# Concept Generation and Analysis

## Product Development

This product will be a combination of several different technologies and applications. COS will be divided into three major parts. The first part of COS will be the user facing portion, that will allow health care workers to use the model. The second portion of COS will be the data storage aspect. And the third part will be the ML model server.

The user facing portion of the COS (UFP-COS) will consist of an ASP.NET MVC based application that health care professionals can log into, and interact with patient data. It will have a MS SQL DB that it will use to store all user, and patient profile information. Where as the scan files will be stored in a object storage provider, and be distributed through a content delivery network (CDN), with in Canada.

The data storage portion of the COS (DS-COS) will be responsible to store the files given to it, through the UFP-COS. We choose this method of object storage since, file serialization to store images in a database is not feasible at scale, especially when the number of images that can be stored is not capped. Another aspect to DS-COS is the fact that the files that are given to UFP-COS will not be in an image format but in the DICOM file format.

The final part of COS is the machine learning aspect (ML-COS). COS will not in itself train models, but rather run data through our pre-trained model in a process known as inference. This will take place on another server which will be running “TensorFlow Serving” Which is then stored on DS-COS and then served to the medical practitioner through UFP-COS.

## Model Development

Introduction

When looking into developing our model we identified the three key chooses, framework selection, hardware selection, and network selection. We identified framework selection as a key choose since, it would act as the key building blocks in constructing the models, and effect how fast we could train them. Where as with hardware selection was important since it would be the primary limiting factor in how fast we could train the model. And finally Network selection would be the most important, since it would greatly effect how effective our model would be after we trained it.

Framework Selection

Due to the exponential expansion of ML research and computing power seen over the last decade. There has also been an explosion of new types of software infrastructure to harness it. This software has come from both academic and commercial sources. The need for this infrastructure arises from the fact that there needs to be a bridge between theory and application. When we looked at what were the most popular frameworks, we found it was a mix of strictly academic and commercial driven software. The four main frameworks were Caffe, Theano, Caffe2 + PyTorch, and Tensor Flow.

When we went about choosing a framework we considered three different factors, community, language, and performance. Community was one the biggest factors, since none of us had real production experience in doing any sort of large scale ML modeling and deployment. The only framework that fulfilled this need was Google’s Tensor Flow. It had been released in 2015 and had been widely open sourced. Leading to many academic researchers to contribute to it. And many smaller companies to use it in their production product pipelines. The combination of both engineers and scientists using it has lead to a lot of community driven decides being made to TF, making it easier to use and deploy. The by product of which is large amounts of documentation written by the community, and a large amount of personal and company blogs, detailing how they used TF to accomplish their goals. The only real competitor at the time of writing it this is Facebook’s PyTorch Library which was just open sourced early this year.

The other factor was the language interface it would use. Since only one person on the team had experience programming in a professional capacity, we wanted an easy to use interface, with which to build out the model. When we looked at what available we found that all of the popular frameworks were written in C++ and CUDA, but had a easy to use Python based interface. The only framework out of the four mentioned above, that only had C++ based interface was Caffe.

The most important part of framework selection was the performance aspect. Most if not all ML research and production use cases happen on Nvidia GPU hardware. This is due to Nvidia’s development of their CUDA programming language for use with their GPUs. It makes parallel programming for their GPUs incredibly easy. This parallelization is what lets the complex matrix operations be computed with incredible speed. There were only two frameworks out of the four we mentioned, that used CUDA its code base. They were Tensor Flow and PyTorch, however PyTorch was not as robust as Tensor Flow in supporting the different version of CUDA.

In the end we choose to go with Tensor Flow since it had a better community and CUDA support. We did not choose to go with its nearest competitor, since it was not as well documented, and its community had just started to grow. Where as Tensor Flow has been thoroughly documented and has had large deployments outside of Google (such as at places like Linkedin, Intel, IBM, and UBER). Another major selling point for Tensor Flow is the fact that, it is free, continually getting newer releases, and is fast becoming an industry standard tool.

A summary of our finding can be seen below.

