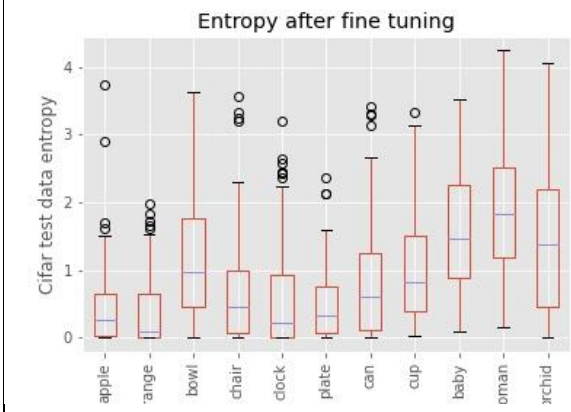
- Entropy – Fine tuning

Training Data set - Cifar100 training data	
Test Data set - Cifar100 test data	
	

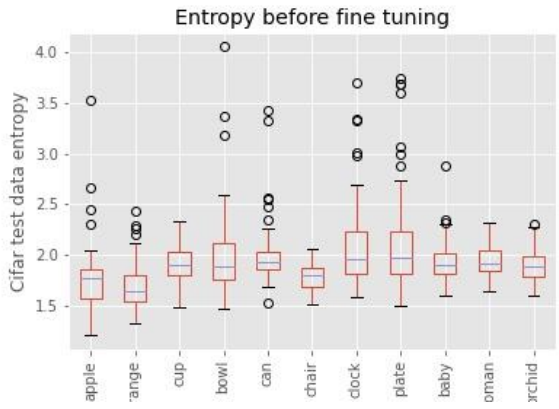
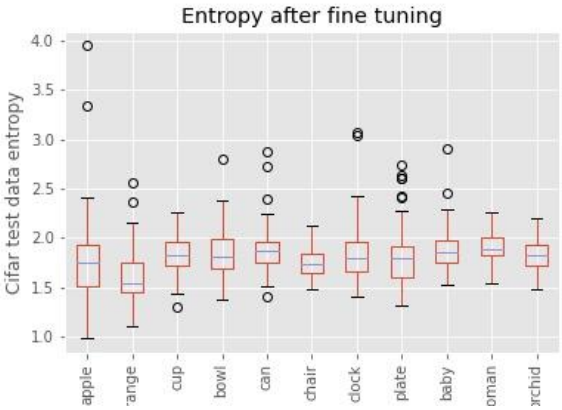
On observing the entropies of classes, chosen on the basis – if they were present in my generated data – we notice that entropies of classes which are already present in the ImageNet – 'apple', 'orange', 'bowl, 'chair', 'clock' and 'plate' are comparatively lower than the classes that are not present in it – 'cup', 'baby', 'woman' and 'orchid'.

Entropy for class like bowl is higher as its abstract features are closer to class plate. Such an observation reflects that our model did pick up the core features rather than train on background noise.

After fine tuning, our model with classifier head has tried to generalise to, cup, baby, woman and orchid. We can see the entropies hare slightly lower.

Training Data set - Cifar100 training data	
Test Data set - Generated test data	
	

The generated test data is inherently higher resolution, that is why is posed a bigger challenge to the fine-tuned model – whose weights have been updated on low resolution Cifar100 dataset. The median entropy for classes has come down, but due to variance in test dataset, there are many images where our model was not at all sure what it was looking at.

Training Data set - Generated training data	
Test Data set - Cifar100 test data	
	

When trained on high resolution dataset, (the model seemed happy to learn from it – given its history dataset), the unsurety around it's predictions has come down tremendously. It can make same kind of prediction for a given image repeatedly. Though the overall confidence remains low.

