FDC322994

Autonomous Geometry Processing Using Machine Learning & Forge

Sandip Jadhav

CEO, Center for Computational Technologies – CCTech

Nem Kumar

Director, Center for Computational Technologies – CCTech

Vijay Mali

CTO, Center for Computational Technologies – CCTech

Learning Objective

- Demonstrate use of machine learning in the future of autonomous manufacturing
- Apply machine learning techniques for problems in geometry
- Integrate Autodesk Forge services in webapp
- Select appropriate AWS computing services for machine

Description

One of the biggest hurdles in using an STL mesh model for a downward manufacturing application is the lack of geometry feature information in the STL file.

This class will demonstrate how machine learning techniques can be applied to extract geometrical information from mesh models; and explain the process of building machine learning models, feature engineering for mesh data, and parameter tuning. We will also cover Autodesk Forge integration for data translation and viewing; and demonstrate how to use AWS cloud infrastructure such as GPU computing for big data and AWS Lambda for fast online predictions.

Your AU Experts

Sandip Jadhav

CEO, CCTech (sandip@cctech.co.in)

Sandip is a successful entrepreneur in the CAD/CAE space. He has co-founded CCTech, Zeus Numerix, Adaptive 3D Technologies, LearnCAx and recently simulationHub, a cloud-based fluid flow simulation web service. Sandip has led several product developments teams in conceptualizing, designing software and implementation of apps in CAD and simulation space. Sandip is a passionate software developer and loves to tinker with technology.

Vijay Mali

CTO, CCTech (vijay@cctech.co.in)

Vijay is a technology explorer, a visionary and a product maker. As CTO of the company, he plays a critical role in deciding the technology vision of the company. He also leads the center of excellence (CoE) department at CCTech which is responsible for exploring new technologies & building a strategy to bring it to common designers.

Nem Kumar

Director, CCTech (nem@cctech.co.in)

Nem Kumar is director of consulting at CCTech and has been doing product development with companies from Manufacturing, Oil & Gas and AEC domain. He has vast experience in Desktop as well as Cloud software development involving CAD, CAM, complex visualization, mathematics and geometric algorithms. Nem has been actively working with Autodesk Vertical, Research and Fusion 360 teams. His current areas of interest are Generative Modeling and Machine Learning.

Abstract

CCTech is building an autonomous geometry processing application which uses machine learning algorithms to decipher intelligent information from STL meshes. These goals are being achieved using 4 steps:

- First segment the STL mesh into six basic surface types viz. cylindrical, spherical, planar, conical, B-spline and torus
- Next group these detected surfaces and try to identify high level features such as holes, taper holes, chamfers, fillets
- Further on, this feature level information to understand parts and assemblies.
- Finally, an AI engine will identify various types of parts and tag them as per the industry

A micro web services has been created that takes STL files as input and gives fully segmented mesh as output. Another web app will facilitate the object detection functionality from 3D CAD files.

Table of Content

1 Introduction

Most of our design techniques as engineers date back thousands of years. Perspective drawing was invented in the 1300s, descriptive geometry was invented in 1765, orthographic projection was invented in 1770, 2D CAD was invented in the 1970s and 3D CAD was invented in the 1980s. As we can see, 3D CAD software came to the market just a couple of decades earlier, but our mainstream manufacturing and architectural industry has been using other old forms of engineering drawings from over a century or so. The products manufactured and the constructions done before the wide scale adaptation of 3D CAD are widely in use today. The designs of most of these wonderful creations would be either done on real paper or may have been digitized into 2D CAD drawing files. We know that in today's world, 3D CAD files are of utmost importance. It helps to carry out renovation work, to perform simulations, and to improvise the designs. And hence, it becomes highly important for us to convert these designs into 3D CAD formats. There are millions of such designs in the world that needs to be converted into 3D CAD files. However, organizations can't afford to deploy CAD designers to carry out this mammoth task manually. If done so, it might take ages to complete. We need some sort of software automation or artificial intelligence to swiftly get this job done.

Let's first discuss solutions to bring the non-digital design data into digital formats –

- Designs which are on paper can be primarily scanned and then software algorithms or AI can be applied to generate their 3D CAD models
- 3D scanners can be used to scan and create point data cloud files from the manufactured products or architectural buildings which do not have design data in any format at all

Once we have the design data in any digital format, we can work on solutions to understand the data and create 3D CAD models containing all feature level information inside it. But what exactly makes 3D CAD models so useful than other format of storing design data? It is the rich information of features and connectivity between the features that is present in the 3D CAD files that makes them so important. Whereas raw files in which design data is stored may have very less information. For example, in STL files, the shape of the 3D geometry is created using many triangles. The file in true sense contains the 3D coordinate values of every vertex of these triangles. You must have figured out by now that the automation solution to generate feature rich CAD model from every raw design data format will vary largely. However, in this era of automation and artificial intelligence, we should not stop with the generation of 3D CAD models from this low-level design information. Instead, we should bring in an aspect of intelligence into the 3D CAD models like detecting contextual names of each part present in a 3D CAD file. For ex. there should be an AI engine which can read a 3D CAD file and identify the names of all objects present in that CAD model. For ex. let's say if I upload a CAD model of a house with all furniture inside it, the AI engine should be able to identify objects like walls, sofa, table, chairs, TV, fridge, etc. Imagine if we start to have 3D CAD models and this type of contextual & intelligent information of every single design ever made in this world, a lot of possibilities open that can be done over these intelligent design data. So, let's make the design information smarter.

Based on our market understanding we realized that one of the major needs is to process the 3D scanner data of a real-world geometry and generate a 3D model out of it. We also found that most of the design data available in digital format are in STL or few other similar formats. Even the output of 3D scanner, i.e. point cloud data, can be easily transformed into formats like that of STL. In our current work, we start with a target to convert STL files into feature rich 3D CAD models. In order to achieve this, we decided to take a phase wise approach –

- **Phase 1.1** – Detect and group triangles that belong to a primitive surface type like cylindrical surface, spherical surface, planar surface, etc.
- **Phase 1.2** – Detect and group a collection of primitive surface types (detected in phase 1) into a feature, for ex. primitive features like chamfer, filler, or engineering features like hole, taper hole, threading for screw, flange in pipe, etc.
- **Phase 1.3** – Use all the information collected from the phases till now to generate the feature rich 3D CAD model of the given STL geometry
- **Phase 2** – Create an AI-Powered solution that can identify names of 3D objects from CAD files, like that of object detection from images.

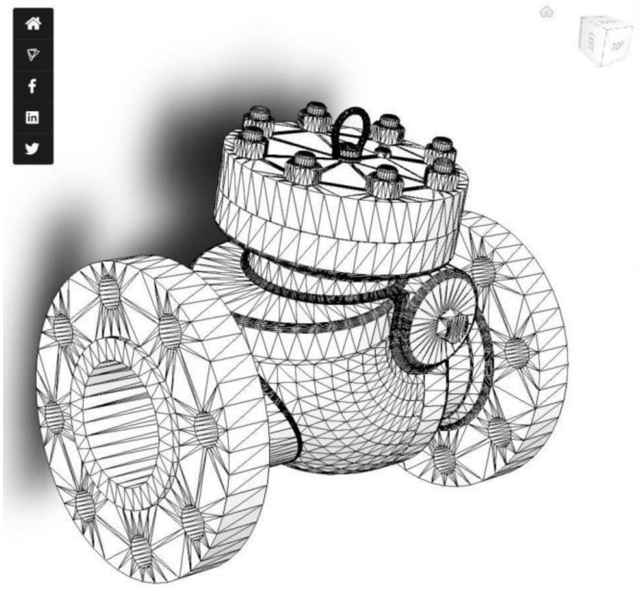
This course limits the scope to phase 1.1 and 2 only i.e. to identify primitive surface types from a STL geometry file and object detection from 3D CAD models. Our first attempt was to create a heuristic algorithm with a lot of rules that can classify every triangle of a given STL file into the following surface types –

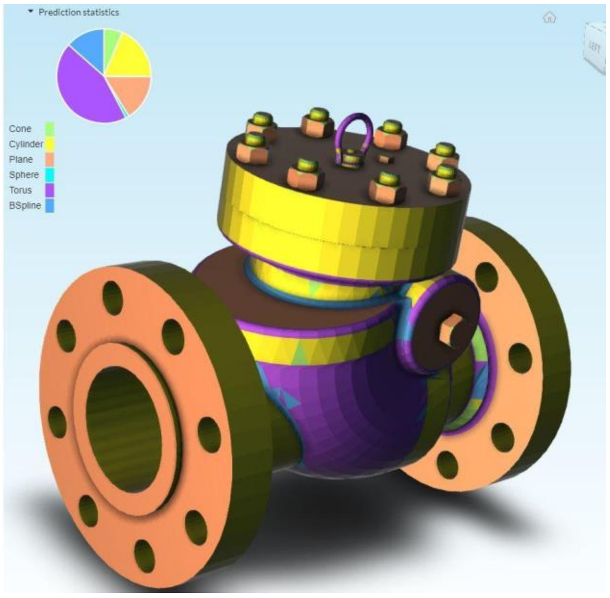
- Cylindrical surface
- Spherical surface
- Torus surface
- Conical surface
- Planar surface
- Any other generic BSpline surface

It gave good success on our collection of geometries used for testing purposes. But as we kept on using it on more and more professional STL files, the accuracy started to drop. We understood that the variations in triangulations from case to case can be enormous and writing rules for each such variation could simply be out of the scope of heuristic approach of solving this problem. The lesson we learnt from our very first attempt was that we need to find a solution using AI and ML here.

2 Phase 1 Problem Definition

Detect and group triangles from an STL file that belong to a primitive surface type

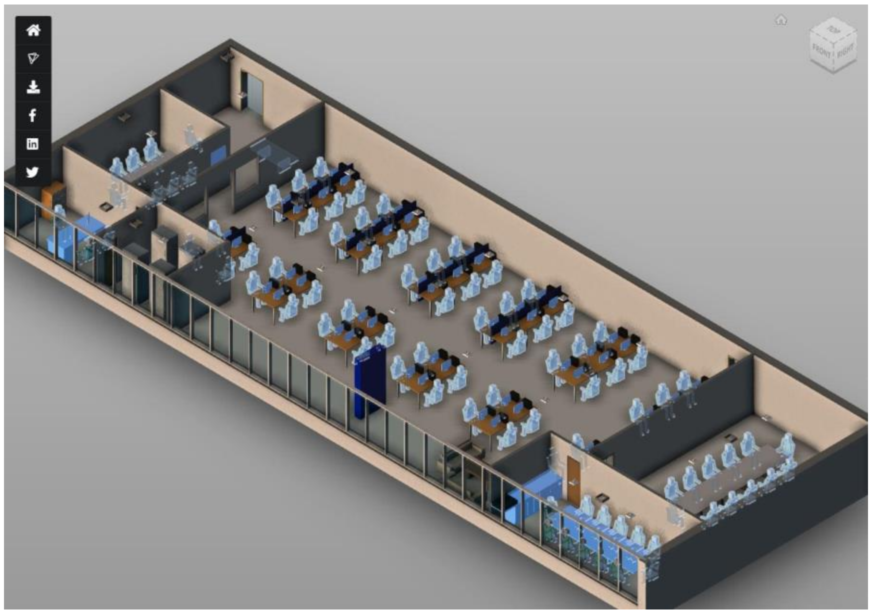


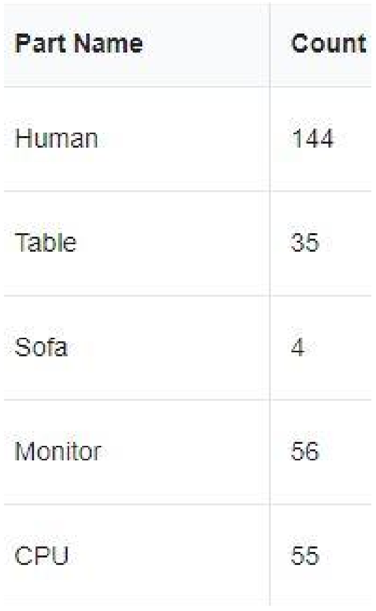


An AI-powered autonomous web application is to be created which will accept a STL file of a 3D geometry (left) and generate a new OBJ file (right) in which all the triangles have been classified into its primitive surface types.