Hardware Selection

CUDA Vs Open CL

In today's high performance computing world there has been a large shift in what components drive the performance of a system. Traditionally the most important part was the central processing unit (CPU), however as Intel and AMD have hit the limits of Moores law. A new technology continues to drive performance gains, that technology being called graphics processing units (GPU). The fundamental difference being that GPUs have thousands of small weak cores, whereas CPUs have a few dozen cores large strong cores. And the way that programs are executed on each of there units. The GPU is designed for highly parallelized computing tasks. Whereas CPUs designed for very linear tasks.

Since the majority of high performance computing deals with numerical approximations, that involve the vectorization of systems of equations. A computing problem can then be broken down into thousands of smaller tasks (such as addition and multiplication) that can be spread across all the cores. However with CPUs the few dozen core get statured with tasks, that leave the other parts of the computation in a queue waiting to be run.

The global leader in GPU technology is Nvidia. Originally their goal was to develop external cards, that could be used render video game geometry with increasing speed and complexity. However in the last decade their focus has now shifted to the development of GPUs in HPC applications. Since there as been an explosion in deep learning and ai research garnered by large technology companies like Amazon, Facebook, and Google.

The adoption of their GPUs as the deep learning standard was made possible by initial release of their CUDA framework, which researcher could use to accelerate there high performance computing tasks. The introduction of CUDA meant that the individuals that used it did not need to know how write multi threaded code. On top of CUDA, Nvidia also developed a large number of highly optimized libraries to do most common commuting tasks (such as matrix multiplication). Which is why a large number of deep learning frameworks run exclusively on Nvidia GPUs. A competitor to Nvidia in the GPU market has been AMD, which has tried to follow Nvidia into the high performance GPU market. AMD has adopted the OpenCL framework that solves the same problems as CUDA, however it is heterogenous to the type of processor it can use. Open CL is open source framework that isn't just for use in any GPU (including Nvidia GPUs), but CPUs as well. As of late, Open CL has yet to gain any traction in the high performance computing (HPC) world. Since it is not as fast as CUDA, and the current machine learning libraries do not fully support it.

Therefore we have chosen to work exclusively with Nvidia based GPUs running the latest CUDA drivers, with the newest version of Tensor Flow.

On Premises vs Cloud

In the initial conception of the idea for this product we were planning to primarily use cloud VMs to train the models. However, after doing more research we found that the cost to benefit ratio did not workout in favour of using cloud VMs. We looked at three major areas when doing our analysis, cost, performance, and maintainability. And compared the following options, AWS, Azure, GCP, and On Premise solution.

As we looked at the cost component, and we found that a lot of VM types offered for by the cloud providers were variably priced, expect in the cases of entering into an enterprise level service contract. The three main providers AWS, Azure, and GCP, each had their own unique pricing strategies. We will not be going into detail for each cloud provider. However they can be separated into three distinct categories, Pay as You Go, Dedicated, and Reserved Instance.

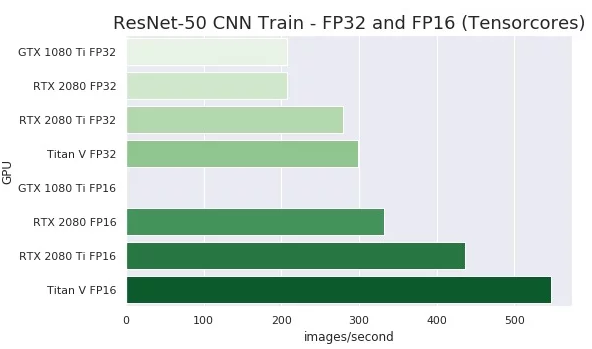
The Pay as You Go pricing model is self explanatory, however the VM gets put to sleep every time you stop using the it, leading to lengthy start up times. Since the power of the VM is on demand you will not be given a discount the longer you use it. And if left running without shut down it can lead large costs. Since this a on demand product it will not get priority within the cloud provider infrastructure.