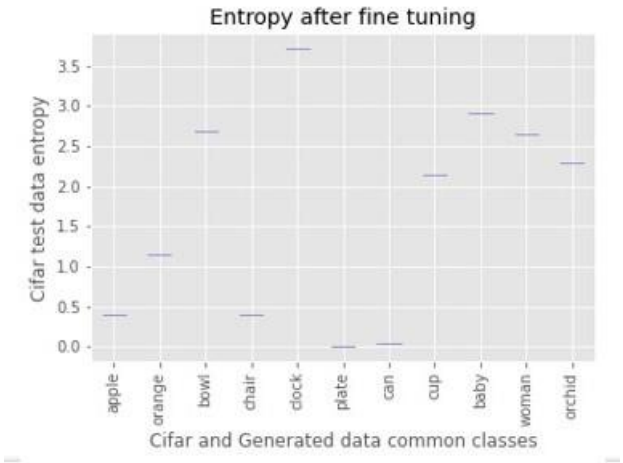
5.2.1 Performance of single input multi output architecture

Trainable parameters = 2,131,456 (mobilnet) + 1,233,898(classifier head)
Training time = 63.05 minutes (before fine tuning) + (for fine tuning)

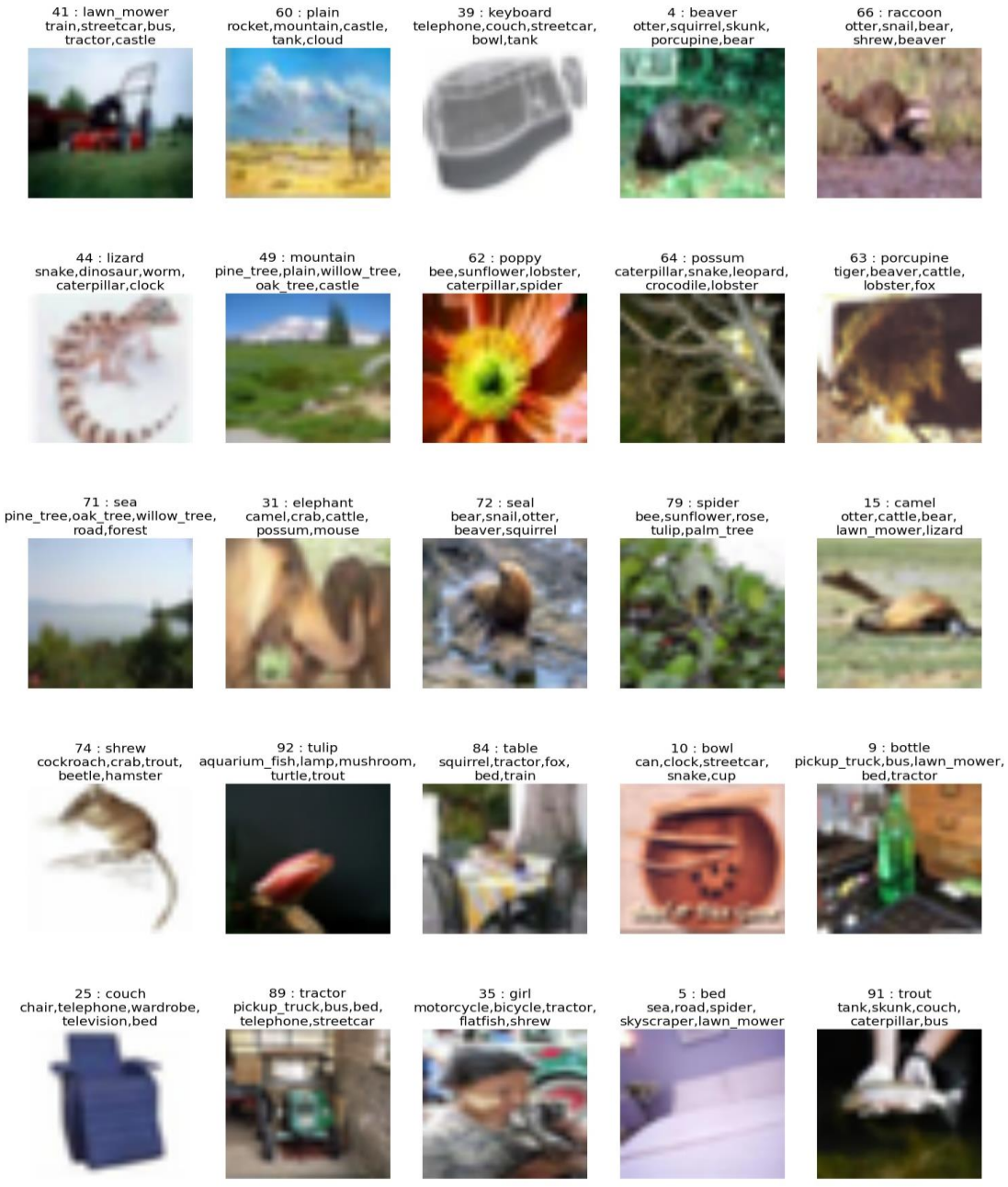
- Accuracy

Fine Tuning	Test data = Cifar100 test set using MobileNet		
	Top 1 – before fine tuning	Top 1 – after fine tuning	Top 5 – after fine tuning
Recycling	0.4112	0.4888	NA
Class	0.5364	0.564	0.8392

- Entropy



- Failed predictions



Analysing failed predictions, it seemed the model did predict an object close to its class in human terms, when it was already familiar with its abstract features. Example – predicting a lawn mover as a tractor – the wrong predictions on classes that it was already familiar with boils down to labels in test set as well. A

background picture with clouds but labelled as plain was predicted as clouds – which is a correct output – but incorrect in terms of what it was labelled as.
This observation stands true for all the classes that are there in ImageNet and Cifar100. The prediction answers reflect its learnings that abstract the images very well.