3 Phase 2 Problem Definition

Identify names of 3D objects from CAD files





Part Name	Count
Human	144
Table	35
Sofa	4
Monitor	56
CPU	55

An AI-powered autonomous web application is to be created which will accept a 3D CAD file and identify each object inside it.

4 What the world is experimenting?

Finding good amount of literature helped us to understand that data scientists around the globe have done commendable work in decomposing a complex 3D CAD model into simpler sub-objects. You will be aware of the term "3D semantic segmentation" which can be described as splitting a 3D geometry based on its connectivity.

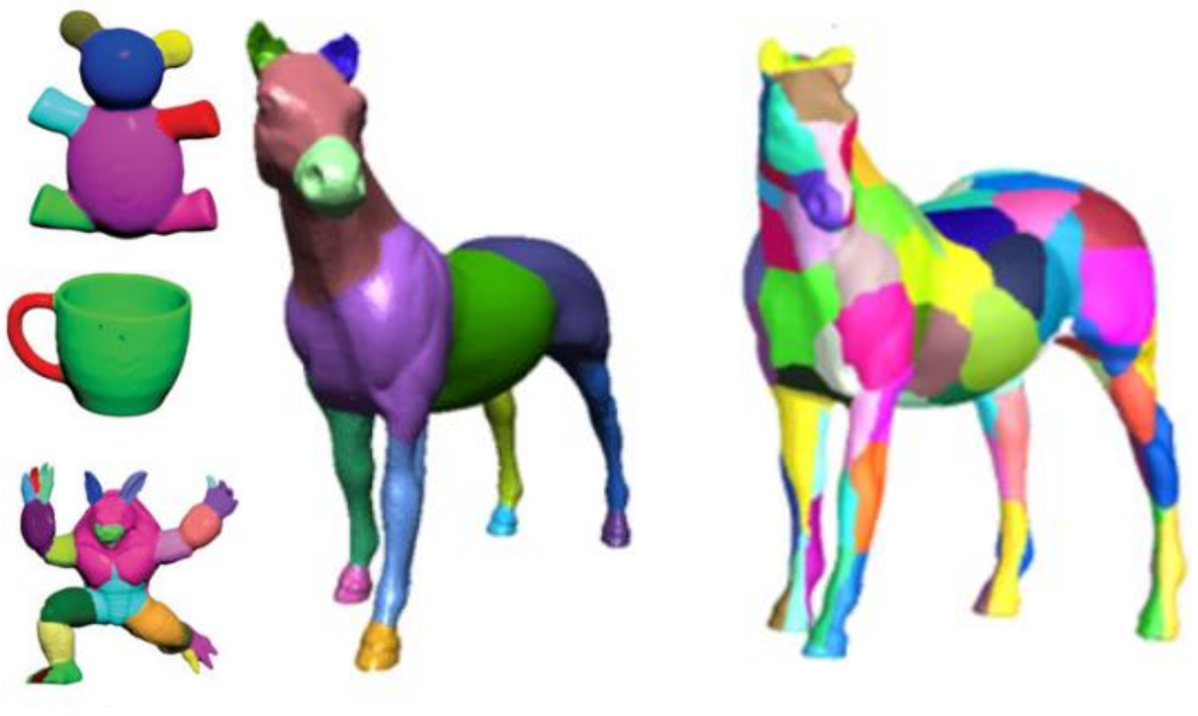


Image courtesy: 1) Shu, Z., et al. Unsupervised 3D shape segmentation and co-segmentation via deep learning. Comput. Aided Geom. Des. (2016), http://dx.doi.org/10.1016/j.cagd.2016.02.015

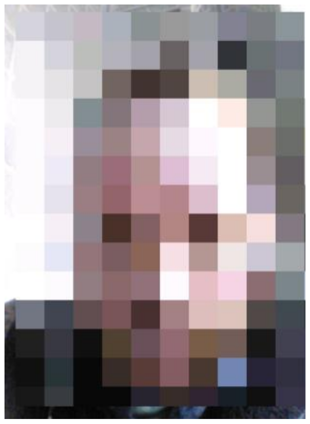
2) A Survey on Mesh Segmentation Techniques, Efi Arazi School of Computer Science, The Interdisciplinary Center, Herzliya, arik@idc.ac.il

There is a community of researcher in "Geometric Deep Learning" which is nicely coming up together to facilitate advancements in the area of geometry using artificial intelligence. There are a lot of research articles and software libraries coming up in this area. We also got inspired with this movement and started applying our knowledge of geometry and AI together.

5 Challenges in applying AI to geometrical data

Most of the machine learning is done on structured dataset. We can call a dataset to be structured when all the data points in the dataset can always be presented with the same number of properties. Let's say we have an image dataset of “cats” vs “non cats” -

And every image in the dataset has the resolution of 512 x 768 px. So, each image can be described using a fixed number of properties.



$$512\ (pixels\ horizontally)\ \times$$

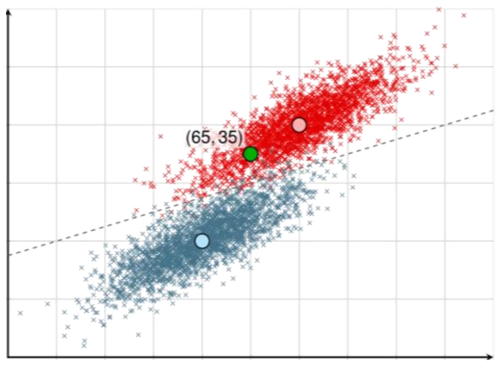
$$768\ (pixels\ vertically)\ \times$$

$$3\ (RGB\ values\ of\ each\ pixel)\ =$$

$$\mathbf{1,179,648}\ (properties)$$

So, the input to the machine learning system will a 2D matrix of RGB values along with a binary output “0” or “1”. If the image is of a cat, then the output is set to 1 else it is set to 0.

Let’s take another example of a simple 2-dimensional classification problem. Here all data points are specified using X and Y coordinates. And hence this dataset is also an example of structured data.

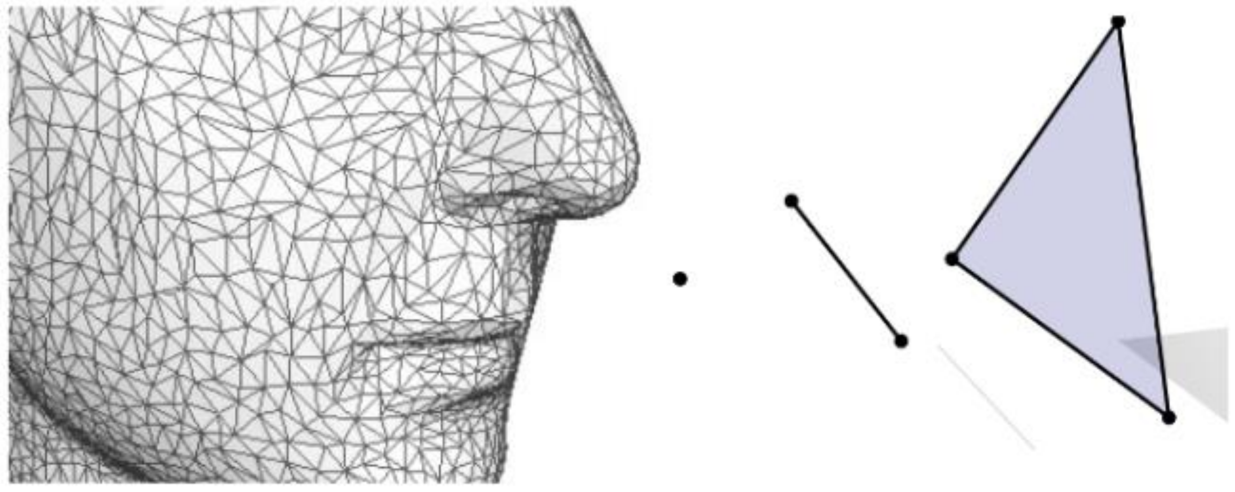


X coordinate	Y coordinate	Class Name
10	5	A
15	10	B

Today in machine learning field, we have achieved tremendous success in datasets consisting of images, text, audio and numbers. All of these are structured data. However, if you see, none of the real-world problems are inherently structured in nature. It is the work of data scientists to convert this unstructured data collected from the real-world problems into a structured pattern so that machine learning algorithms can be applied on them.

In our case, this challenge popped up as the first hurdle in the project. what we figured out was just like images are built out of pixels like that 3D meshes are built from triangles, edges and vertices. If we can form structured data out of images, then we can try the same for mesh also.

We brainstormed for more than a month and came up with an innovative strategy to encode STL data into a machine learnable format.



6 Phase 1 work

6.1 Dataset preparation

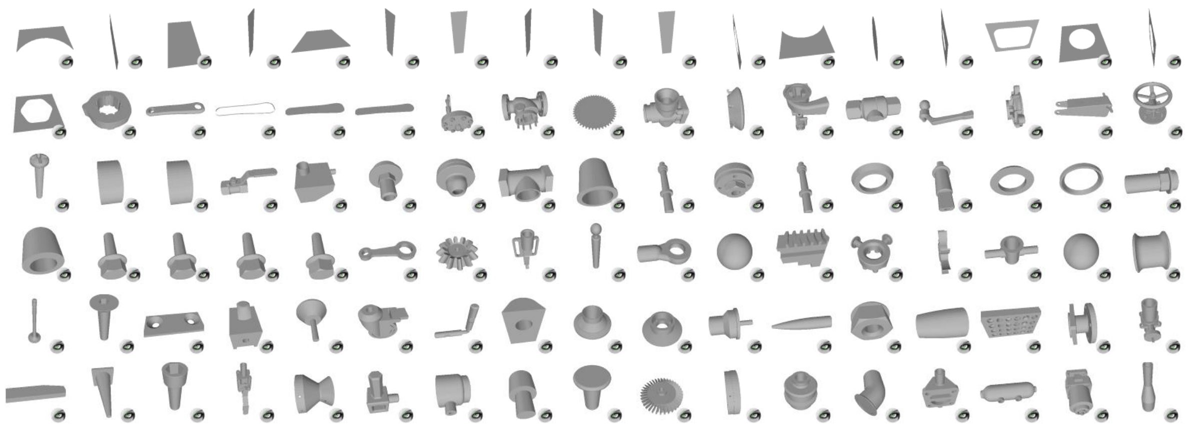
Machine learning depends heavily on data. If you can't make sense of data records, a machine will be nearly useless or perhaps even harmful. That's why dataset preparation is such an important step in the machine learning process. In a nutshell, data preparation is a set of procedures that helps make your dataset more suitable for machine learning.