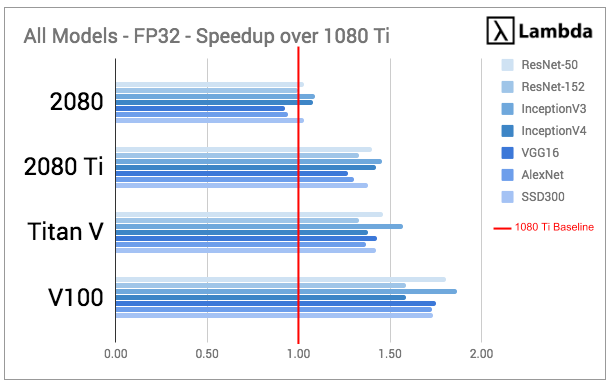
The Dedicated pricing model solves a number of issues that the Pay as You Go pricing model has. However, this comes at a much larger cost, since the VM you are getting is tied to a dedicated physical machine in the data center. Another draw back to Dedicated VMs require to you to fully manage all the software on the machines them selves, which adds another layer of complexity.

The Reserved instance pricing model is a hybrid of the Dedicated and Pay As you go Pricing model. Where you enter into an agreement to use X amount of instances over Y years, at a large discount. However in doing so you can not reduce your capacity at any time. And if you are using less then you payed for, you will not be given a refund or credit. So you have to be very carful with this pricing plan.

The On Premises solution would have the largest upfront cost when compared to the any of the cloud based solutions. However over the long run it would cost as much as the cloud VMs since we could use it as much or as little as we wanted with not penalty. We would also be guaranteed the amount of power in the machine it’s self, since we would not have to fight for priority system resources.

When we looked at the performance between On Premises vs Cloud VMs we found that the newer cloud VMs built for the express purpose of training models was machine higher. Since they had Nvidia’s latest server grade ML GPU known as the Tesla V100s. These V100 VM instances were only offered in a handful of data centers around the world, and they were on average 4 times as expensive as the regular ML VMs. Interestingly we did find that consumer grade hardware did beat out server grade hardware in the mid range GPUs. Which made building our own On Premises machine more viable.





For the maintainability aspect, the On premise solution was the best by far. Since all VMs found in the cloud run some form of Linux by default running a command line interface. This poses a large problem for us since no one on the team has any experience with using configuring the needed Nvidia drivers, and machine learning libraries to train our models.

It is for all the reasons we listed above that we decided to go with a On premise solution. We used a machine we already had, with the following specifications.

|  |  |
| --- | --- |
| CPU | I7 7700K |
| RAM | 32 GB GDDR4 |
| GPU | 2 x GTX 1080 Ti |
| Hard Drive | 1 TB SSD |

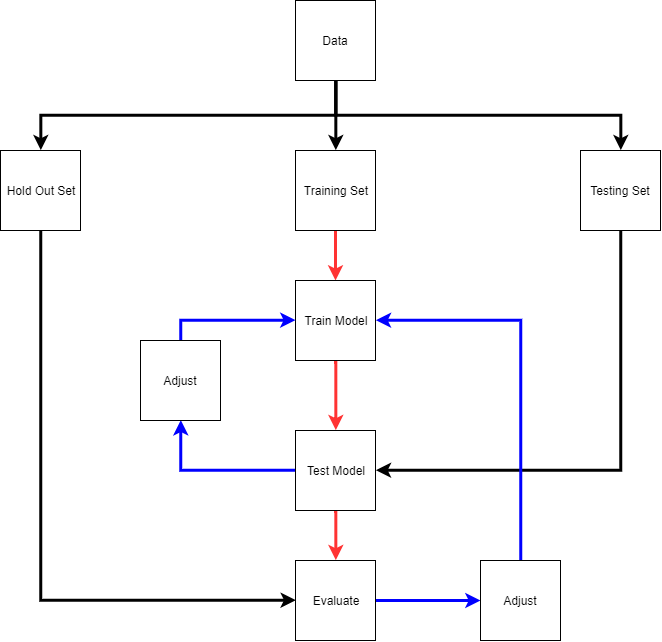
The only new piece of hardware we bought was a new GTX 1080 Ti, since it was all we could afford. Since this computer was already a pre existing computer we had access to, we would be interacting with the standard Windows 10 interface, with a pre existing installation of the drivers and Tensor Flow libraries.