Same cannot be said for the classes that are not present in ImageNet but are there in Cifar100. A girl or a woman is predicted as bicycle, flatfish or shrew. I infer that this is because base model had no abstractions to start with and even when the learning progressed on the new data, it did not modify the weights enough to accommodate for new data.

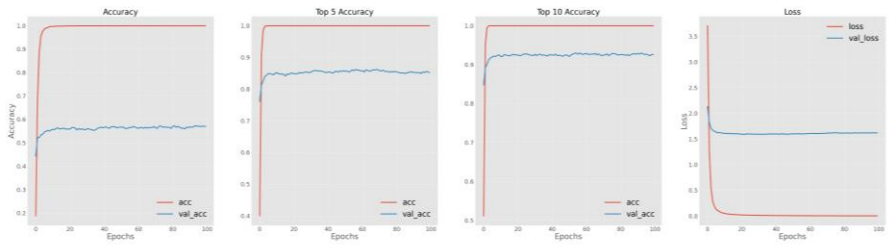
5.3 Learnings

5.3.1 Feature Extraction vs Fine Tuning

Feature Extraction lets us achieve acceptable accuracy, but it was also evident that the model was overfitting with our small dataset, with no method at hand that could resolve this issue. On the positive side, the method is very cheap.

Fine Tuning method is very expensive (if GPU is not employed) but it curbs overfitting as it allows the use of data augmentation

Feature Extraction - Evident overfitting



Fine Tuning – No overfitting

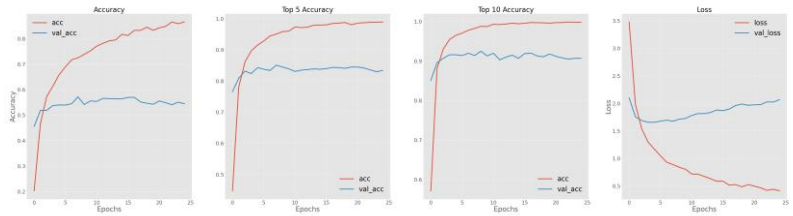
History plot of MobileNet model fine-tuned on Cifar100 dataset

MobileNet fine tuned model history plots

5.3.2 Heavy Augmentation vs relevant augmentation

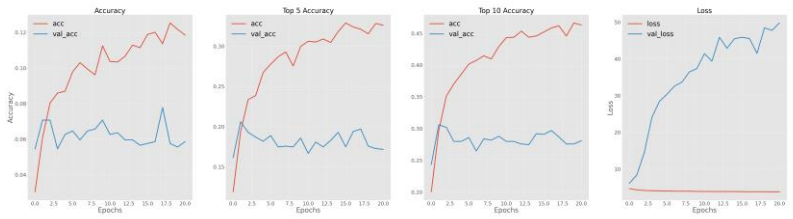
Use of non – relevant data augmentations like vertical flip and zca whitening, rescaling adversely affected the leaning transfer. Instead of boosting accuracy, it hampered it. Use of random, realistic augmentations not only helped fit the model properly to parameter space but also improved accuracy.

