

# Machine Learning Engineer Nanodegree

## Capstone Project

Mayank Bhatnagar, Udacity  
May 13<sup>th</sup>, 2019

## I. Definition

*(approx. 1-2 pages)*

### Project Overview

This project is about '**Fine-Grained Categorization**'. The technique is advance compared to object identification problems. We can use such techniques to recognize different species of flowers, birds or any other species. It has wider use in identifying and grouping objects and even evaluating whether we are finding a new species next existing or explored earlier.

Usual steps required to approach such problems are to identify the large data set, qualify the data set to be relevant to the problem in hand. Create a good benchmarking and use that benchmarking to create a suitable algorithm or we can even create a new algorithm. Modify the parameters to fine tune the algorithm. Use the findings to enhance your algorithm.

The information related to this project can be found at:  
<https://www.kaggle.com/slothkong/10-monkey-species>

### Other References

**Fine-Grained Categorization:** <https://vision.cornell.edu/se3/fine-grained-categorization/>  
**Fine-Grained Categorization:**  
[https://www.researchgate.net/publication/301452581\\_Croatian\\_Fish\\_Dataset\\_Fine-grained\\_classification\\_of\\_fish\\_species\\_in\\_their\\_natural\\_habitat](https://www.researchgate.net/publication/301452581_Croatian_Fish_Dataset_Fine-grained_classification_of_fish_species_in_their_natural_habitat)

---

### Problem Statement

I am planning to use CNN to help identify different specifics of monkeys by creating different models. My training and validation data set will help train the model. Once model is trained I wish to check how it is doing against various parameters primarily epochs and batch-size. I would ideally like to play around with number of layers and other model parameters to understand how models are getting mature.

Once perfected, this model can be used to classify different specifics of monkeys or even help identify if a new species of monkey is found which does not match with any of the existing ones.

I would primarily use accuracy of the model to identify how good they are doing.

At a high level, this is how I am trying to work on the problem:

1. Understand and load the data set
2. Create few models ( I created two transfer models and one regular model)
3. Compare the accuracy of the models

### **Expected Solution**

Based on my hypothesis one of the transfer model should achieve higher and consistent accuracy as compared to the model created from scratch. Since transfer models are created based on 1000s of images whereas the new model is trained with very limited data set.

The model could be taken further to test samples, which I am not doing at this time. Which could be explored further.

---

## **Metrics**

I would be tracking following metrics to measure the performance of the model.

**Training Time:** The model created should be efficient and run fast for the data of this model. This measure is important to create a balance between the times taken by the model vs the accuracy it achieves. Given the size of the data set though it may not be very significant.

### **Training Time Definition**

Training Time = Final time after training end - Initial time before training start

$$T_{\text{Training}} = T_{\text{Start}} - T_{\text{End}}$$

**Training / Validation Accuracy:** It is important that we avoid over-fitting and under-fitting of the model. If it over-fitting we would be biased and if it under-fitting model will not perform well. I would be measuring this as one of the key metrics. This model is not fully balanced (-5% - 11% variation). Either I will balance the model to use this metrics or use other like F1 score.

**Training / Validation Loss:** Similar to training / validation accuracy, training / validation loss also helps to understand whether the model is over fitting or under fitting. If the training loss is much lower, than validation loss than the model is over-fitting. If both the values are close to each other that means the model is under fitting.

### **Accuracy Definition**

**Model Accuracy =  $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ True Population}}$**

$\Sigma \text{ True Population} = \Sigma \text{ True positive} + \Sigma \text{ True negative} + \Sigma \text{ False positive} + \Sigma \text{ False negative}$

**True Positives:** The cases in which we predicted YES and the actual output was YES.  
**True Negatives:** The cases in which we predicted NO and the actual output was NO.  
**False Positives:** The cases in which we predicted YES and the actual output was NO.  
**False Negatives:** The cases in which we predicted NO and the actual output was YES.

### Loss Function Definition

Loss function is an important part in artificial neural networks, which is used to measure the inconsistency between predicted value ( $\hat{y}$ ) and actual label ( $y$ ).

Cross Entropy is commonly used in binary classification (labels are assumed to take values 0 or 1) as a loss function, which is computed by:

$$L = -\frac{1}{n} \sum_{i=1}^n [y(i) \log(\hat{y}(i)) + (1 - y(i)) \log(1 - \hat{y}(i))]$$

**Ref:** [https://isaacchanghai.github.io/post/loss\\_functions/](https://isaacchanghai.github.io/post/loss_functions/)

## II. Analysis

(approx. 2-4 pages)

### Data Exploration

I have used Kaggle website to get data set required for this project. Converted the data stored in compressed format to normal format for extraction and processing.

I have explored the existing data. Data has two sections, one for training and other for validation. Each folder contains 10 subfolders corresponding to 10 different monkey species. The data set consists of large number of image files with minimum size of 300 \* 400 Pixels. However, for the model preparation perspective I have reduced the size of the images.

Details of different monkey species is available at [Wikipedia's monkey cladogram](#). Following is the mapping of the folder name, species, common name, train images count and validation images count:



S.No	Label	Latin Name	Common Name	Train Images	Validation Images
1	n0	alouatta_palliata	mantled_howler	105	26
2	n1	erythrocebus_patas	patas_monkey	111	28
3	n2	cacajao_calvus	bald_uakari	110	27

4	n3	macaca_fuscata	japanese_macaque	122	30
5	n4	cebuella_pygmea	pygmy_marmoset	105	26
6	n5	cebus_capucinus	white_headed_capuchin	113	28
7	n6	mico_argentatus	silvery_marmoset	106	26
8	n7	saimiri_sciureus	common_squirrel_monkey	114	28
9	n8	aotus_nigriceps	black_headed_night_monkey	106	27
10	n9	trachypithecus_johnii	nilgiri_langur	106	26