The process for getting data ready for a machine learning algorithm can be summarized in the following steps:

6.1.1 Dataset collection

Dataset are an integral part of the field of machine learning. Having a rich data helps the learning algorithms to learn in more precise manner and make the ML model more reliable. High quality labeled training datasets for supervised and semi-supervised learning algorithms are usually difficult and expensive to produce.

In our study, we build our own 3D CAD model database as a test bed for the research. All the CAD models are collected from several mechanical manufacturing enterprises. The models are designed by engineers with mainstream commercial CAD toolkits such as Autodesk Inventor. There are a total of 1500 models belonging to several generic categories: engine, valves, gears, screws, nuts, wheels, keys, bearing houses, flanges, washers, etc. These 1500 models contain 37 million triangle count which are split into three parts train, validation and test. The training set and validation set are used to perform model selection and hyper parameter selection, whereas the test set is used to evaluate the final generalization error and compare different classifiers in an unbiased way. Fig shows a portion of the 3D models in our dataset.



Triangles label in complete Dataset is categorized as follows:

Torus	Cylinder	Plane	B Spline	Sphere	Cone
28.47 %	23.16 %	16.30%	14.5 %	10.9 %	6.67 %

6.1.2 Feature generation

Feature generation is also known as feature construction, feature extraction or feature engineering. There are different interpretations of the terms feature generation, construction, extraction and engineering. Some nearly equivalent, yet differing definitions for these terms are:

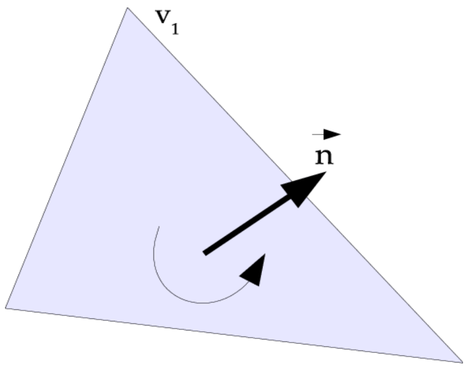
- Construction of features from raw data
- Creating a mapping to convert original features to new features
- Creating new features from one or multiple features

It is a process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning and is both difficult and expensive.

A feature is an attribute or property shared by all of the independent units on which analysis or prediction is to be done. Any attribute could be a feature, as long as it is useful to the model.

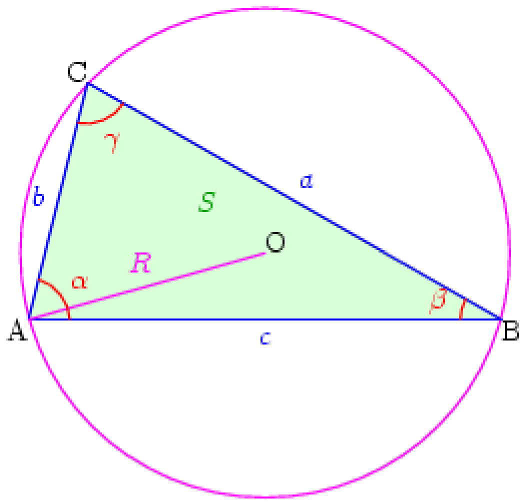
According to the specific nature of the problem domain, selecting features that have obvious distinguishable meaning is a critical step in the pipeline of our approach. The features should be invariant to irrelevant deformation, insensitive to noise, and very effective for distinguishing different categories of CAD models.

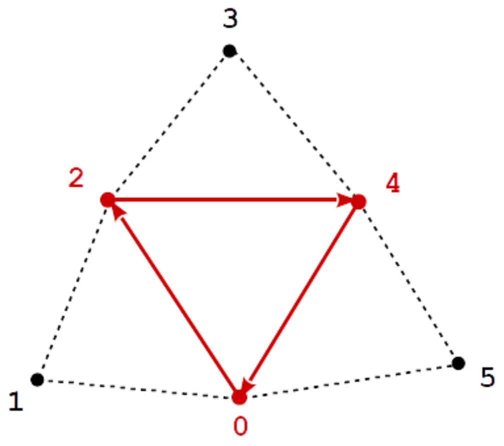
With the increase of data, it has become clear that the input of learning algorithm can have a significant impact on the performance of learning algorithms. Therefore, data pre-processing is becoming more and more important. Data pre-processing is a collective name for all methods that aims to ensure the quality of the data. So, in data pre-processing we focus on two methods called feature generation and feature selection.

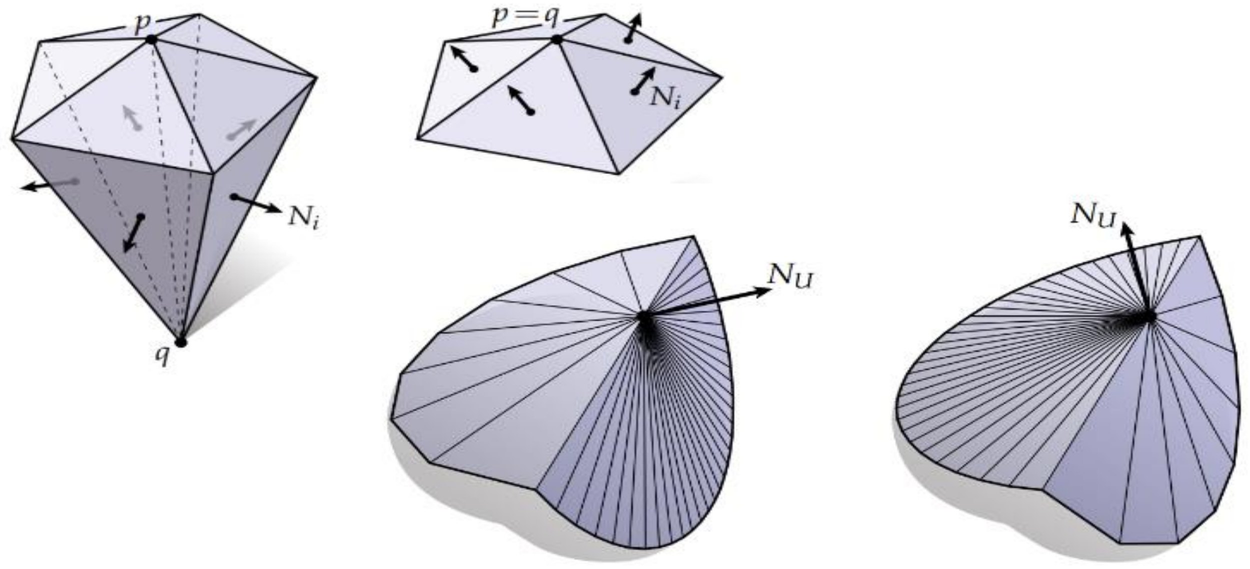


In our case we have only coordinates of points in three-dimensional space. From these coordinates there are certain properties of triangle which can be derived like area, aspect ratio, edge lengths, circumradius, centroid distance from all the three vertex, internal angles, normal of a triangle, valency of triangle, type of triangle and many others.

We have also derived features from above properties because derived features is a way to inject expert knowledge into training process and so to accelerate it. Some of our derived features are ratio of angle between normal, ratio of edge length, featuring lower and upper quartile values of numerical parameters, distance of centroid from other triangle, triangle illumination etc.







We wanted to give our ML algorithm knowledge of visual information of all triangles that it will try to categorize. And solid angle was our choice to make that happen. A solid angle is a measure of the amount of the field of view from some particular point that a given object covers. That is, it is a measure of how large the object appears to an observer looking from that point.

The total count of features added up to **100**.

6.2 Data analysis and visualization

After getting the dataset, the next step in the model building workflow is always data analysis. Analyzing the data by using general statistical tools and obtaining numerical values consisting count, max value, min value, mean, median, standard deviation and quartile range. These numerical values provide the univariate analysis of data which sometimes is tough to read through values. Analyzing the data is easier by using data visualization tools, some tools help us to relate various features within data.

Understanding insights using files becomes more difficult when the size of the dataset increases. In data visualization we have used different graphs and plots to visualize our data to ease the discovery of data patterns. It helped us to identify areas that need attention e.g. outliers and to understand the factors that have more impact on our result.

All the data is transferred into some form of plots and analyzed further from that. Software packages like NumPy, Pandas and Matplotlib are great for visualization. Seaborn is also a great package which offers a lot more appealing plot.

There are different types of analysis as mentioned below:

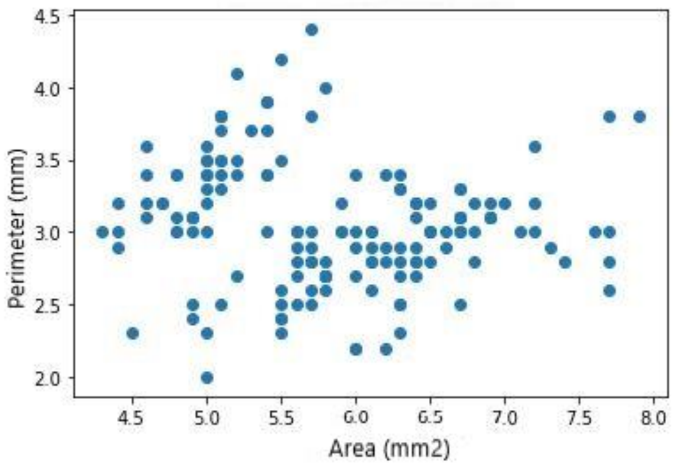
- **Univariate**: In univariate analysis we use single feature to analyze almost all of its properties

- **Bivariate**: When we compare the data between exactly 2 features then it's called bivariate analysis
- **Multivariate**: Comparing more than 2 variables is called as Multivariate analysis

Visualization Analysis used:

6.2.1 Scatter plots

Scatter plots are bivariate or triradiate plots of variable against each other. They helped us to understand the relationships among the variables of dataset. This plot also gave us a representation of where each point in the entire dataset are present.