No augmentation – history plots



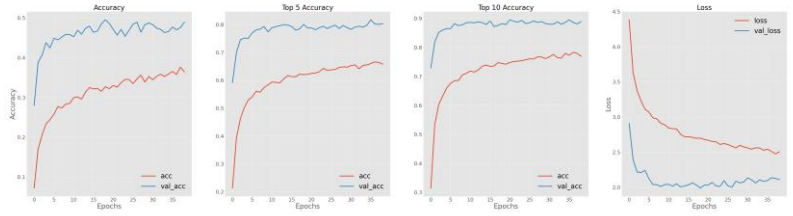
Evident overfitting on cifardata

Heavy augmentation – history plots



Overfitting and very hindered learning on cifar data

Relevant augmentation – history plots



No overfitting , heavy regularization on cifar data

Relevant data augmentation chosen were - rotation range, height shift range, zoom range, fill mode, brightness range, channel shift range, horizontal flip.

Another point to notice is that one must not employ both rescaling in data augmentation and pre-processing function of the model. It is either one – as the purpose of rescaling is to clip the range of pixels between [0-1] or [-1, 1]. Function pre-process of the model includes this step. Employing both would mean normalizing the values twice.

Lastly as data augmentation layers are only active during training and are inactive in inference mode, therefore validation set must not be augmented.

5.3.3 Choice of correct data up sampling

Too much up sampling of Cifar100 data, or too little, hampered the validation loss while training the model. 160 x160 resolution worked well in case of fine tuning , while 224 x224 upscaling of data for feature extraction gave higher accuracy.

MobileNet	Epoch	Validation Loss
224 x 224	28	1.99781
160 x 160	19	1.99132
128 x 128	22	2.19313

5.3.4 Test data handling

Fine Tuning: No pre-processing was applied to test data, as pre-processing severely affected the test results.

	Pre-processed test data			Not processed		
Fine tuning	Top 1 accuracy	Top 5 accuracy	Top 10 accuracy	Top 1 accuracy	Top 5 accuracy	Top 10 accuracy
MobileNet	0.0	0.06	0.08	0.47	0.79	0.87

Feature Extraction: Opposite is true for feature extraction, as dismay able results were notices unless all sets (training, validation and test set were pre-processed before extracting features)

5.3.5 Effects of other regularisations and managing model's capacity

Small dataset training is prone to overfitting especially with pre-trained model's million parameters. This aspect of the experiment was controlled:

- By dropout
- Using correct batch size
- Data augmentation
- Choice of correct number of layers to fine tune

Not only overfitting was cubed by using rigorous dropout but employing these layers also boosted accuracy on validation set.