### Characteristics of each data set dimensions

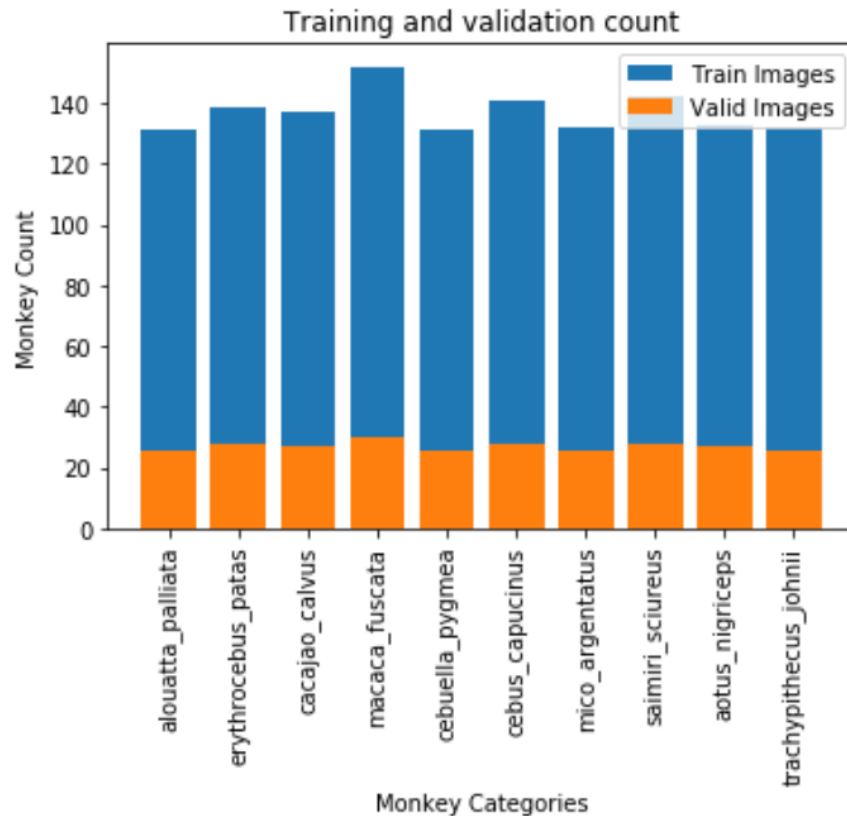
Column Name	Column Description	Type
S. No	Counter	Number
Label	Code for Monkey Categories	Text
Latin Name	Latin name of the Monkey Species	Text
Common Name	Common Name of the Monkey Species	Text
Train Images	Count of training images	Number
Validation Images	Count of validation images	Number

Few examples of the data set is below:

1. Mantled Howler   <i>Alouatta Palliata</i>	2. Patas Monkey   <i>Erythrocebus Patas</i>
	

## Exploratory Visualization

The following diagram depict the training and validation images count in graphical representation. Using below graph we can clearly see that both training and validation data set is consistent across various categories.



- The training data is required to train the model, and validation is to validate the accuracy of the model.
- The data set represented in not balanced for both training and validation image count. The variation is between -5% and 11%. At an overall level, this variation does not seem to be very significant.

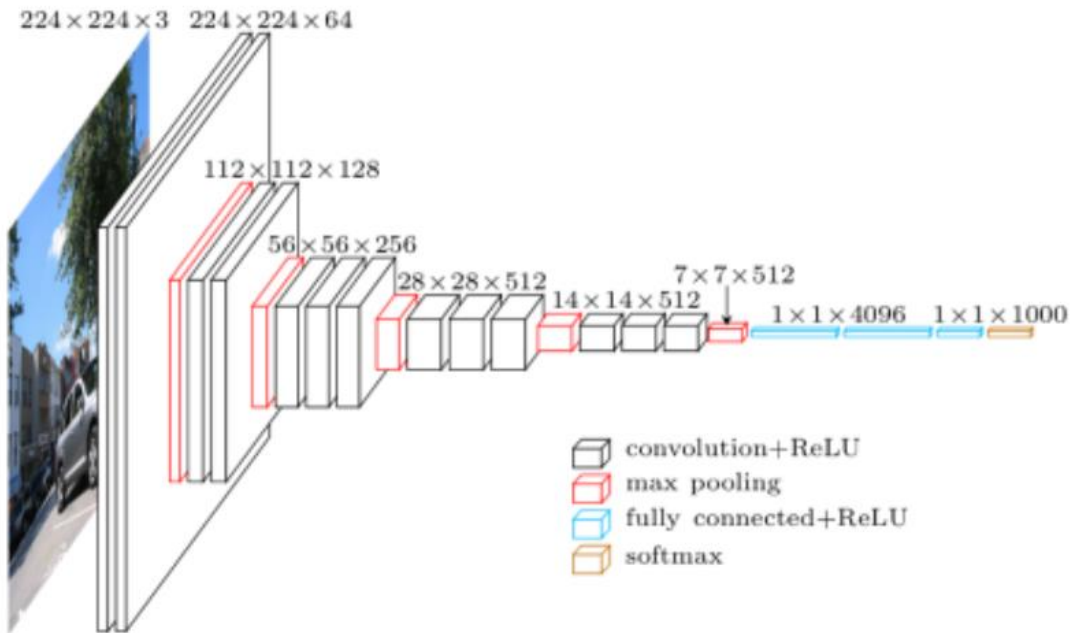
## Algorithms and Techniques

My focus in this exercise was to create a model that provide best accuracy given set of parameters. I learned that ‘Transfer Model’ technique is very efficient to get much better accuracy as these are pre-defined and pre-learned model, which could be further customized for the out-come we are expecting.

I chose ResNet50 and VGG16 models because they are considered one of the best architectures when comes to image classification.

VGG has two versions, VGG16 and VGG19. VGG16 has 16 layers in the network. It has 3X3 convolutions broken by 2X2 pooling layers finished by three fully connected layers.