## Network Selection

There are a variety of different approaches that can be taken when dealing with applying ML to image segmentation. Upon further research we found that the ML approach that had the best results, for image based learning tasks were neural networks (NN). NNs come in several varieties, such as Convolutional, Generative Adversarial, and Recurrent NNs. We choose CNNs since they had the most positive results, for ML in medical image processing. NNs have a number of different key attributes, that effect its ability to learn from training data.

## Training Methodology

When training the model, we wanted to reduce the inherit risks involved, such as overfitting, underfitting, and overcomplexity. To do mitigate these risks we decided to develop the model in three different phases. Each phase would involve a new set of training data, and once the accuracy was meet, we would move on to the next phase. This was accomplished by breaking up the data into three parts. The initial data set would be the Training set, which would contain the most data among the three. The second set of data would be the Test set, which would be used to test the how well the model learned from the initial training set. If there was a large amount of error, we would then make the needed changes. The final phase of testing would be using the Hold out set, which would involve us doing one final test to see if the model had learned properly. This whole process would require user intervention at different stages, and would be a iterative cycle. Since we would also be testing different data preprocessing techniques, and hyper parameters among other things.



## Training Data

The data we will be using to train our models, comes from one of the challenges of the Consortium for Open Medical Image Computing (COMIC). They provide a platform for researchers and individuals to compete in ML challenges related to medical images. One such challenge is called Promise 12, in which the challenge is to create a ML model that segments the prostrate in MR scans, with the highest degree of accuracy.

We are given 50 training cases and 30 testing cases. Each case will be in the form of a transversal T2 - Weighted MRI. To increase the difficulty and real world applications of the challenge, each MRI scan comes form different clinical settings, and use different vendors of MRIs. The data its self comes in the form of a combination of an MHD file that stores the image an ASCII readable header file, and a separate binary data file in the form a RAW file.

## Model Evaluation Parameters

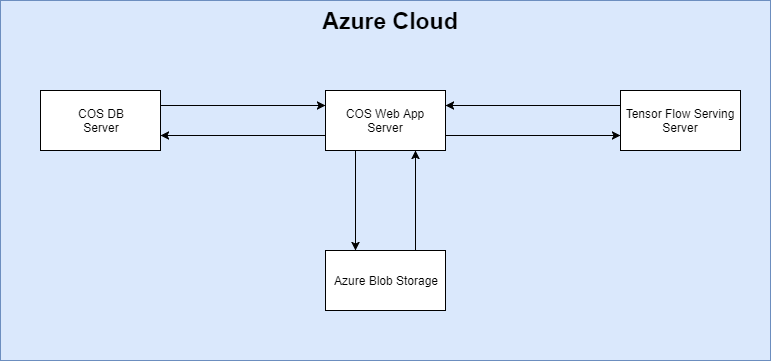
We will be using 4 parameters to evaluate the accuracy of the output of the trained models. The first being the DICE coefficient, which is the measure of how similar two objects are. The second being the

# Conceptional System Design

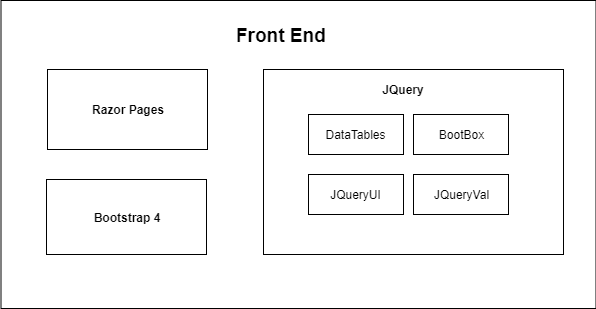
## Application Design

COS is going to be split up into a number of different parts, Front End, Back End, ML Server. The Front End will be responsible for everything the end user will see on screen. The Back End will handle all the logic, and data storage of the application. And the ML Server will host and serve the results of the data provided to it. All three parts will work together to provide a seamless experience to the end user.