6 Experiments' evaluation and further work

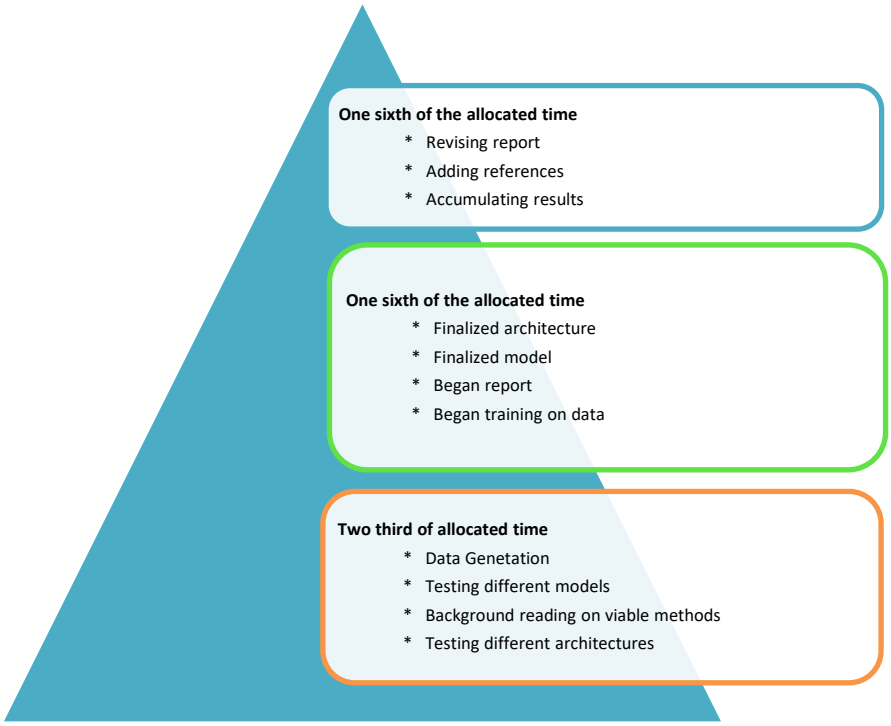
Experiment variables included:

- Pre-trained models – MobileNet
- Train data – Cifar100, generated data
- Methods – Feature Extraction, Fine tuning

The result combinations include:

- MobileNet's performance when trained on Cifar100 data
- MobileNet's performance when trained on generated data

6.1 Project's timeline



Two third for the allocated time:

- **Data Generation** – before it could commence, I tallied ImageNet and cifar100 to decide most appropriate classes that can be divided into recycling and non-recycling items and can also represent the overlap on the two data sets.

- **Testing different model** – Keras provide 9 major models to base one's experimentation on. My initial choice was Resnet152V, due to my

inclination for residual architecture – but due to parameter space, my system would run of memory too often. I Initially tested a basic data set on VGG16 and Resnet50 as well, to capture computation time on these models.

- **Testing different architectures** – The project started with single input and single output architecture and for initial experimentation, I used a multi- hot vector to train and evaluate results, but it became slightly difficult to segregate learning for two outputs and made me wonder if weight updates for classes through single branch might be held back (or not converge to better minima) due to recycling based weight updates. So, new experiment was started to explore single input, multi output architectures. I again tested VGG16 to on this architecture to note any compilation difficulties. It was a success. So I took it further to MobileNet, due to very low number of parameters. But was faced with very bad results even on easy dataset. Further reading suggested, it was due to updates on batch-normalization layer that must be kept in inference mode throughout training. VGG16 does not have batch-normalization layers.

- Background reading – To approach this project, I relied on F. Chollet's Deep learning book, Keras and tensorflow documentation to understand Transfer Learning and data augmentation.

The initial study to understand the building blocks of the experiment occupied the first two months for the project.

Next stage

- Single input multi output architecture for finalized for the experiment in combination with Mobilenet model
- Data sets were processed and readied for training and evaluation.
- I started my report to pen down my motivation and process that I was following
- Training of the model began – that took most time of this phase as I was met with several bottlenecks on the way:
    - Memory errors
    - Different data sizes
    - Importing relevant modules as they many are not well supported by Keras yet
    - Architecture building as putting layers together became a challenge with different data sizes – which let to broken graphs
    - Very low accuracy, due to overfitting and wrong learning rates at different points in the training
    - NAN loss
    - CuDNN failing to initialize

- Dealing with special layer in the model

Last Stage

- Accumulating results – Now the model had matured, and I could run different data sizes and data sets through it.
- Honed my report

6.2 Further work

The constraints that rested on this experiment due to memory and computation cost and that can be valid candidate for further work, to test model's robustness includes:

Training

- Experimenting both training sets with higher/ better models like Inception and Efficient nets
- Accumulating more generated train data
- Accumulating generated train and test data for classes that are neither in Cifar100 and ImageNet
- Accumulating generated train and test data for classes that only in ImageNet
- Experiment with more than two output branches in my architecture
- Build model on pre- trained weights different than ImageNet
- Pre – train my own model on data more relevant to my cause.