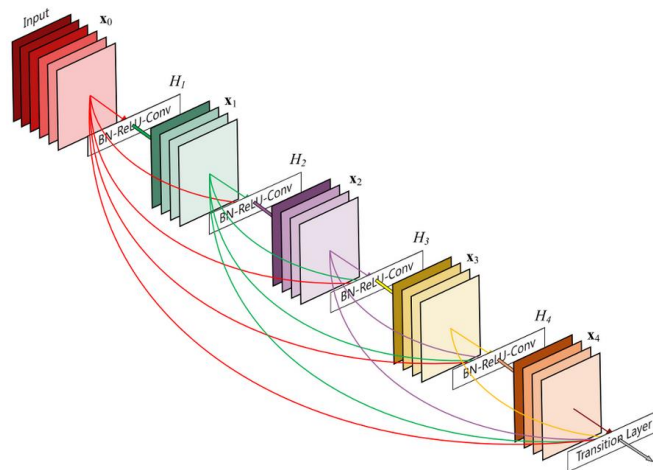
## VGG Architecture



Source: <https://www.cs.toronto.edu/~frossard/post/vgg16/>

ResNet50 is similar to VGG in which layers are repeatedly repeated, and like VGG, ResNET has multiple versions. ResNet due to its architecture has close to human accuracy in classifying images using ImageNet database. The fundamental breakthrough with ResNet was it allowed us to train deep neural networks with 150+ layers successfully. I am using ResNet50 version of ResNet for my program.

## ResNet50 Architecture



Source: Google

I have used ImageNet data-source is used for both the above architectures.

Imagenet is a set of more than 14 million images with more than 10,000 image categories. ImageNet is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet, majority of them are nouns (80,000+). Images of each concept are quality-controlled and human-annotated.

I started with ResNet50 and VGG16 transfer models and one of my own from scratch. It is difficult to show what all was tried before I reached to the algorithm which is finally presented here. It was a combination of changing various parameters and observing the results.

Below is the structure of the model used for the transfer model.

S No	Model Steps	Parameters Used	
1	Create base model	Weights:	Imagenet
		Include Top:	False
		Input Shape:	(224, 224, 3)
		Pooling:	Avg
2	Create a neural network Dense layer	Activation:	Relu
		Number of Nodes:	512
3	Create a drop-out layer with 0.5		
4	Create an neural network Dense layer	Activation:	Relu
		Number of Nodes:	512
5	Create a drop-out layer with 0.5		
6	Finally, create a Dense layer	Activation:	Softmax
		Number of Nodes:	11

- A. I created ResNet50 transfer model using above structure
- B. After the model was created. I suppressed the entire training layer so that the model is not trained again. I realized that the resources needed to retrain such model is huge.
- C. Post this model I compiled using below parameters
  - Optimizer = Rmsprop
  - Loss = Categorical Crossentropy
  - Metrics = Accuracy
- D. Once the model was created, I trained using the training and validation data set created. Details of the data set is explained earlier in the section.
- E. During the training accuracy and loss was stored in following parameters for visualization
  - Accuracy
  - Validation Accuracy
  - Loss
  - Validation Loss
- F. Finally, total test accuracy and time taken was captured for comparison.

Exact same process was repeated for VGG16 transfer model preparation too. Post this a new model was created from scratch.

Created one model from the scratch:

Below is the structure of the model used for the transfer model.

S No	Model Steps	Parameters Used
1	Create convolutional layer	Filters: 16 Kernel Size: 2 Padding: Same Input Shape: (224, 224, 3) Activation: Relu
2	Add max pooling with pool size = 2	
3	Create another convolutional layer	Filters: 32 Kernel Size: 2 Padding: Same Input Shape: (224, 224, 3) Activation: Relu
4	Add max pooling with pool size = 2	
5	Repeat steps 2 and 4 for filters 64 and 128	
6	Create a drop-out layer with 0.3	
7	Create an neural network Dense layer	Activation: Relu Number of Nodes: 256
8	Create a flatten layer	
9	Create a Dense layer	Activation: Relu Number of Nodes: 11
10	Create a drop-out layer with 0.3	
11	Finally, create a Dense layer	Activation: Softmax Number of Nodes: 11

## Benchmark

As stated earlier, I created two transfer models using ResNet50 and VGG16 as the benchmark models to compare accuracy. These models with different hyper-parameters suggested the kind of accuracy could be achieved. The model I created had much less over-all accuracy.

I could have ideally tested the models for higher epochs but did not have the system capacity to do so. Below is the comparative study of our benchmark models vs our newly created model:

### Benchmark Model Accuracy

As expected benchmark models accuracy is much higher than that of newly created model.



	ResNet50		VGG16		New Model	
Epochs ↓ / Batch Size →	16	32	16	32	16	32
10 Epochs	26.84%	94.49%	65.81%	74.26%	41.91%	49.26%
20 Epochs	95.96%	95.59%	81.25%	83.09%	52.21%	53.68%
40 Epochs	95.22%	94.85%	85.29%	85.66%	53.68%	55.51%
Benchmark Model Accuracy %						

### **Benchmark Model Time**

As expected benchmark models are taking much more time to execute due to size of the input parameters.

	ResNet50		VGG16		New Model	
Epochs ↓ / Batch Size →	16	32	16	32	16	32
10 Epochs	170	142	261	230	44	40
20 Epochs	334	284	517	454	86	80
40 Epochs	667	567	1034	908	172	160
Benchmark Model Time						

*\*All measurements are in seconds*

Details of the run is explained in the result section.

## **III. Methodology**

*(Approx. 3-5 pages)*

### **Data Preprocessing**

I have done following transformation on the images before supplying it to the Keras model.

1. A function was created which takes a string-valued file path to a color image as input and returns a 4D tensor
2. The function first loads the image and resizes it to a square image that is 224×224 pixels.
3. Converted image is then transformed into an array, which is then resized to a 4D tensor.
4. Finally, I rescaled the images by dividing every pixel in every image by 255.

Other pre-processing steps like rotating images, making them in grey scale could also be tried to improve the efficiency of the model.