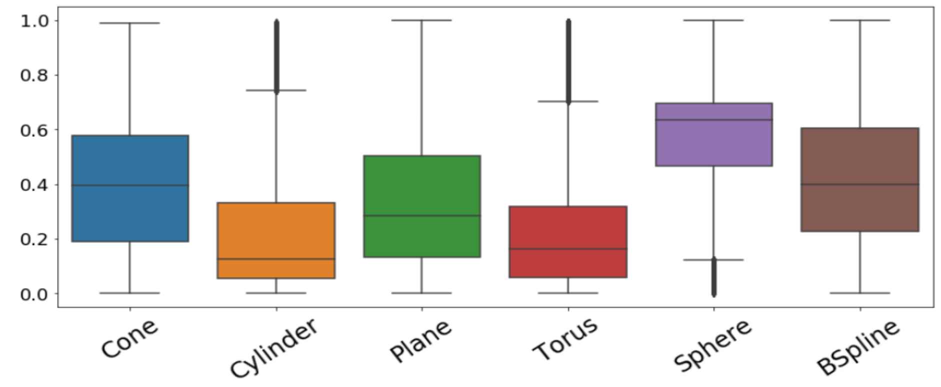
To find the relationship between features like Area and Perimeter, Area and Aspect Ratio, Area and Interior Angle, Perimeter and Aspect ratio and various other. We have concluded how differently all such parameters vary, as we can assume that triangle having greater area must have greater perimeter also but as seen in the above graph some triangles having smaller area also have larger perimeter, thus finding out such a huge variation within data provides more insight for training. Such non uniformity is also observed between Perimeter and Aspect Ratio, Edge length and Area and various others.

6.2.2 Box Plots

This is the type of plot that is used to obtain more of statistical details about the data. The straight lines at the maximum and minimum are also called as whiskers. Points outside of whiskers will be inferred as an outlier. The box plot gives us a representation of the 25th, 50th ,75th quartiles. From box plot we can also see the Interquartile range(IQR) where maximum details of the data will be present. It also gives us a clear overview of outlier points in the data.

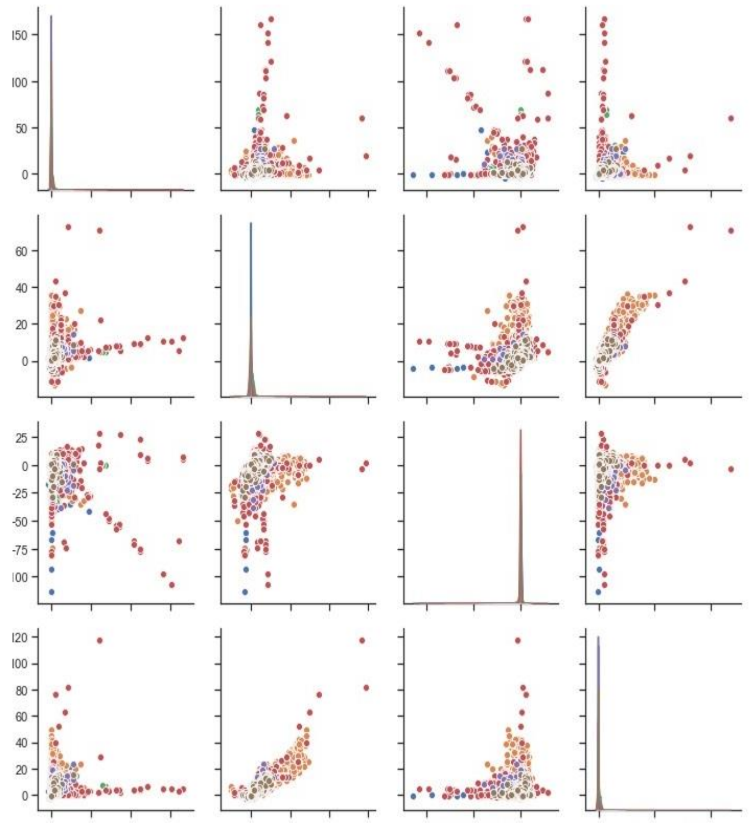
In our dataset box plot helped us to find outliers in features such as Area, Perimeter, Valency, Angle between Normals and various other. This visualization helped us to understand the distribution of data points within a column. In feature like edge length we found there are greater outliers and now the decision is with us either to remove such values, replace with maximum or mean value, or go with the same. After analyzing the cause of such large edge length values in dataset we found out there are some planer triangles having larger edge length than usual and we cannot ignore such values.

Thus, Box plot helps us to analyze the variation in data is important or not.



6.2.3 Pair Plots

Let's say we have n number of features in a data, Pair plot will create a figure where the diagonal plots will be histogram plot of the feature corresponding to that row and rest of the plots are combination of features from each row in y axis and feature from each column in x axis. As we can see in below plot, we have plotted solid angles. We can see which two angles can separate the data very well.



6.3 Standardization

Scaling ensures that the values have a common scale, which makes analysis easier

- Standardize units: Ensured all observations under a variable have a common and consistent unit, e.g. converted cm to mm, radian to degree etc.
- Scale values such that variable have a common scale
- Standardize precision for better presentation of data, e.g. 4.5312341 mm to 4.53 mm
- Removed outliers: Removed high and low values that would disproportionately affect the results of our analysis

We must normalize our data because our features do not have a uniform scale. Data normalization is the process of rescaling one or more attributes to the range of 0 to 1. This means that the largest value for each attribute is 1 and the smallest value is 0.

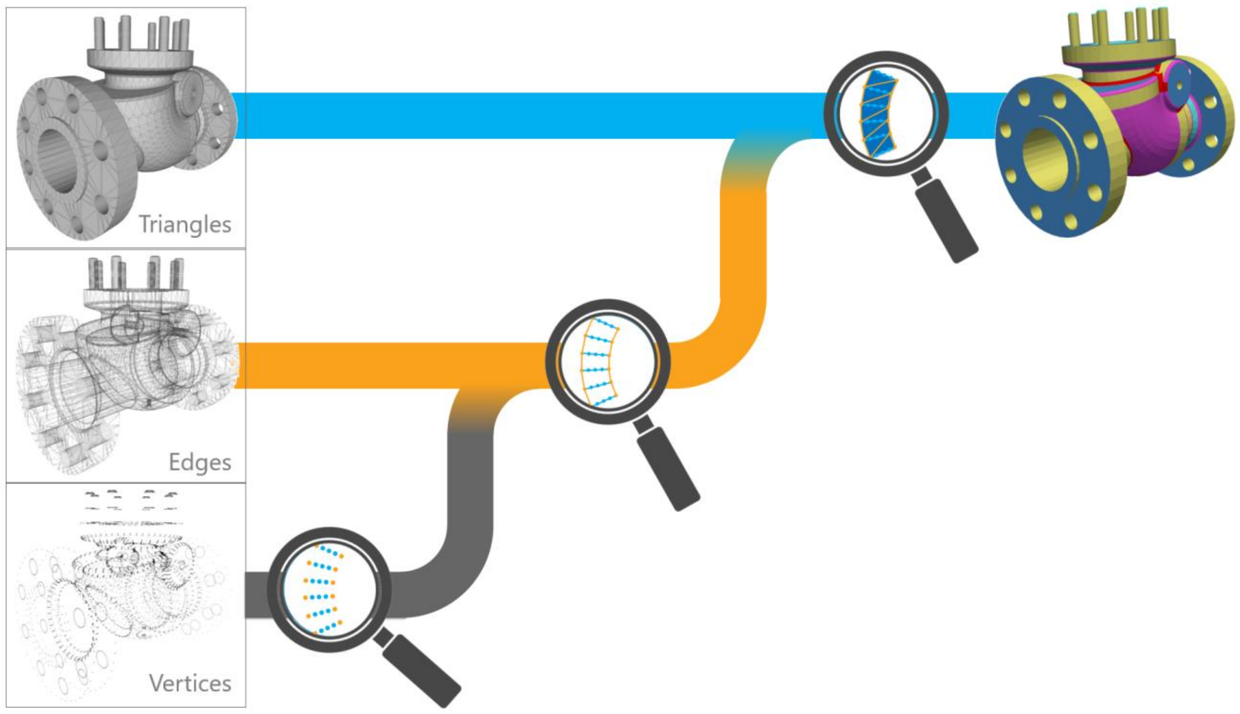
Before Normalizing the dataset, we have Angle1 feature in range of (0 - 119.87)

Feature	Mean	Std	Min	25%	50%	75 %
Angle1	44.97	32.43	0.00	14.57	38.23	74.31

After Normalizing the dataset, we have modified Angle1 feature in range of (0 - 1.00)

Feature	Mean	Std	Min	25%	50%	75 %	Max
Angle1	0.38	0.27	0.00	0.12	0.32	0.62	1.00

6.3.1 Multi-Level Evaluator



Having an stl file which consists of only coordinates value of vertices of a triangle and normal, we first work on converting the raw coordinates into derived one as we discussed in previous sections and created an ML model to detect vertices and label them into seven categories one extra one is of boundary, further these predicted vertices serve as an input along with derived edge feature for classifying edges connected to two vertices, we have also tried to classify edge without using vertex prediction but our resultant outputs are much more enhanced by labeling vertex of that edge. Now we are much more closer to predict triangle of which the stl file is composed of, as triangle is made up of three edges and three vertices and we have expected output of all of them, so along with derived features of triangle we pass all vertex and edge prediction to classify and results obtained are much more precise and accurate. This finally help us to categorize triangles in between six classifiers.

6.4 Training the Machine Learning Models

We used the following tools to develop our machine learning model:

TensorFlow - TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks.

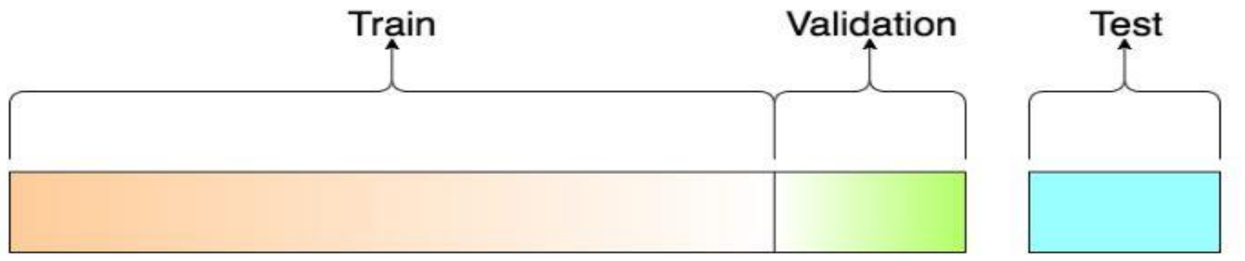
Keras - Keras is an open source neural network library written in Python. It can run on top of TensorFlow, Microsoft Cognitive Toolkit, or Theano. Its user-friendly API interface enabled us to do fast experimentation with deep neural networks.

6.4.1 Training

The process of training an ML model involves providing an ML algorithm with training data to learn from it. Here we have dataset consisting of 65 million data points which consists of 150 input variables/ features. We have considered 70% of data for training else 20% for validation and remaining 10% for test.

To create a successful machine learning model, it is important that split the data into train, test validation.

- **Training Dataset**: The actual dataset that we use to train the model. The model sees and learn from this data.
- **Validation Dataset**: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.
- **Test Dataset**: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.



Training such a huge amount of data on local machine is not possible, as the size of training dataset is 0.7*65 million = (45.5 million)

Considering a batch size of 50,000 our training machine has to store a vector size more than of [50,000 ,150, number of neurons in Network], as this size of matrix is more dependent on network size and having a huge network to learn complex information computation power needed is very huge.

