COEN 499 – Independent Study

Winter 2019

DeepDash – Deep Learning Dashboard Project

Under the guidance of Prof. Angela Musurlian

Jayant Kashyap – W1433401

Joy Sequeira – W1290264

Karthik Basavaraju – W1409883

Saurabh Somani – W1420248

Audience & Intent Behind This Project

90% of the data has been generated in the last 5 years. With this rate of gigantic amount of incoming data Machine Learning, Deep Learning, Big Data Paradigm and Artificial Technologies have become an essential need to solve the business problems in day to day life. Data in the form of images is the heart of this revolution. Studying at Santa Clara University, located at the core of silicon-valley, we decided to leverage the opportunity of building a dashboard which will address the current technology spectrum and cover the real time business problem.

DeepDash, is a deep learning dashboard which builds predictive stats from image data using 2 models: K Nearest Neighbors and VGG19. These stats are then displayed on the front end for user to analyse and develop/choose a strategy to solve the business problem he/her is working on. This first release of DeepDash is intended for people who have access to image datasets (public or private).

Table of Contents

Table of Figures

Abbreviation

ANN – Artificial Neural Networks

API – Application Programming Interface

AWS – Amazon Web Services

CNN – Convolutional Neural Network

CV – Computer Vision

DL – Deep Learning

KNN – K Nearest Neighbor

KPI – Key Performance Indicators

ML – Machine Learning

POC – Proof of Concept

REST – Representational State Transfer

RL – Reinforcement Learning

SCU – Santa Clara University

UML – Unified Modeling Language

Introduction

2.5 quintillion bytes of data is generated everyday (1 quintillion = 1 followed by 30 Zeros). Tech giants like Google, Facebook, LinkedIn, Microsoft, Nvidia, Tesla and many others are working on full throttle to leverage this big data and generate new products which has changed the aspect of daily human routine. This big data is the root element for rise of data science, machine learning (ML), deep learning (DL), reinforcement learning (RL), artificial intelligence (AI), computer vision (CV), distributed computing and DevOps. Directly or indirectly all of us have been contributing to this big data paradigm.

Have you ever wondered why did websites used to ask you to identify a store front or a tree or an advertising board at the end of your transaction? Do you really believe it was from the security purpose for your transaction? The answer is no. You were a part of user testing which was evaluating the predictive score for the machine learning model the company had built to recognize the image. With your input, the engineers would then compare the output of their models and make the necessary tweaks to their parameters used in the ML model and enhance its performance. This shows that ML, DL, AI, RL and CV were all operating on a large scale under the hood before becoming public in the early months of 2017.

So how do these machines are able to recognize these images with an accuracy preceding human vision? Why is this technology considered as a breakthrough solution? Why did it take so much time to develop these ML & DL frameworks? And finally, what is meant by the key words of Neural Network, training data, testing data and ML models?

All these questions are addressed in this project and are covered in the report going forward.

Neural Networks

Human brain is the standard example of a neural network. ML and DL models are implemented on the working principle of neurons in the human brain. We as humans are able to differentiate between a cat and a dog with ease because our brain has been trained on the parameters which distinguishes a cat from a dog. Surprised? Were you able to do the same thing when you were a year old or two years old? The answer is no. Over the years our brain has seen ‘n’ number of cats & dogs and also learnt about the factors which separates them. After this extensive training we can now confidently answer the question if a given animal is a cat or a dog.

Similarly, in the case of ML or a DL model, a Neural Network (acts as the brain of the machine) is trained with millions of images of a cats and dogs after which a ML model can achieve the same feet as that of human of recognizing a cat from a dog. This is the power of Neural Network and deep learning. What took years for us to master is now available just from a couple of Python imports and API calls. But will this neural network be able to recognize a goat if it is trained on cats and dogs? The answer would be no. Will you be able to recognize an entity you have never seen before in your first attempt? The obvious answer is no, but once you have seen it, read about it, gathered more info about it, the next time we show you the same entity you will be able to connect the dots. This is exactly how ML and DL models works. As they get trained with more and more valid data, they will be able to recognize the image with more confidence rate.