## Implementation

The implementation process was divided into following sections:

1. Understanding input data, and Loading data set
2. Pre-processing of the data
3. Creating benchmark models using ResNet50 and VGG16
4. Creating new model from scratch

During model creation, we can see modularization at its peak. I created set-of three functions to carry out all the activities:

For the ResNet50 model following three functions created, trained and evaluated the model:

1. Def ResNet50\_mod()
2. Def train\_model()
3. Def create\_graph()

For the VGG16 model I reused two of the above functions and created a new VGG16\_mod() function for VGG16 model. Thus, I have:

1. Def ResNet50\_mod()
2. Def train\_model()
3. Def create\_graph()

Finally, for the new regular model I reused two of the existing functions and created a new monkey\_mod() function for CNN model. Thus, I have:

1. Def monkey\_mod()
2. Def train\_model()
3. Def create\_graph()

For each of the model ResNet50 and VGG16 model 'imagenet' weights were used. The models were modified

Layer trainable is set to 'False' so that I should not retrain the model. Even system resources were not enough to re-train the model.

---

## Code Optimization

First, I created different logic for each model implementation. Later I realized that I am repeating a lot of code. I modularized the code to create functions. These functions are generic and parametrized to provide me various variations of the algorithms.

Function Definition 1	def train_model (model, epochs_val, batch_val)	
Function Input	Model	
	Epochs_Val	
	Batch_Val	
Function Output	Acc	Training Accuracy
	Loss	Training Loss
	Val_Acc	Validation Accuracy
	Val_Loss	Validation Loss
	Test_Accu	Testing Accuracy

Function Definition 2	def create_graph ( epochs_val, batch_val, acc, val_acc, loss, val_loss, test_acc, time_taken)	
Function Input	Epochs_Val	Number of Epochs
	Batch_Val	Number of Batches
	Acc	Training Accuracy
	Loss	Training Loss
	Val_Acc	Validation Accuracy
	Val_Loss	Validation Loss
	Test_Accu	Testing Accuracy
Function Output	None	No Output: Graphs were printed

## Sample Function

```
## Plotting all the graphs together as a generic function
def create_graph(epochs_val, batch_val, acc, val_acc, loss, val_loss, test_acc):
    x = 0
    ## Iterate for all the values for each epochs
    for e_val in epochs_val:
        for b_val in batch_val:
            plt.title('Training and validation accuracy: Epoch (' + str(e_val) + '), Batch Size (' + str(b_val) + ')')
            plt.plot(range(1,e_val+1), acc[x], 'red', label='Training acc')
            plt.plot(range(1,e_val+1), val_acc[x], 'blue', label='Validation acc')
            plt.legend()

            plt.figure()
            plt.title('Training and validation loss: Epoch (' + str(e_val) + '), Batch Size (' + str(b_val) + ')')
            plt.plot(range(1,e_val+1), loss[x], 'red', label='Training loss')
            plt.plot(range(1,e_val+1), val_loss[x], 'blue', label='Validation loss')
            plt.legend()
            plt.show()

            ## Printing accuracy information for each batch run
            print('Test Accuracy: %.4f%%' % test_accu[x])

            x += 1
```

## Refinement

From the time I started on this project and until its completion I did various improvements. To name a few:

1. Changed parameters like Epochs and Batch Size to understand
2. Modularized the code

I experimented with **Epochs** and **Batch size** to understand which configuration gives better and efficient model.

**Epochs:** It is the number of time the complete data set goes back and forth through the neural networks. As the number of epochs increases, more number of times the weight are changed in the neural network and the curve goes from under-fitting to optimal to overfitting curve.

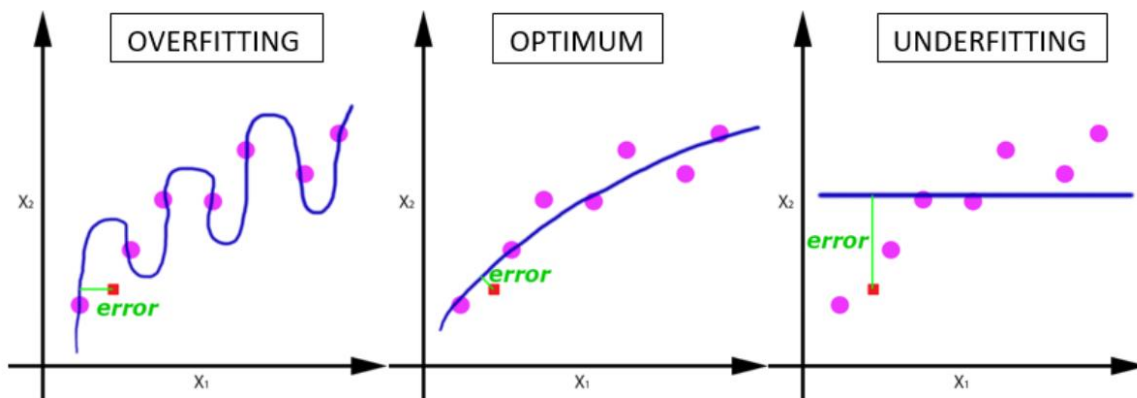


Fig Ref: <https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>

**Batch Size:** It is the size of the data set processed through the Epochs. If the entire dataset is divided into 10 batches, then it will take 10 iterations for the complete data set to go through one epoch.

I worked with two different transfer models and here are the results. If we analyze the results, we could find that –

1. ResNet50 model achieves higher accuracy very fast at lower epochs
2. VGG16 model has consistent increase in accuracy per epochs, but didn't reach very high
3. The model I created from scratch wasn't very consistent, probably due to the size of the data

There were certain steps taken to structure and modularize the code. Further details on that is explained in the above section 'Code Optimization'.

## IV. Results

(approx. 2-3 pages)

### Model Evaluation and Validation

I selected ResNet50 transfer model as the final model due to the highest accuracy in the results. Given the accuracy results from the various models, I would prefer model with Epochs of 40, and Batch Size of 32. It provides most optimal model in most of the cases, though the difference is not much.