For training the ML model on huge data size we have two options either purchase a GPU machine having Tesla card and huge processing power with multi core processor that costs around $10,000 or rent a same machine from various service providers. We have opted for renting an EC2 machine from AWS having below specifications.

6.4.2 ML Training Machine Specifications

- 4 x Tesla V100
- 64 GB GPU Memory
- 32 CPU Cores

- 244 GB Ram Memory
- Cost - $12.24 / hour
- 2000 ML training iterations completed in 25 hours using above machine (~ $150)

We started our training on EC2 machine. At an initial phase we set number of epochs to be 1000, optimizer is ‘relu’ and loss function to be ‘categorical cross entropy’. After training is done, we have the final model at 1000th epoch which has larger loss than 769th epoch model. To fix this problem and save the optimal model we used the concept of Model Checkpoint provided by Keras.

6.4.3 Model Checkpoint

A callback is a set of functions to be applied at given stages of the training procedure. We can use callbacks to get a view on internal states and statistics of the model during training. This callback will save our model as a checkpoint file (in hdf5 format) to disk after each successful epoch. We can also write either the loss value or accuracy value as part of the log’s file name.

Now we have fixed the problem to save model with minimum loss, but still the accuracy figures are in the range of 81% which are not satisfactory. Our next move was to tune hyperparameters in the network such as network size, number of neurons in each layer, change the activation function, assign the maximum number of batch size that is possible and changing the learning rate of different optimizer. Here is the brief detail on hyperparameters.

6.4.4 Changing Hyperparameters

Model optimization is one of the toughest challenges in the implementation of machine learning solutions. Hyperparameters are settings that can be tuned to control the behavior of a machine learning algorithm. Conceptually, hyperparameters can be considered orthogonal to the learning model itself in the sense that, although they live outside the model, there is a direct relationship between them.

Hyperparameters we have used are:

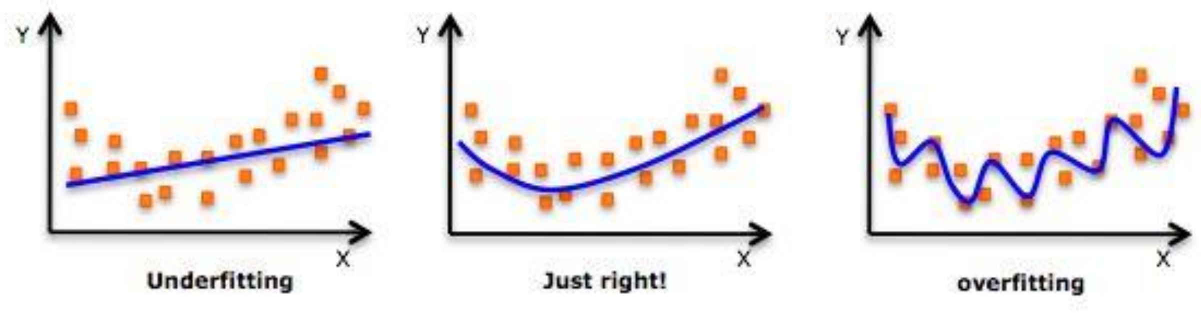
- **Number of hidden units** - A hidden layer in an artificial neural network is a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function. Are classic hyperparameter in deep learning algorithm, the number of hidden units is key to regulate the representational capacity of a model.
- **Number of Neurons** - Assigning the particular number of neurons in a definite layer plays a major role in making the model more accurate.
- **Learning rate of model** - It quantifies the learning process of a model in a way that can be used to optimize its capacity

- **Optimizer** - Optimizers shape and mold your model into its most accurate possible form by futzing with the weights.
- **Epochs Rate** - The number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset.
- **Batch Size** - The batch size is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated.

After tuning all the hyperparameters we obtained the desired training accuracy of approx. 89%. Now we are more concentrated to test our model on unseen test data in which one or two things might happen: we overfit our model or we underfit our model. We don't want any of these things to happen, because they affect the predictability of our model - we might be using a model that has lower accuracy and/or is not generalized. So, let us explain to you what these terms are:

- **Overfitting**: Overfitting means that model we trained "too well" i.e. this model is very accurate on training data, but it is not very accurate on untrained or new data. Basically, when this happens, the model learns or describes the "noise" in the training data instead of the actual relationships between variables in the data. This noise, obviously, isn't part of any new dataset, and cannot be applied to it.
- **Underfitting**: In contrast to overfitting, when a model is underfitted, it means that the model does not fit the training data and therefore misses the trends in the data. It also means the model cannot be generalized to new data.

We want to avoid both of those problems in data analysis. You might say we are trying to find the middle ground between under and overfitting our model.



At first attempt when we test our model on unseen data its accuracy was in the range of 64.9 %. Because on training data it learned very well with an accuracy of 89% but when we test it on unseen data it was not predicting accurately. Our model was overfit, the reason was we have selected all the features.

Our feature count was 150. We decided to study all the features and for this study we did feature importance study. We all may have faced this problem of identifying the related features from a set of data and removing the irrelevant or less important features with do not contribute much to our target variable in order to achieve better accuracy for our model.

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. After computing feature importance by using python library 'eli5' which helps to debug machine learning classifiers and explain their predictions, we selected 100 features for our training.

The benefits we got after performing feature selection are:

- Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.
- Improves Accuracy: Less misleading data means modeling accuracy improves.
- Reduces Training Time: Fewer data points reduce algorithm complexity and algorithms train faster.

Other main issue arises while training large set of data is computational time and machine cost, and after deriving optimal set of hyperparameters and selecting all the contributing features for training we generally set higher number of epochs to best fit the model. Large epochs mean larger amount of iteration and after setting the optimize weights in the model, the loss does not further reduce but as we have set the range of epoch to be large, we must invest time and money both for training. Thus, to overcome this problem and stop the training if there is no further reduction in loss and accuracy is not getting improved, we can use early stopping method provided by Keras.

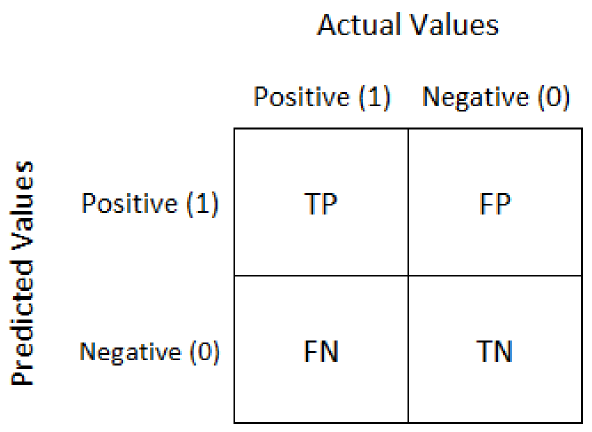
6.4.5 Early Stopping Method

One technique to reduce overfitting in neural networks is to use early stopping. Early stopping prevents overtraining of our model by terminating the training process if it's not really learning anything. This is flexible — we can control what metric to monitor, how much it needs to change to be considered "still learning", and how many epochs in a row it can falter before the model stops training.

6.5 Validation

After training the model how we can measure the effectiveness of our model. Better the effectiveness, performance and that's what we want. And it is where the Confusion Matrix comes.

Confusion Matrix is a performance measurement for machine learning. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your model is confused when it makes predictions. It gives us insight not only into the errors being made by a model but more importantly the types of errors that are being made.



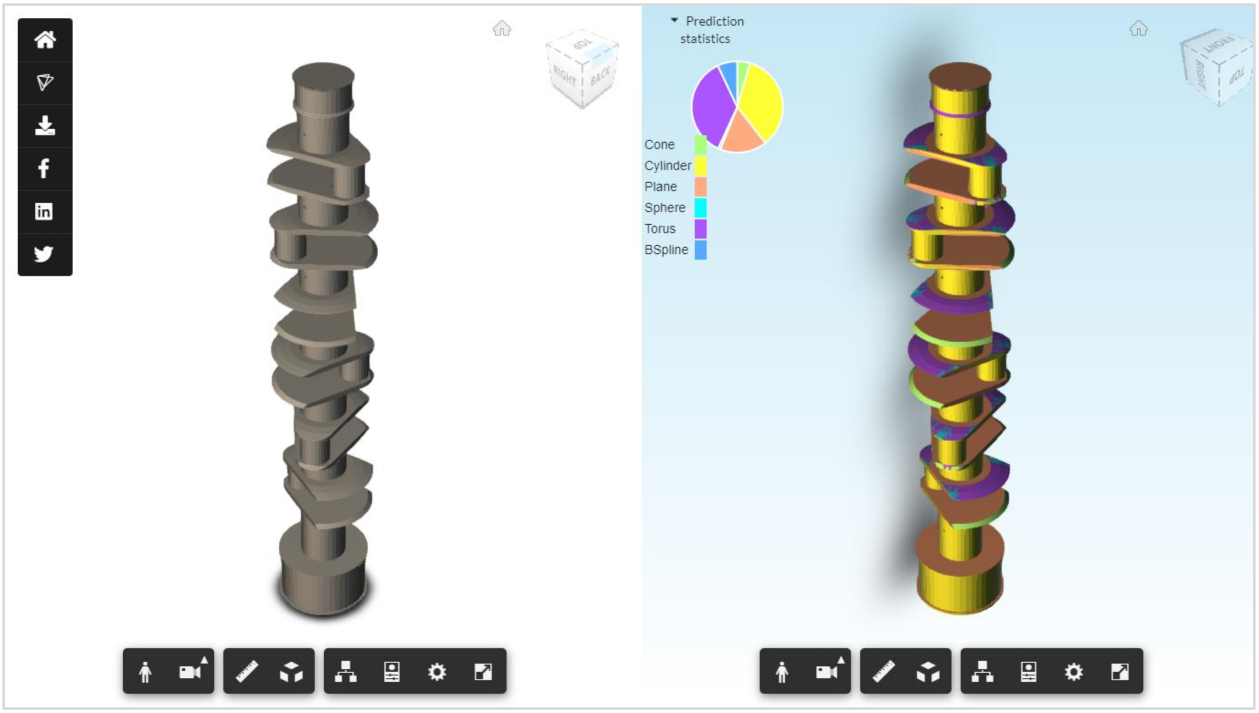
		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