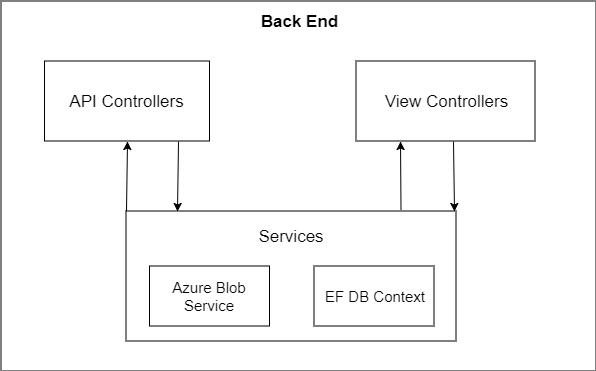
The set of technologies we decided to use to create the application side of the course primarily comes from Microsoft and the open source community. The main components coming from Microsoft being the use the MS SQL database, Azure Blob Storage, ASP.NET MVC web development framework, the use of C# as the primary development language, and the use of Azure. We choose these technologies since it was what we could be the most productive in. Another major reason is that Azure is the only cloud provider that conforms to medical data protections standards in Canada, and it the only cloud provider with two data centers in the country. This is very important since by law all medical data that would be collect can never cross the boarder, and we are required to have replicate our data in two separate data centers for redundancy reasons.



The Front end of the application will be using a mixture of server side and client-side rendering to create the webpages seen by the user. Server side rendering is the process in which the server generates a html document and sends it to the user’s web browser. Where as client side rendering is when the server sends Java Script to the browser once, and then that Java Script talks to server to get data and display it on screen. For the server side rendering aspect we will be the “V” in ASP.NET MVC. The “V” is the ASP.NET’s view engine known as Razor Pages. These Razor Pages allow us to add dynamic content to the HTML document as it is being generated using embedded C#. The Client side rendering aspect is going be using JQuery, which is open source client side library, that uses Java Script to create dynamic interactions to the user. Such as updating new content on the screen without having to refresh the page.



The Backend portion of the application will be using the “M” and “C” in MVC. Which stand for Model and Controller. The Model acts as a Plain Old C-Sharp Object (POCO) that represents an entity and serve to the user. This is accomplished by using a intermediary abstraction between it’s self and the database known as a DB Context. This context is used to populate the POCO instead of executing SQL commands directly against the database. The controllers will be responsible enforcing application logic and telling the front end what to show the end users.



The ML severing part of the application will be responsible for providing an API that can be used by our main web application. This is accomplished with the use of “Tensor Flow Serving” (TFS) which is a technology created by Google to make the deployment of models into production environments much easier. What TFS does, is give you a way to automatically generate a REST API server that you can control to run data into your pre trained model and get the results back out again.

This is in part possible through the use of a virtualization technology called Docker. It allows us to deploy, and preconfigure applications on to any type of hardware. This gives us the ability to write what we need into a Docker file, and have it create the application in a container, inside the primary OS. Whereas applications such VMware require the use of OS at the base of an existing operating system.

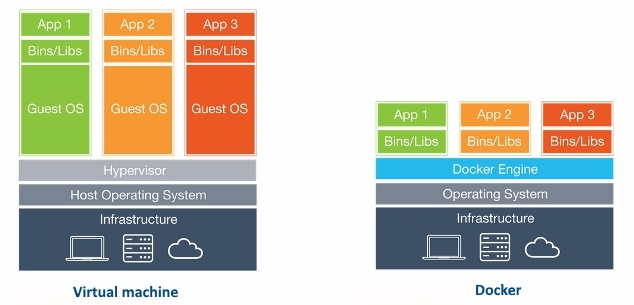


Figure 1 As seen on the right, when application are deployed using docker they do not need a existing OS to be deployed with them.

In conclusion our application is going to a complex integration of a number of different parts. That span a wide variety of technologies. However we have broken it down into two major parts, the application and the ML model.

## Model Design

Data

Accuracy Measure

Architecture

Data preprocessing

System Initialization

Convolutional Layer

Activation Layer

Pooling Layer

Optimization Type

Hyperparameter Optimization

Limiting Factors