Ideally, we could have increased the number of epochs, but I could not do it due to machine performance limitations. Here are the results of the various model testing I did:

#### Model - Training Time

Epochs ↓ / Batch Size →	ResNet50		VGG16		New Model	
	16	32	16	32	16	32
10 Epochs	170	142	261	230	44	40
20 Epochs	334	284	517	454	86	80
40 Epochs	667	567	1034	908	172	160

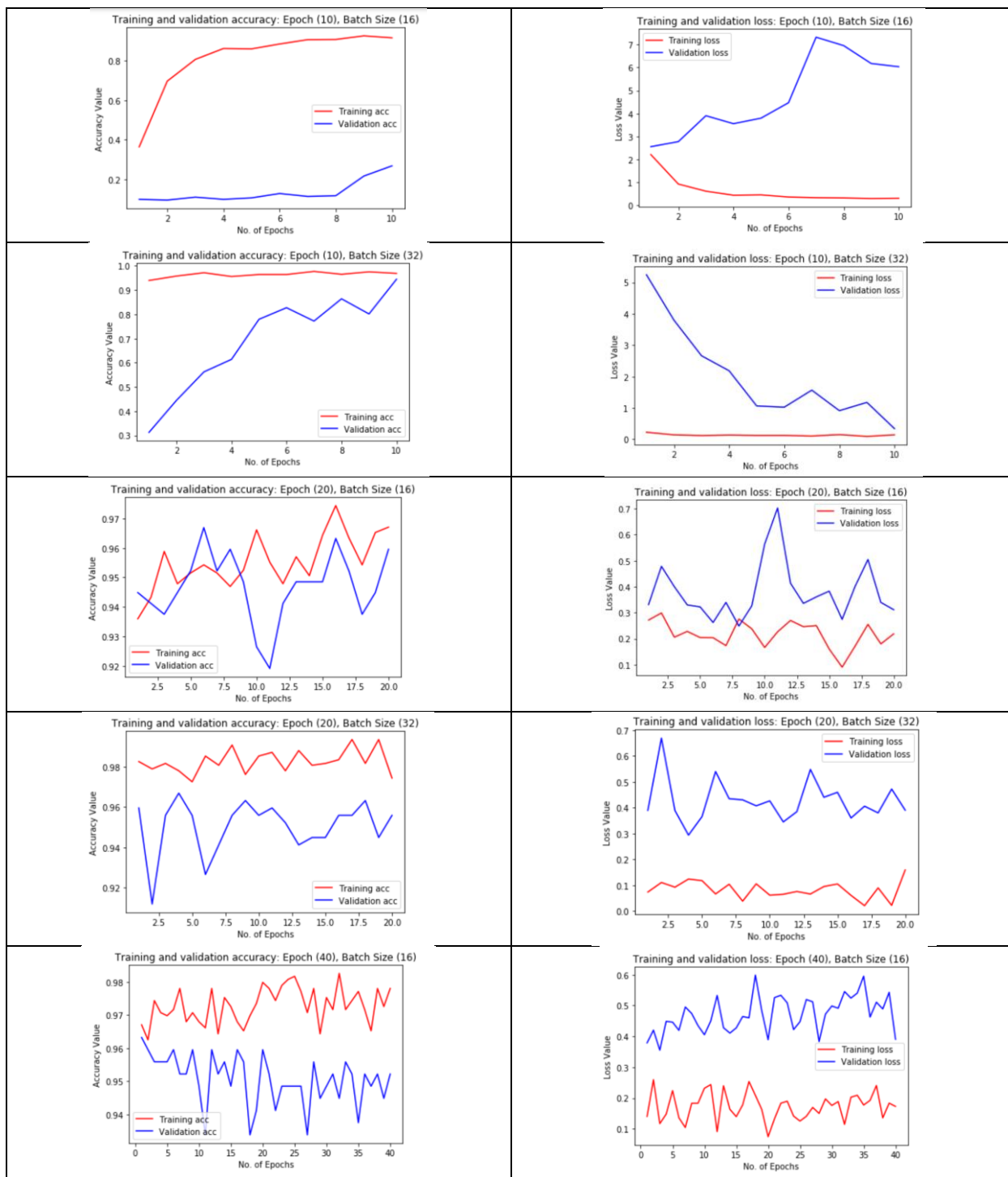
*\*All measurements are in seconds*

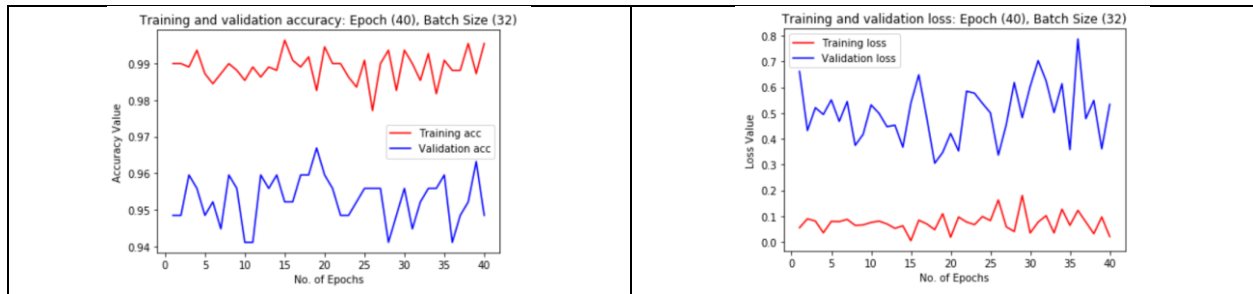
Based on the time taken by the models it is evident that batch-size of 32 takes relatively less time to train the model. Thus, our ideal batch-size between 16 and 32 should be 32. Comparing different models, VGG16 model is taking more time whereas New Model is taking least time. New Model taking least time is evident due to the number of data set it goes through, whereas other two models go through 1000s of images.

Timings are not core conclusive data points until a particular set starts taking proportionately very high time. It does provides a good view of the GPU time system is taking.