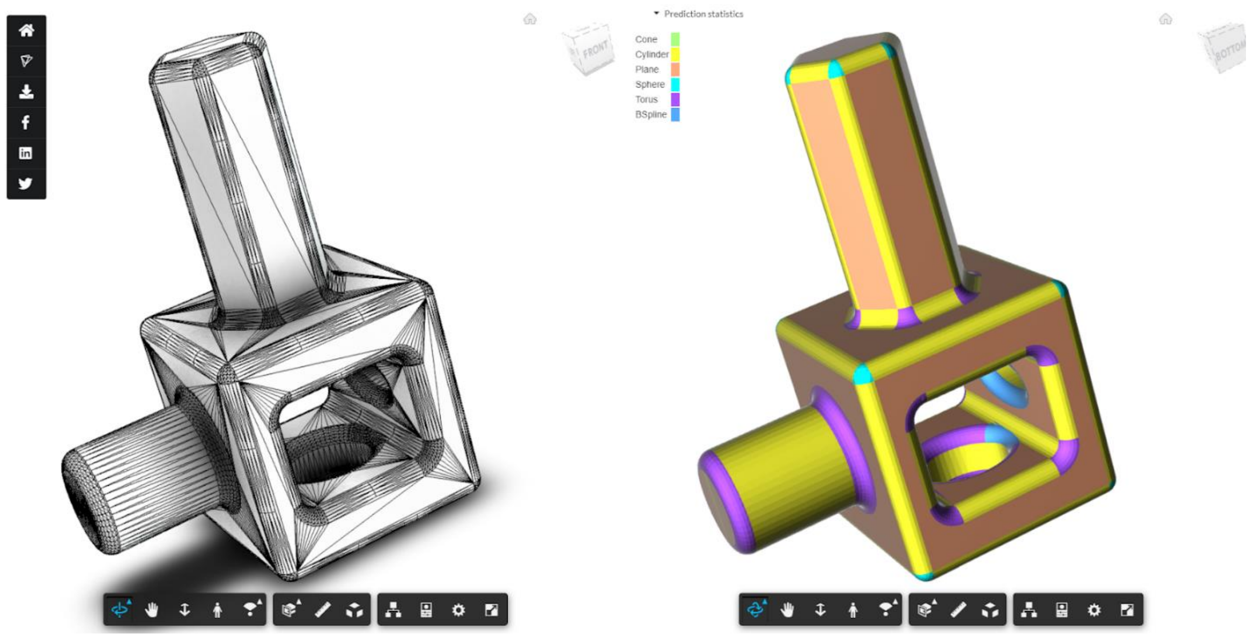
- Positive (P): Observation is positive.
- Negative (N): Observation is not positive.
- True Positive (TP): Observation is positive and is predicted to be positive.
- False Negative (FN): Observation is positive but is predicted negative.
- True Negative (TN): Observation is negative and is predicted to be negative.
- False Positive (FP): Observation is negative but is predicted positive.

	Plane	Torus
Plane	98.2%	1.8%
Torus	3.9%	96.1%

As planer triangles are much larger in size and cover more space than triangle of torus, thus our concentration was more to improve true positive of planer triangles i.e. rightly classify planer triangles, as we see from our results that model have marked 98.2% planer triangles to be planer.

7 Creation of Serverless web app using Autodesk Forge





7.1 Why a webapp?

With widespread coverage of internet and with a global average internet speed of 11.03 mbps, webapps have taken over traditional desktop software packages. Webapps have the following advantages over traditional desktop apps –

- No installation required
- Access from any device
- Data is auto saved onto the servers, making it secured with full backup
- Continuous feature updates, i.e. always use the latest software
- Effective tech support as you can provide access of your data to the support engineer in one click
- Heavy duty processing is offloaded to the app's servers, hence reducing cost of hardware

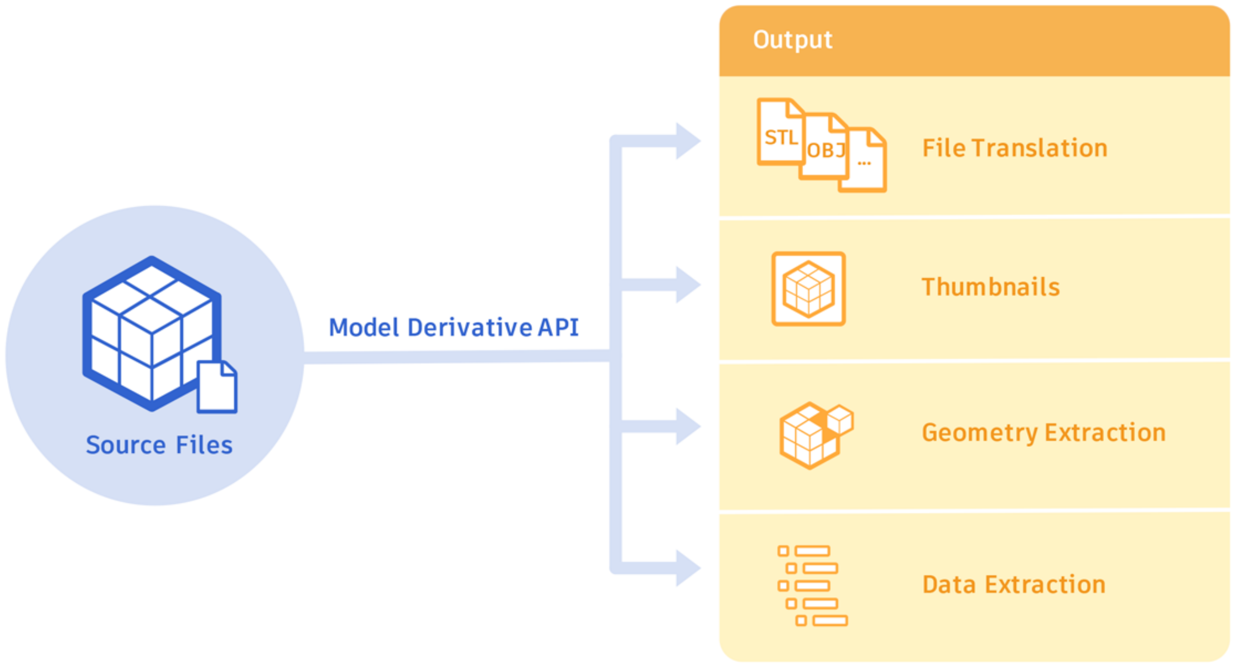
7.2 Serverless cloud technology

Serverless cloud is a solution by the cloud providers where they manage the infrastructure that is needed to host and run software makers' code. This offloads the software makers from doing operations and administrative activities to setup and manage infrastructure in the cloud. Allowing software makers to focus only on developing business logic. Due to the following benefits our company decided to completely shift to serverless computing:

- No server management is necessary
- Tremendous cost reduction as software makers only pay for exactly how much their customers consume, indirectly reducing the cost of software to the end customer
- Serverless applications are inherently scalable, i.e. apps can successfully serve a sudden increase of usage when large number of customers come online at the same time.
- Managed databases come with in-built security and backup
- Code can run closer to the end user, decreasing latency

7.3 Geometry processor for the web?

We choose Amazon web services as our cloud provider. And we used mostly all open source software stack to build up the application but, when it comes to implementing a 3D viewer and a geometry kernel for any web application, the only answer is Autodesk's Forge Platform. Just like the word "Search" has been replaced by "Google", similarly, now-a-days the term "web 3D viewer" has been replaced by "Forge viewer". These days, we, in CCTech, create almost one new web app every month based on the Forge platform.



7.4 Model Derivative API

The Model Derivative API enables users to represent and share their designs in different formats, as well as to extract valuable metadata. (Its translation functionality was previously bundled as part of the "View and Data API".)

The API offers the following features:

- Translate designs into SVF format for rendering in the Forge Viewer.
- Extract design metadata and integrate it into your app. Including object hierarchy trees, model views, and object properties, - such as materials, density and volume, etc.
- Extract selected parts of a design and export the set of geometries in OBJ format.
- Create different-sized thumbnails from design files.
- Translate designs into different formats, such as STL and OBJ.

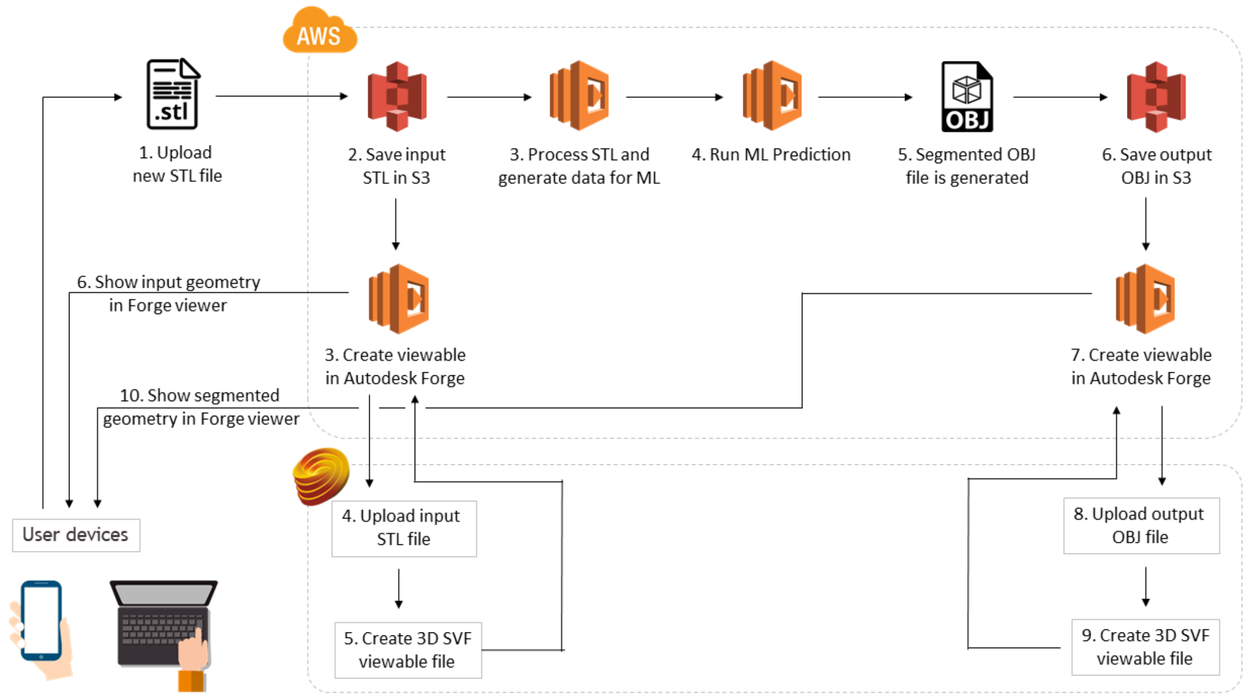
Get started by taking a look at forge Step-by-Step Tutorials to see typical Model Derivative API workflows. The API Basics and Field Guide sections will help you understand the high-level concepts, and the API Reference section will give you the information you need to refine and create your own custom flows.

7.5 ML Mesh Segmentation app

ML Mesh Segmentation - the app allows users to upload an STL file and in real-time get an output OBJ file which has all the surfaces detected. As the application in powered by machine learning techniques, the users of the app can also contribute in improving our app's surface detection capabilities in future. Social media integration has been done so that the users of the application can easily share their results with just a click.

7.6 App Architecture

Following is the architecture diagram of the application.