Artificial neural networks (ANN) or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a

biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called 'edges'. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

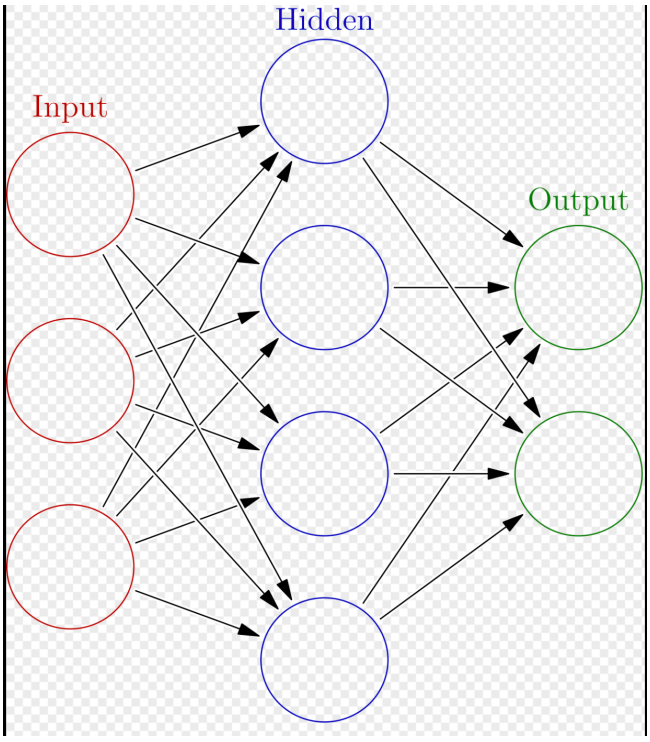

Figure 1 - Visualization of ANN at Abstract Level

Dashboards

Dashboard is a commonly used tool across the software industry. It acts as a bridge for translating technical results of a code into user-friendly front-end result. This helps to drive the client-side communications for a product. Managers, analysts or senior members in a project use the dashboard to draw insights from the output that is generated by the code written by an engineer at the backend. Dashboard is a crucial part of the "*DeepDash*" project. Since the outcome of the ML and DL models used in this project are probabilistic numbers and confidence intervals, it is very unlikely that a person without a sound knowledge about the field of data science would be able to interpret the result. Hence, the dashboard will act as a medium to bridge this gap of understanding and help the user to use the project to the fullest.

Dashboards are categorized into 3 types:

1. Operational Dashboard
2. Strategic Dashboard
3. Analytical Dashboard

DeepDash's dashboard can be classified under the first category, i.e., Operational Dashboard given the design behind it and the questions that it answers. An operational dashboard is a reporting tool that is used to monitor business processes that frequently change and to track current performance of key metrics and KPIs. Compared to other types of dashboards, the data updates very frequently, sometimes even on a minute-by-minute basis. Operational dashboards are designed to be viewed multiple times throughout the day. They are often used to monitor progress towards a target.

Below are the deciding factors that were taken into consideration while designing DeepDash's dashboard from the business requirement perspective.

1. Data awareness and time sensitive data for solving the business problem.
2. Line of business managers; business users' people who will be using the dashboard.

3. Intra-daily performance used for the identification of gaps to develop a set of metrics that will be used as the basis for the development of the dashboard.
4. Employee awareness and tracking against goals. This is important to identify what we hope to achieve by using dashboards as well as align those goals to the project's overall strategy before developing metrics.

From the technical requirement aspects, below factors were key to the design process of the dashboard.

1. Web APIs (REST), databases, spreadsheet data to maintain the state of data infrastructure.
2. Low latency, time sensitive, and real time for addressing the data latency requirements.
3. Improve visibility into multiple systems and applications to monitor the number of the data sources we are looking at and locating the data residence in our project.

KNN and VGG-19

KNN (K Nearest Neighbors) and VGG-19 are the two models which are used to implement the predictive analysis of the image in DeepDash. VGG-19 is a convolutional neural network that is trained on more than a million images from the ImageNet database and is a deep learning model. The network is 19 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224. We can use 'classify' to classify new images using the VGG-19 network.

It is recommended to use a pre-trained model while building your own instance of VGG-19. A pre-trained model has been previously trained on a dataset and contains the weights and biases that represent the features of whichever dataset it was trained on. Learned features are often transferable to different data. For example, a model trained on a large dataset of bird images will contain learned features like edges or horizontal lines that you would be transferable your dataset. Pre-trained models are beneficial to us for many reasons. By using a pre-trained model, you are saving time. Someone else has already spent the time and compute resources to learn a lot of features and your model will likely benefit from it.