Testing

- Testing ImageNet weights on other datasets like Stanford, flowers etc
- Evaluate model on other parameters than entropy and accuracy

Further exploration

- I would like to take this experiment further to object detection in an image and image segmentation

7 Professional Issues – Ethics of using open source code

Ethics for different domain may carry different conduct guidelines, but the underlying logic and essence and 'why someone must behave in a certain way' are elementary and so, common to all aspects of life – at home or at professional front. Why I need to reiterate such a common knowledge, is because making a very straightforward concept difficult, is inherent in human community and usually rely heavily to what the community's mission was or what were their fundamental beliefs that ushered their quest for a certain goal. But even strong motives like mission and beliefs cannot twist something fundamental – moral principles or obligations (that is a sign of righteous nature), to do good by an individual and do good by the society.

Currently there seems to be a big debate amongst open source society, on giving their code away, especially when their work can potentially be put to sinister practices. Even bigger consequence to accept with open sourcing is that the creator might never know what their code evolved into and where it was implemented. 'Open sourcing' idea is very novel, and the tentacles of its rudimentary philosophies sink deep into honesty. An honest code developed by an optimist coder who expects an honest feedback and hopes that his work be put to honest means! There is so much that can go wrong with this approach and it is right to be apprehensive about if the bargain on the other side of the free published code will be honoured.

Let us get the definition of open source out of the way – in computing community it refers to open source code and allows for a collective effort to modify the code and share it back with the community. A proprietary solution on other hand is buying Windowns 10 from Microsoft, after digging through all the terms and conditions and accepting them (and possibly forfeiting hundred odd rights) before one can use it. I do not have a stand as to which is more vulnerable, as both can be exploited by an individual of fair intellect. But with open source code, there could be real and lasting impact on the world. For example, someone found that that the code they wrote was being used by US immigration customs enforcement (ICE) to help separate parents from their children. Other issues include mass surveillance, protestor suppression, anti- immigrant violence, deployment of weapons and other human right violation. It seems that there will always be two implications of the open source content – one that the coder intended, other that a profiteering or a political institute would plan for it.

With open sourcing a coder has no strings attached to the consequences of the code he writes, but if he realises that the code he/she created was being used to bring harm to or threaten a well being of another individual, I am very sure that developer will have hard time sleeping at night.

One solution stands out – is to levy conditions and ethical terms on open sourcing (which some would argue won't sit right with the mission that open sourcing started with in first place). But as I mentioned before that fundamental cornerstones of humanity, that uphold the definition of being human are worth protecting more than a society's mission to propel technology. One such license is the Hippocratic License [17]. It restricts the software from being used by individuals,

corporations, governments, or other groups for systems or activities that actively and knowingly endanger, harm, or otherwise threaten the physical, mental, economic, or general well-being of underprivileged individuals or groups in violation of the United Nations Universal Declaration of Human Rights License [18] was met by claims that it will not work as its terms were self-contradictory.

Another license is Anti-996 License [19] which requires the company using software under this license to comply with local labour laws.

It is hard to draft a licence that 'restricts 'for something that was designed to be free. Yet there can be efforts made to understand what it truly means by 'harm or act of injustice' in current open source context. As the community evolves so can the acceptable conduct with in it – which can be drip fed as we go further (may be a mentor company that can guide others to respect the code and its intentions)

To conclude, open source ethics and what they mean is very subjective, as its black and white definition does not sit well with people with agendas – so in its most basic form it will always be contentious. Open source can be perceived as ethical because of the freedoms that it promotes but on the other hand the solution published as open source might not be moral. All over the world governments and community in general are still oblivious of what human rights really mean, given that we have been here since 200,000 years – it would be naïvely optimistic, to expect our community to crack open sourcing and define its ethics whilst it only been around for 20 odd years.

8 How to use worksheets

My worksheets are drafted in Jupyter Notebook come with

- Saved weight
- Saved Plots
- Saved Models

To use these elements to recreate my work, a new empty model with same structure as my models in the worksheet must be defined and weights should be loaded in them.