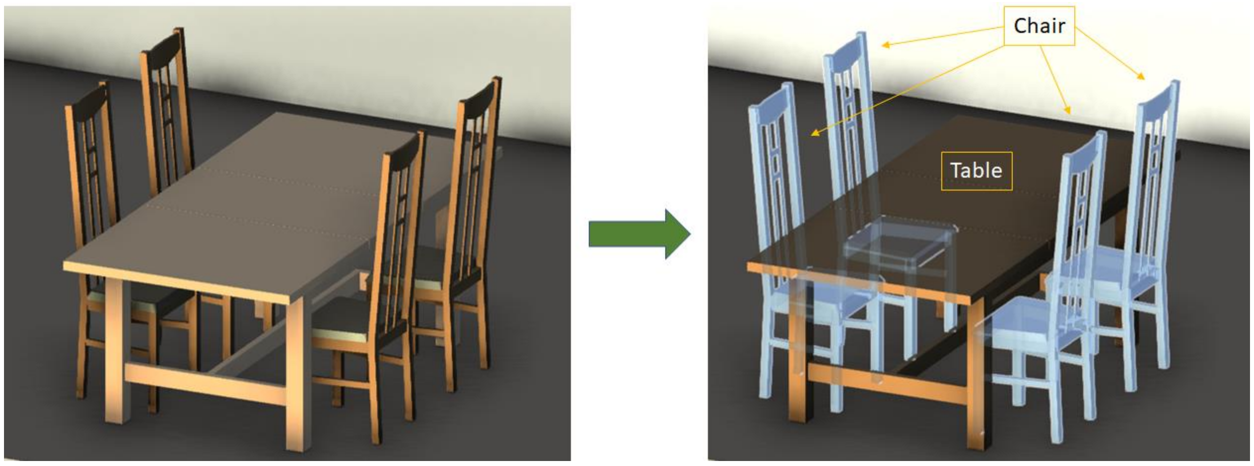

AWS
.stl
1. Upload new STL file
2. Save input STL in S3
3. Process STL and generate data for ML
4. Run ML Prediction
OBJ
5. Segmented OBJ file is generated
6. Save output OBJ in S3
6. Show input geometry in Forge viewer
3. Create viewable in Autodesk Forge
7. Create viewable in Autodesk Forge
10. Show segmented geometry in Forge viewer
User devices
4. Upload input STL file
5. Create 3D SVF viewable file
8. Upload output OBJ file
9. Create 3D SVF viewable file

- The browser app provides a UI using which the user can upload an STL file
- Uploaded STL file are saved into AWS S3 storage
- ASP.Net code runs inside AWS Serverless Lambda functions and converts the unstructured triangulated mesh information into a specialized structured columnar dataset
- Using the dataset, we innovatively run ML predictions inside AWS Lambda functions. As the ML algorithm takes structured data as input, its output is also in a structured format.
- A segmented OBJ file is generated as the output
- Output segmented OBJ file is saved in AWS S3 storage
- Using Autodesk Forge Platform services 3D viewable are created

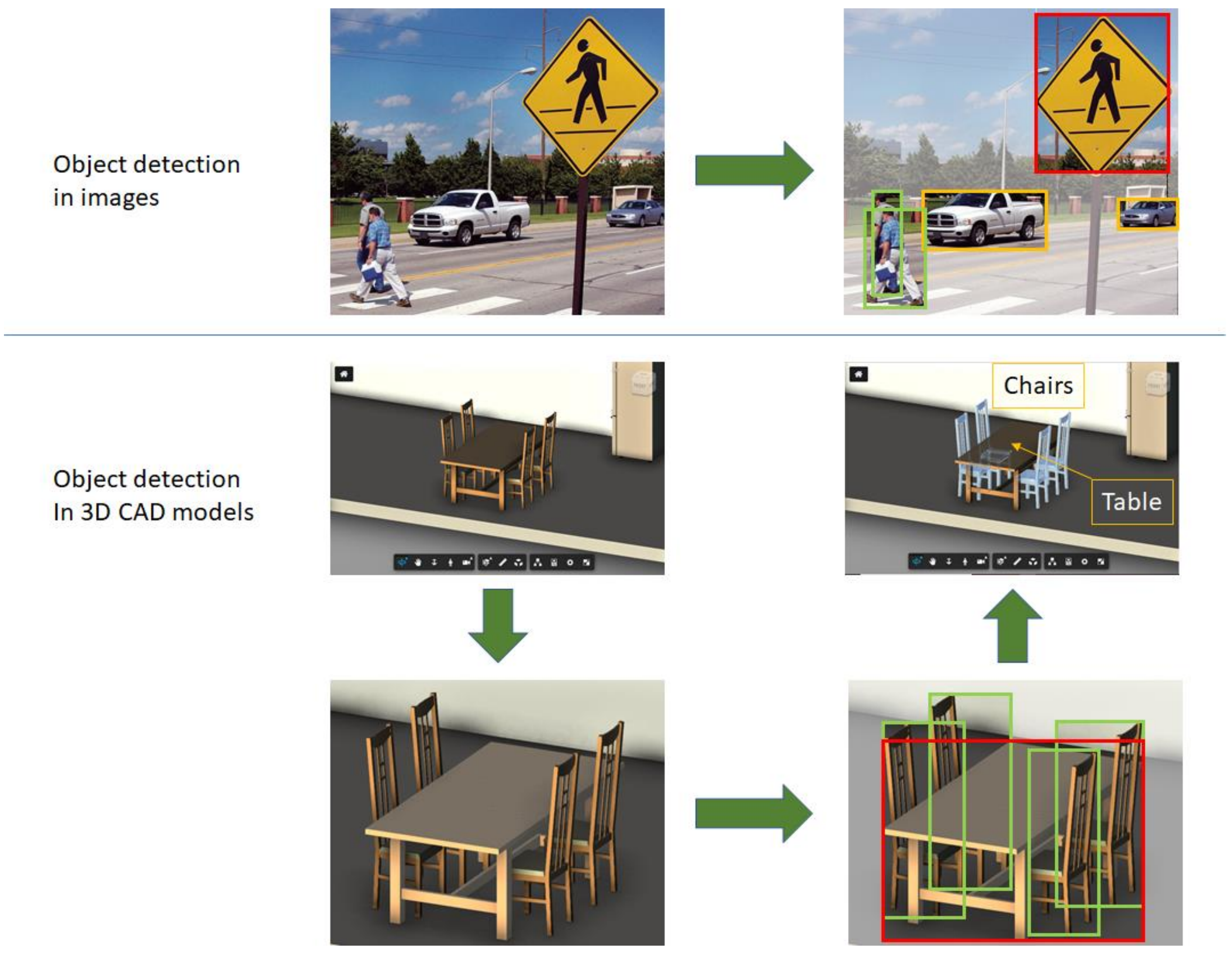
The web application is hosted at https://experiments.cctech.co.in/ml-mesh-seg. Capabilities of Autodesk Forge and AWS serverless computing platforms helped us to bootstrap with just 2 engineers to create this web app in a short timeline.

8 Phase 2 work

Our objective of phase 2 is to read a 3D CAD file and detect the objects present in it. For example, in the below CAD model, the proposed AI algorithm could be able to detect that there are 4 chairs and a table.



Today, object detection from images has got tremendous success. We decided of capturing snapshots of a 3D CAD model by opening it in Autodesk Forge Viewer and then using the proven image processing techniques to detect objects in them.



We came up with a 4-step strategy to detect objects from CAD files -

- Open 3D model in Autodesk Forge Viewer
- Capture a snapshot
- Detect objects from the snapshot using ML
- Map the objects from the snapshot to the 3D model

While brainstorming over the strategy, we soon realized that not every object of the CAD model is visible from a single snapshot. To solve this, we can go for multiple snapshots from different camera angles, but what if an object is completely enclosed inside another object? Hence, we decided to go with making one object visible at a time and taking snapshots from multiple angles.

Now, the major tasks involved in creation of this web application were –

- Opening CAD files in the cloud and taking snapshots
- Figuring out the best possible ML model to detect objects from CAD rendered images

- A strategy to map detected objects back to the CAD model

Running Autodesk Forge Viewer inside AWS's serverless Lambda function was of great help in achieving this objective. We created our own microservice API which takes the following as input and returns the links to download the created images.

INPUT

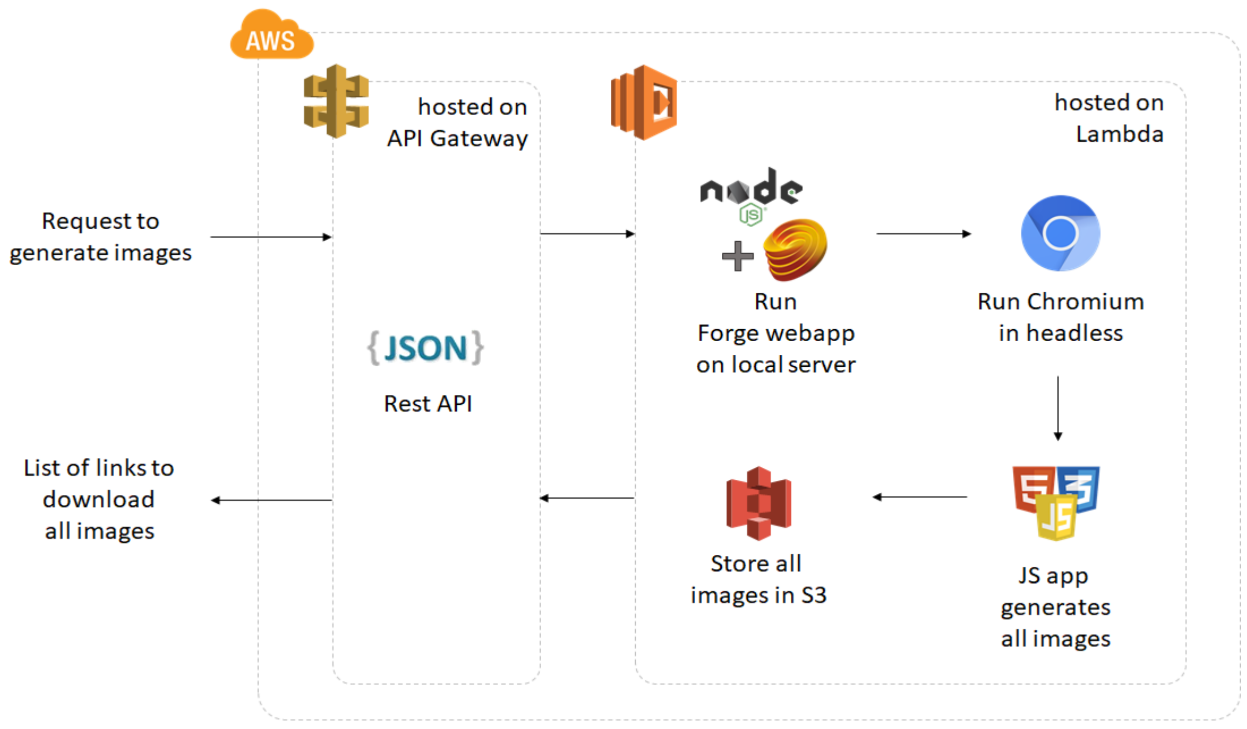
- Forge Access Token
- Forge URN
- List of images to be captured with following parameters
    - Camera angle
    - Camera position
    - List of part IDs to show/hide
- Image resolution
- Webhook (for completion)