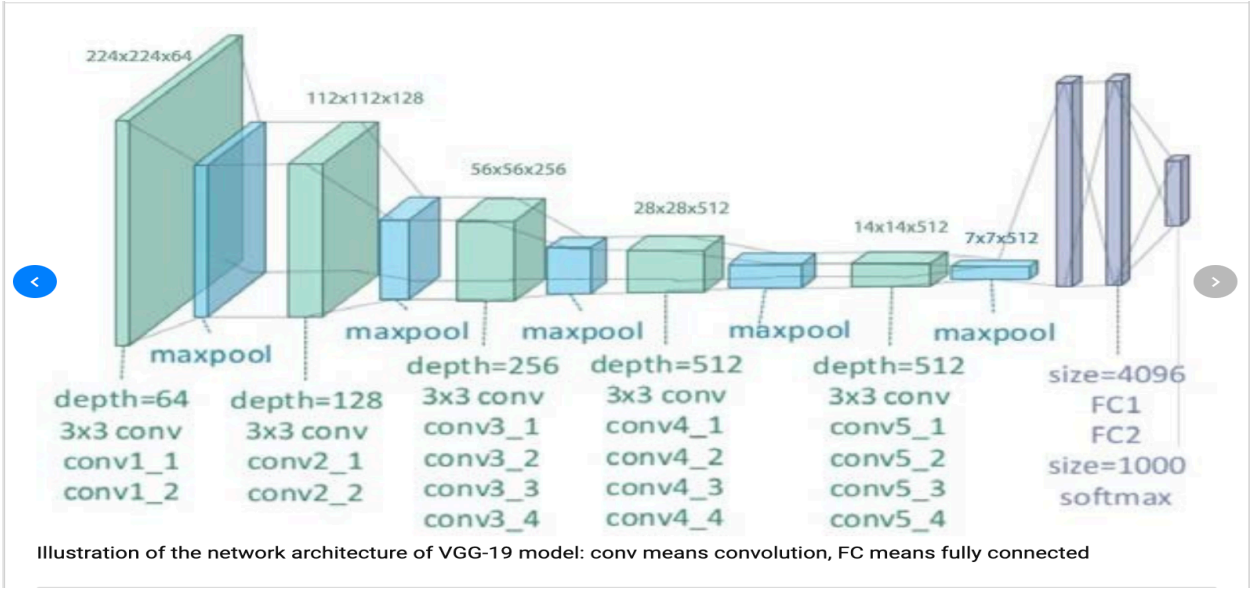

Figure 2 - VGG-19 Architecture

KNN is a supervised machine learning algorithm used for classification. The k-NN algorithm is among the simplest of all machine learning algorithms. KNN is the most commonly used pattern recognition algorithm in every machine learning project for different purposes. In real time, KNN is used to discover the input data and help the data scientists make some sense out of it. This helps in establishing relationships among the columns and the values inside it. Once the relationship has been figured on the basis of mathematical scores of distances, we can choose what ML technique or algorithm best suites our context. It is very crucial in the field of data science. If you don't know what the problem is in first place, how do we expect to come up with the solution. This is where KNN is most helpful.

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. In k-NN regression, the output is the property value for the object. This value is the average of the values of k' nearest neighbors. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until classification.

Both for classification and regression, a useful technique can be used to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