This newly initialized model can then be used to evaluate new data.

Care must ne taken that data is fed as 32 x 32 resolution and no other, as all trained model shave been trained to accept this resolution

For evaluating the main worksheet - worksheet 11. Transfer Learning for recycling predictions, function model.evaluate form library cannot be used as it output (?, 2) list. Where first index is predictions for recycling and second are predictions for classes.

There are more than 6 worksheets with corresponsing 'best weight', 'weights', history', 'model' and 'plots', but due to size constraints only few are uploaded.

Bibliography

[1] U. . Tiankaew, P. . Chunpongthong and V. . Mettanant, "A Food Photography App with Image Recognition for Thai Food," , 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8523925. [Accessed 22 8 2020].

[2] A. C. Reddy, V. . Indragandhi, L. . Ravi and V. . Subramaniyaswamy, "Detection of Cracks and damage in wind turbine blades using artificial intelligence-based image analytics," *Measurement,* vol. 147, no. , p. 106823, 2019.

[3] A. B. Tufail, Q.-N. . Zhang and Y.-K. . Ma, "Binary Classification of Alzheimer Disease using sMRI Imaging modality and Deep Learning.," *arXiv: Computer Vision and Pattern Recognition,* vol. , no. , p. , 2018.

[4] C. . Papageorgiou and T. . Poggio, "A Trainable System for Object Detection," *International Journal of Computer Vision,* vol. 38, no. 1, pp. 15-33, 2000.

[5] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009. [Online].

[6] "Dataset preprocessing," [Online].

[7] F. . Chollet, "Deep Learning with Python," , 2017. [Online]. Available: https://amazon.com/deep-learning-python-francois-chollet/dp/1617294438. [Accessed 22 8 2020].

[8] "ImageNet," [Online].

[9] "CUDA Toolkit documentation," [Online]. Available: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html.

[10] "Tesnsorflow GPU support," [Online]. Available: https://www.tensorflow.org/install/gpu .

[11] A. Z. Karen Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014.

[12] Q. V. L. Mingxing Tan, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *arXiv:1905.11946* , 2019.

[13] M. Z. B. C. D. K. W. W. T. W. M. A. H. A. Andrew G. Howard, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv:1704.04861* , 2017.

[14] A. H. M. Z. A. Z. L.-C. C. Mark Sandler, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *arXiv:1801.04381,* 2018.

[15] "Densely Connected Convolutional Networks," *https://arxiv.org/abs/1608.06993,* 2017.

[16] V. V. S. I. J. S. Z. W. Christian Szegedy, "Rethinking the Inception Architecture for Computer Vision," *arXiv:1512.00567* , 2015.

[17] "Hippocratic License," [Online]. Available: https://firstdonoharm.dev/. .

[18] "United Nations Universal Declaration of Human Rights," [Online]. Available: (https://www.un.org/en/universal-declaration-human-rights/).Hippocratic.

[19] "Anti - 996 License," [Online]. Available: https://github.com/996icu/996.ICU/blob/master/LICENSE .

[20] M. . Kruithof, H. . Bouma, N. M. Fischer and K. . Schutte, "Object recognition using deep convolutional neural networks with complete transfer and partial frozen layers," , 2016. [Online]. Available: https://repository.tudelft.nl/view/tno/uuid:eae4c9a8-7548-4c18-bc7a-d2c287e4fc29. [Accessed 22 8 2020].

[21] A. . Sharma, "Traffic Sign Recognition & Detection using Transfer learning," , 2019. [Online]. Available: http://trap.ncirl.ie/3854. [Accessed 22 8 2020].

[22] U. . Michelucci, "Advanced CNNs and Transfer Learning," , 2019. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-4976-5_4. [Accessed 22 8 2020].

[23] A. K. S. E. Hinton, "ImageNet classification with deep convolutional neural networks," *2012.*

[24] E. R. J. D. Marcel Simon, "ImageNet pre-trained models with batch normalization," 2016.

[25] J. S. a. Q. V. L. G. B. Simon Kornblith∗, "Do Better ImageNet Models Transfer Better?," *IEEE Xplorer.*