#### ResNet50 Model - Accuracy

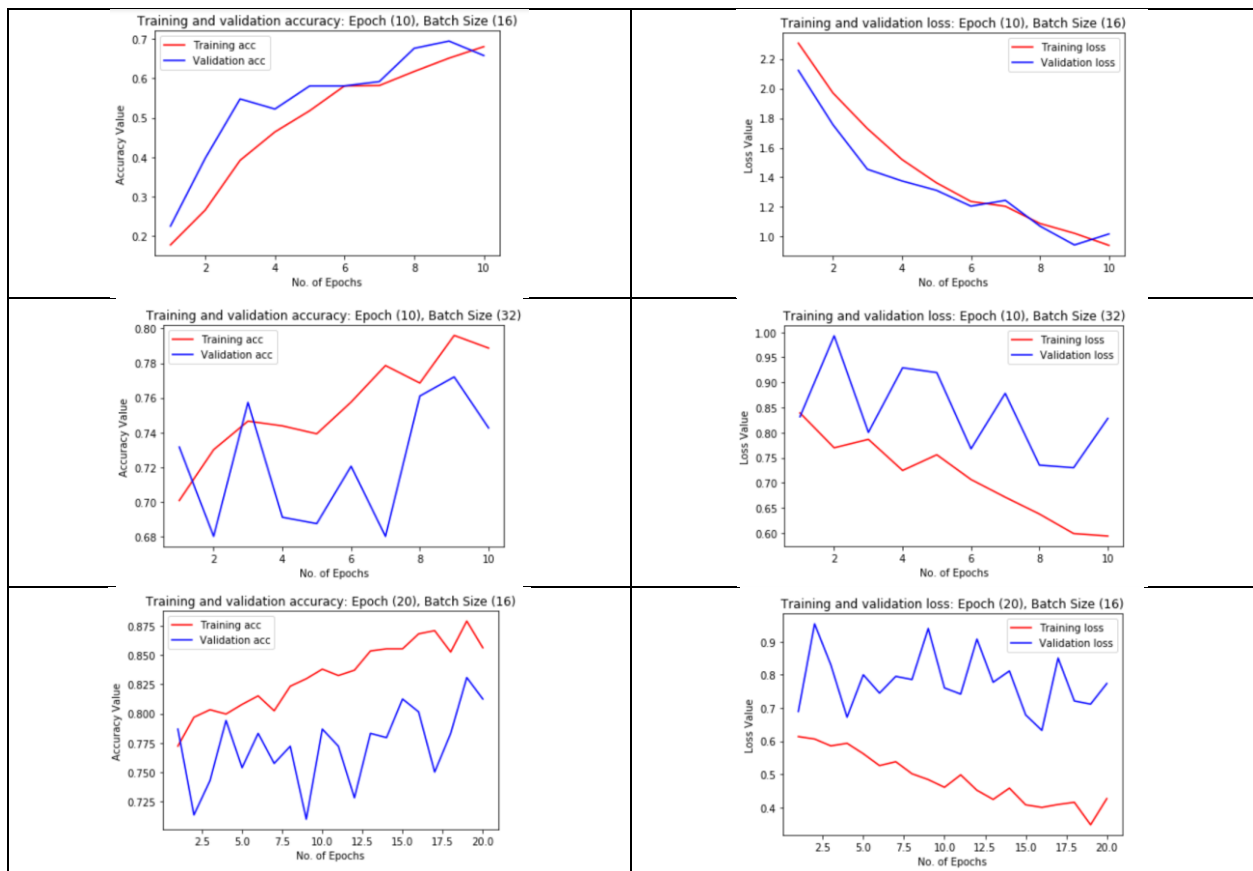
Epochs ↓ / Batch Size →	16	32
10 Epochs	26.84%	94.49%
20 Epochs	95.96%	95.59%
40 Epochs	95.22%	94.85%