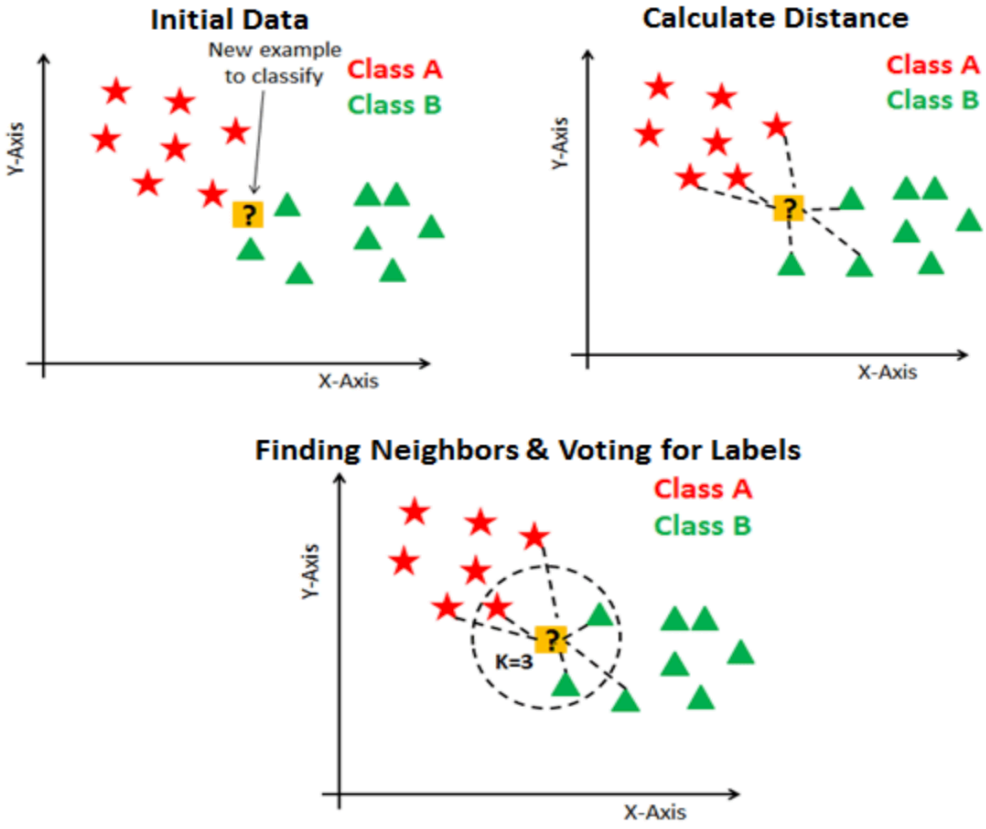

Figure 3 - K Nearest Neighbors

Project LifeCycle: Planning, Execution & Learnings

We undertook this project to get hands on experience on implementing machine learning and deep learning models with RESTful web services and delievering a end to end product. Since there is no available provision in SCU's current coursework for the above requirement, we approached Prof. Angela Musurlian to guide and provide us the resources for undertaking this project.

Planning

The time of completion available to the team for this project was approximately 11 weeks, i.e, from 01/07/2019 to 03/22/2019. In a team of 4, we initially planned to have the system deployed on AWS and have the graphs displayed on real time at the front end with the python backend engine and middleware API calls.

We planned to use Git as the vaersion control and use Pep8 as the coding standard for variable and function naming. The middleware was planned to be developed in Node.js and creating RESTful calls from it. React to be used as the front end framework.

Prof. Angela Musurlian, advised us to keep a track of challenges and changes encountered while implementing the final version of the project. We have documented them accordingly in this report.

Execution

We were successfully in implementing a POC with majority of the requirements as planned. Major changes involved use of Flask (Python Web Framework) instead of Node.js to build API calls as it was easy to integrate with the Python back end engine. Second major change was to have the system deployed locally instead of AWS because of the lack of financial resources. Image datasets used for the training and testing purpose are *FlickrLogos-32 Dataset* and *Animal* dataset consisting of cats, dogs and pandas. Displaying graphs and bars on the front end were

rescheduled for future scope implementation (deliverable by 06/13/2019) due to time constraints.

Learnings

This project and the guidance of Prof. Angela Musurlian has helped us learn not only the technological aspect while designing and building product but also the real time factors of time, resources, change of business requirements and the importance of deliverable results . Below are some key points which reflect these learnings in brief.

1. Understanding the essence of gathering business requirements and laying down the expecatations for the end product clearly at the start of the project.
2. Drawing up schemas, UML's and creating multiple paths to implement the same functionality in order to reduce redundancy, figuring out the optimal algorithm/path to implement the code.
3. Establishing coding standards and version control software to be used for the project. Being ready to change the tech stack as per the coding comfort and the readiness to accept to add, drop or alter features from the initial plan for delivering the POC.
4. Keeping track of availability of teammates and their time, balancing academic grades without ignoring other courses and maintaining the financial log to have a real time estimate of the resources for delivering the project.
5. Reading research papers, effective use of Stack Overflow, Google, GitHub and Reddit to lookup queries related to code and solving the problems or finding a work around them.
6. Making notes for future implementations and learning from the initial pitfalls of the project.

Future Scope

Going forward for Spring 2019 quarter, one of the team members, Jayant Kashyap, will be continuing the project and implementing the real time bars and graphs displayed on the front end. Another major update would be regarding the customized model building. As of now user can make the decision of choosing either KNN or VGG-19 as the model which he/she needs to be used for prediction. In the next version, user will have the privilege to choose the number of layers he/she would like to have in the neural network to be built. Also there would be introduction of Computer Vision concepts which will be integrated into the existing project and will be specified by Jayant in his abstract for the Spring 2019.

References

1. https://github.com/jayantkashyap/DeepDash
2. https://www.mathworks.com/help/deeplearning/ref/vgg19.html;jsessionid=f0a7f5bba1fb4b1b1f1f57286082
3. https://www.klipfolio.com/choosing-the-right-dashboard-report
4. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
5. https://en.wikipedia.org/wiki/Artificial_neural_network
6. https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn
7. https://www.researchgate.net/figure/llustration-of-the-network-architecture-of-VGG-19-model-conv-means-convolution-FC-means_fig2_325137356
8. https://www.kaggle.com/keras/vgg19/home
9. http://www.multimedia-computing.de/flickrlogos/