OUTPUT

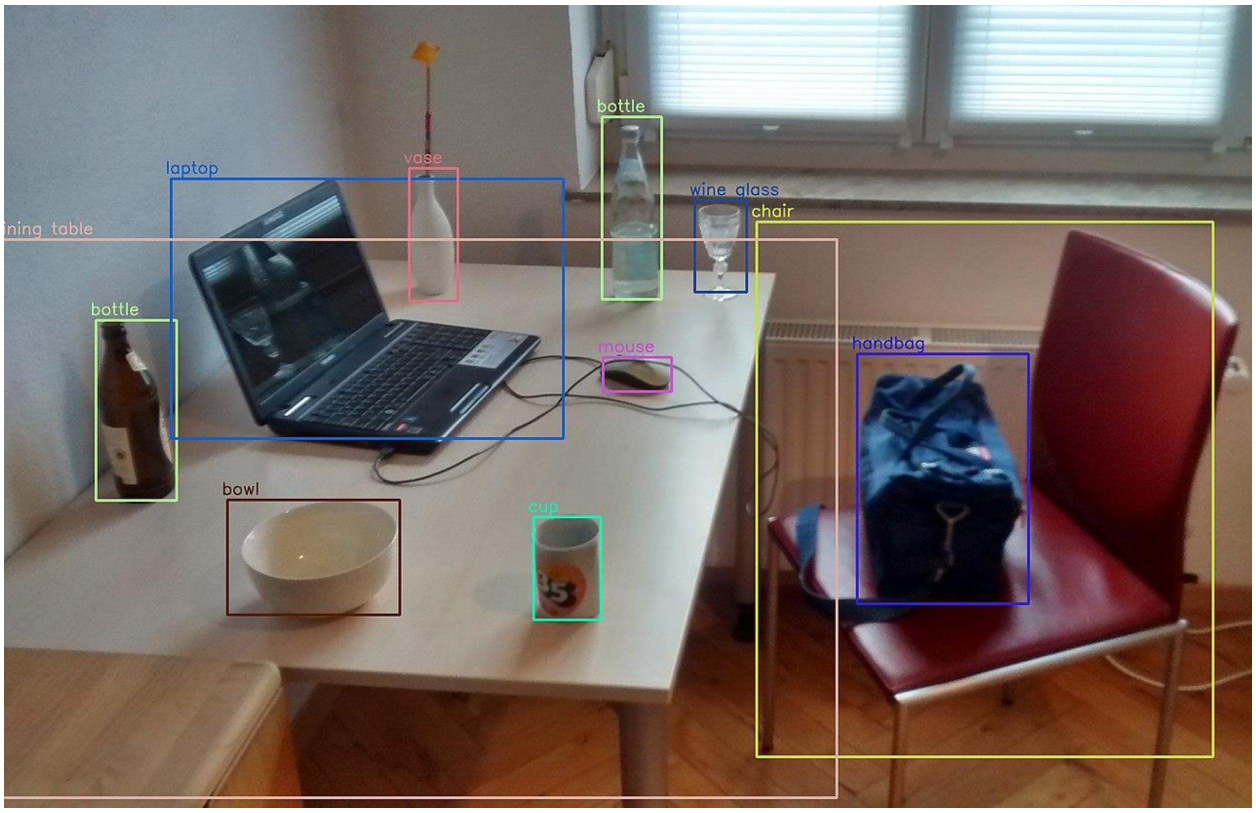
- List of images created
    - Link to download

8.1 Image capturing web service

Following is the architecture of the new web service for image capturing


AWS
hosted on API Gateway
hosted on Lambda
Request to generate images
{JSON}
Rest API
Run Forge webapp on local server
Run Chromium in headless
List of links to download all images
Store all images in S3
JS app generates all images

8.2 Object detection



Every object class has its own special features that helps in classifying the class – for example all circles are round. Object class detection uses these special features. For example, when looking for circles, objects that are at a distance from a point (i.e. the center) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed. A similar approach is used for face identification where eyes, nose, and lips can be found and features like skin color and distance between eyes can be found.

Methods for object detection generally fall into either machine learning-based approaches or deep learning-based approaches. For Machine Learning approaches, it becomes necessary to first define features using one of the methods below, then using a technique such as support vector machine (SVM) to do the classification. On the other hand, deep learning techniques are able to do end-to-end object detection without specifically defining features and are typically based on convolutional neural networks (CNN).

Since object detection in image has been advanced by many, we decided not to reinvent the wheel. We explored multiple web services options. Finally, we decided to go with Google's AutoML web services.

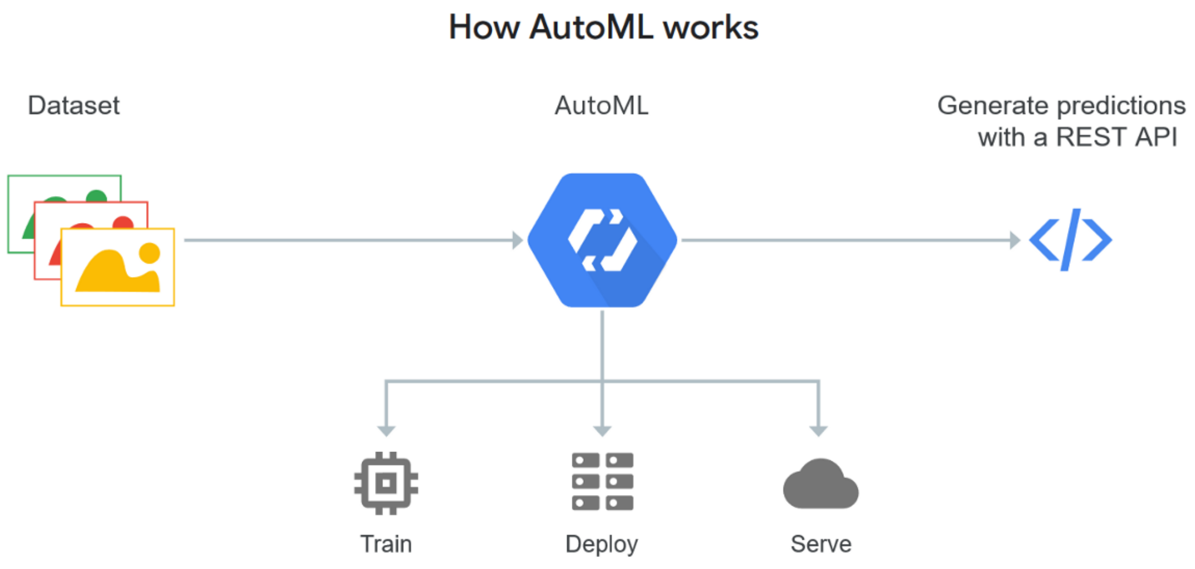
8.3 Google AutoML Webservices

Cloud AutoML makes the power of machine learning available to you even if you have limited knowledge of machine learning. You can use AutoML to build on Google's machine learning capabilities to create your own custom machine learning models that are tailored to your business needs, and then integrate those models into your applications and web sites.

Cloud AutoML Vision Object Detection enables developers to train custom machine learning models that are capable of detecting individual objects in a given image along with its bounding box and label.

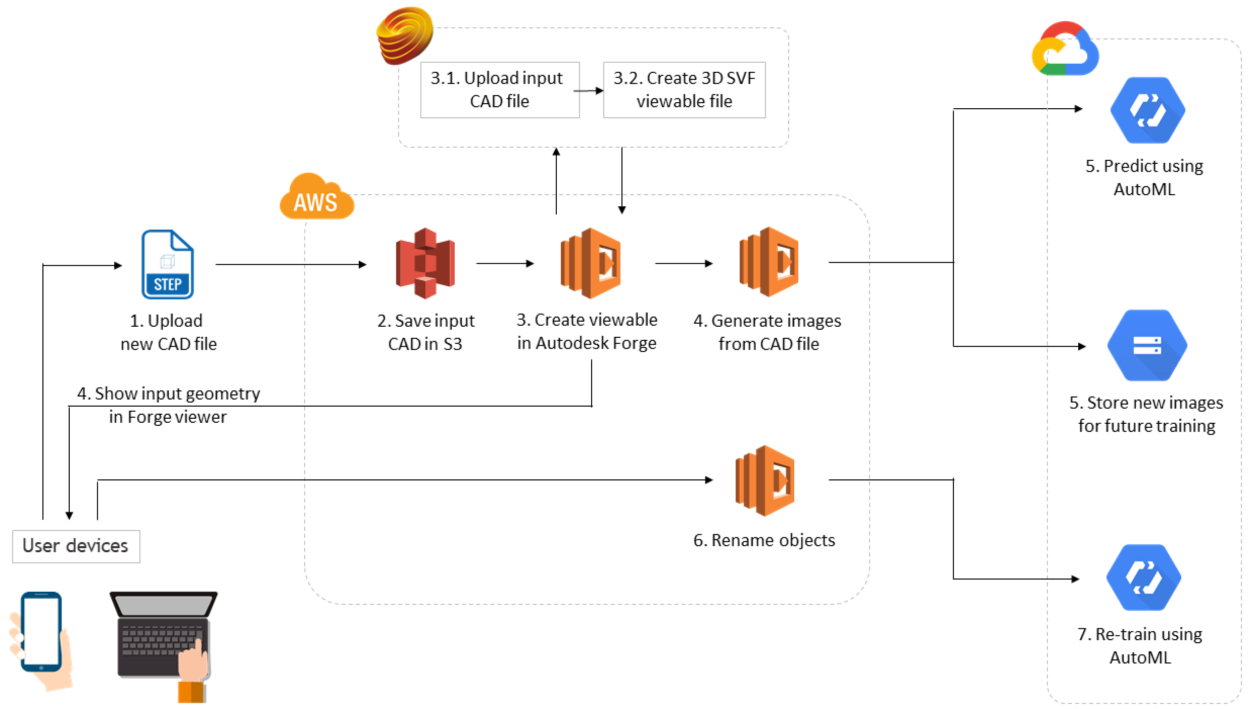
The Cloud AutoML Vision Object Detection Beta release includes the following features:

- Object localization - Detects multiple objects in an image and provides information about the object and where the object was found in the image.
- API/UI - Provides an API and custom user interface for importing your dataset from a Google Cloud Storage hosted CSV file and training images, for adding and removing annotations from imported images, for training and reviewing model evaluation metrics, and for using your model with online prediction.



8.4 App Architecture

Let's now understand the architecture of this serverless web application - AI Object Detector


3.1. Upload input CAD file
3.2. Create 3D SVF viewable file
AWS
1. Upload new CAD file
2. Save input CAD in S3
3. Create viewable in Autodesk Forge
4. Generate images from CAD file
4. Show input geometry in Forge viewer
6. Rename objects
User devices
5. Predict using AutoML
5. Store new images for future training
7. Re-train using AutoML

- The browser app provides a UI using which the user can upload a CAD file
- We take the CAD file and save it to AWS S3 storage
- We consume Autodesk Forge Platform services to create 3D viewable files for the web
- Using the in-house Rest API we generate images from CAD file
- Object detection is done on images using Google's AutoML and images are stored for future training
- If the user finds that the detected name of an object is incorrect, she gives a new name to that object
- AutoML model is re-trained with the new images and the corrected names

Do visit the web application at https://experiments.cctech.co.in/ai-object-detector. Thanks to Autodesk Forge, AWS serverless computing, and Google AutoML platforms a team of just 1 engineer was able to put up this web app in a short timeline of 1 month.