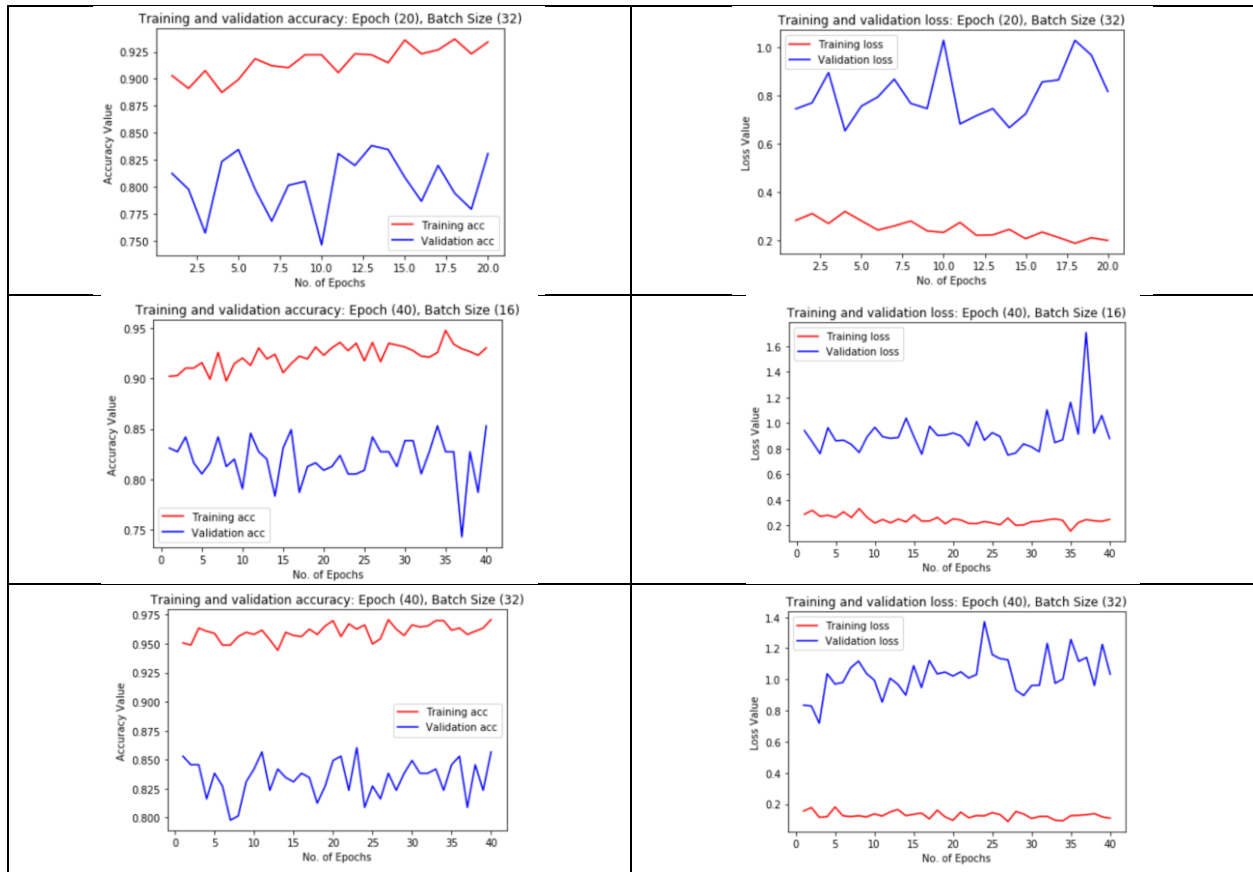




## VGG16 Model - Accuracy

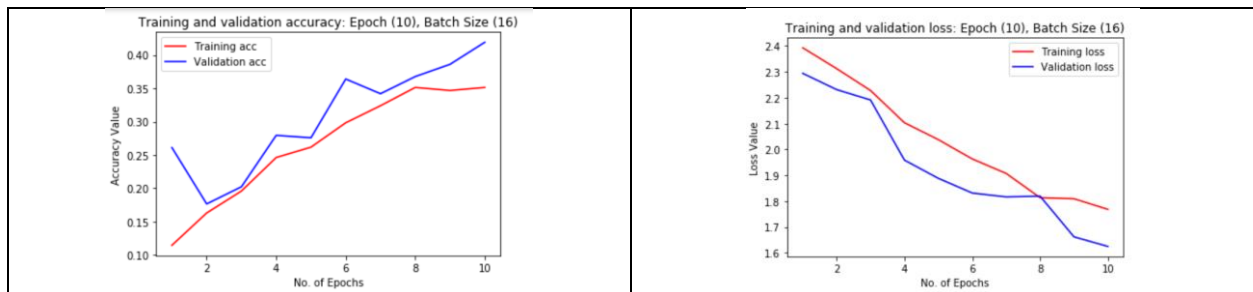
Epochs ↓ / Batch Size →	16	32
10 Epochs	65.81%	74.26%
20 Epochs	81.25%	83.09%
40 Epochs	85.29%	85.66%



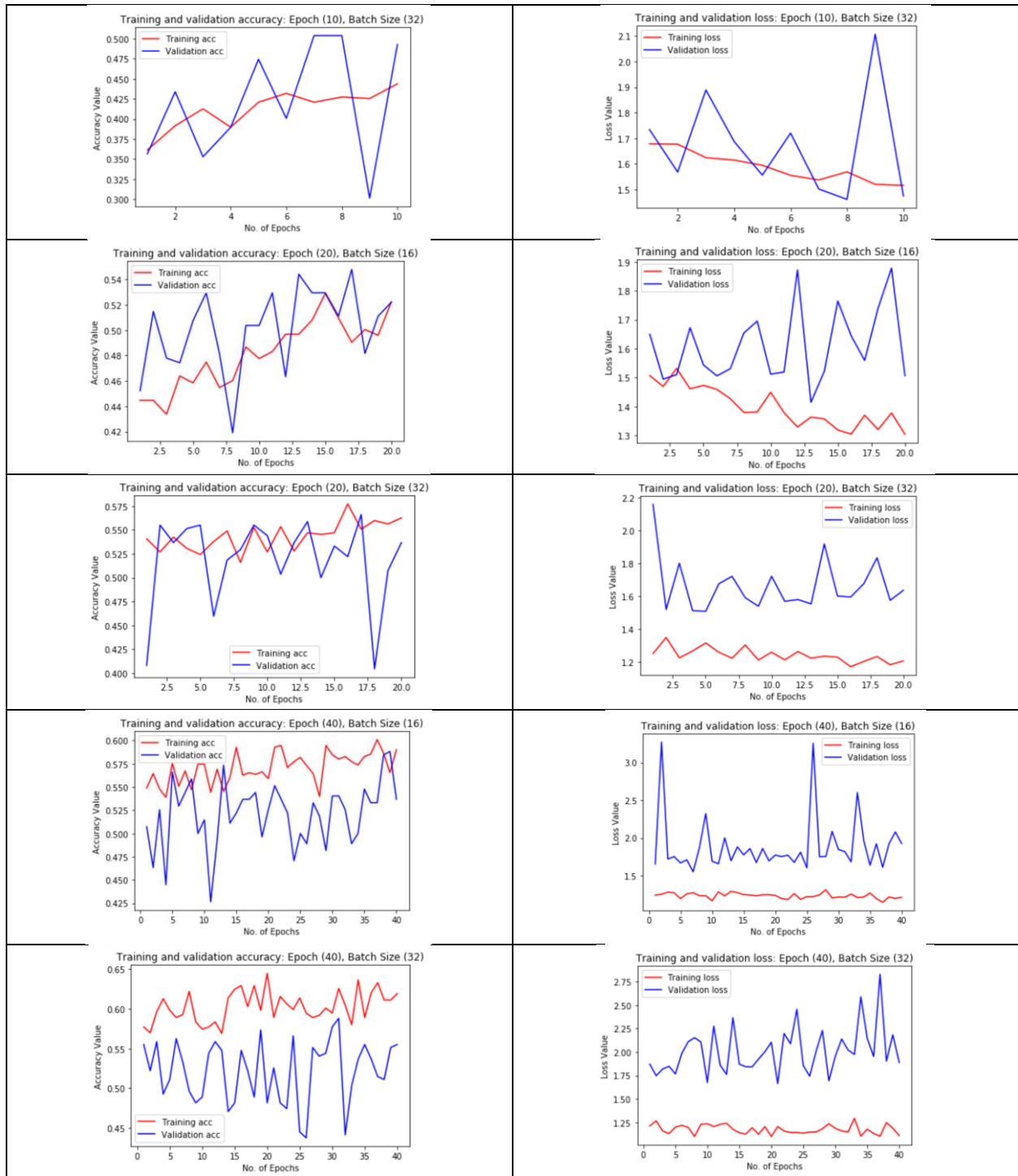


## New CNN Model - Accuracy

Epochs ↓ / Batch Size →	16	32
10 Epochs	41.91%	49.26%
20 Epochs	52.21%	53.68%
40 Epochs	53.68%	55.51%







## Justification

I used two transfer models and one self-built model. The results of the **ResNet50** transfer models were much superior and consistent compared to any other model. However, I see **VGG16** transfer model improvements seems to be exciting.

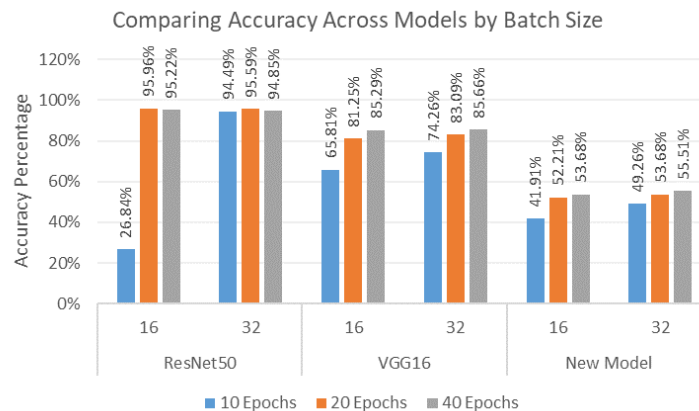
## V. Conclusion

(approx. 1-2 pages)

### Free-Form Visualization

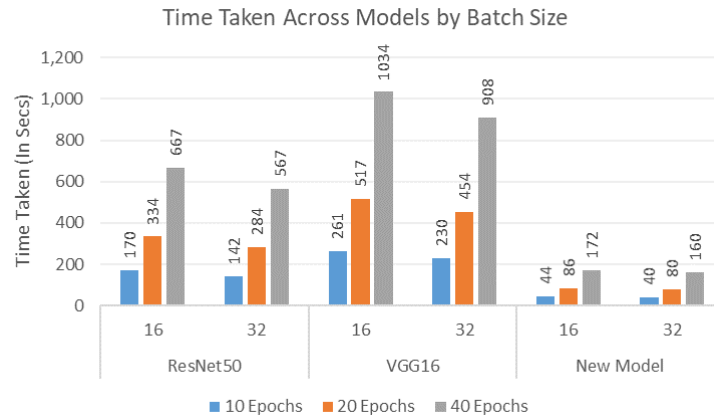
So far we looked into how accuracy moved as epochs increased for a given model training for a particular epoch and batch size. When I do comparative analysis of accuracy across epochs and batch sizes we could see that ResNet50 has increased to very high accuracy, whereas VGG16 progressed gradually. Though the number of epochs were not enough to make any conclusion, it is still an interesting trend.

Ideally, we could continue doing this with increased epochs and see at what epoch count VGG16 accuracy starts matching or beating that of ResNet50. The new model created was lacking the data-set impacting its overall metrics.



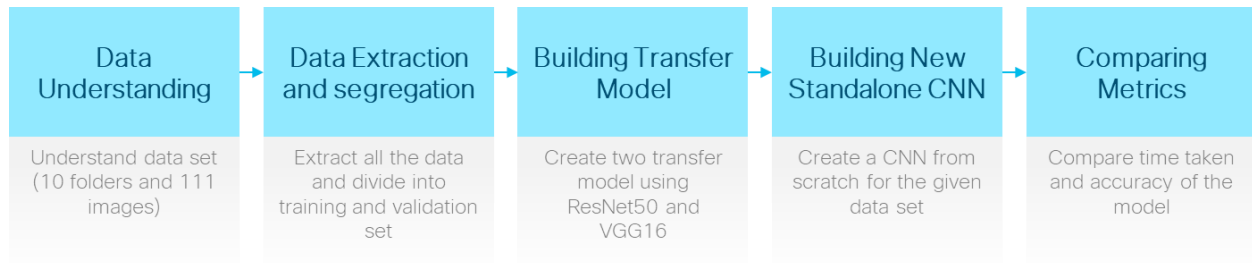
I further tried to analyze what we could make out of the time taken by different models and the settings. As mentioned in the results, batch-size of 32 takes relatively less time compared to batch-size 16, but overall ResNet50 model looks more promising compared to other models.

Similar to Accuracy findings, we need more processing to evaluate the best trained model.



## Reflection

The complete solution can be divided into five big buckets



1. **Data Understanding:** Collecting large set of base data and understanding its structure. This step was important as future model architecture was based on this.
2. **Data Extraction and Segregation:** Dividing it into training and validation sets. This step was important and required to have balanced data set to tune the model.
3. **Building Transfer Model:** Transfer model is a technique to have an efficient model using pre-existing large data set. I used ResNet50 and VGG16 Keras pre-existing models to create transfer models.
4. **Building New Standalone CNN:** After creating transfer model using pre-existing data set, I created a standalone CNN to check the accuracy of this model. As expected, the accuracy of this model was much less than that of Transfer Models.
5. **Comparing Metrics:** I compared all the three models. Details of it are listed in previous section.

I feel either of ResNet50 and VGG16 Transfer model could be used for the real life training and validation. ResNet50 reached high accuracy very fast whereas VGG16 was showing gradual improvements. My laptop resources were limited to check it completely. I would recommend training the model with higher epochs and more training and validation data. This could have increased the accuracy.

## **Improvement**

I completed this project using Udacity GPU set-up, which was on the cloud. I tried with Kaggle GPU, but could not set-it-up properly. Since my Udacity GPU was giving me required out-put I did not look back on Kaggle. I indeed spent significant effort to sort this. If I had a better hardware on my machine or easier software set-up (on cloud), I believe I would have spent quality time on refining algorithm.

Apart from hardware and software challenges, following improvements in the implementation and approach could have resulted in a better solution:

1. I created training and validation data set in file folders. I could have designed data load using zip files to save effort to load file. I did not spend time on this due to limited bandwidth.
  2. The code could be modularized to check the efficiency of more Keras models. I tried VGG16 and ResNet50. By making models parametrized, we can fetch the proper records.
  3. I did explore few parameters, given more time I would have explored more and check how the models are doing in different settings.
  4. I do believe we could create a better model than I have created one here. I had limited knowledge of various hyper parameters, else, could explore more.
-