George Mason University

DAEN 690

Summer 2019

**Image Detection of Simpson’s characters**

Manju Prasad

Ravi Rane

Yinchen Niu

Zegang Liu

**Table of Contents**

[**1**  **Abstract** 3](#_Toc14634996)

[**2**  **Problem Definition** 3](#_Toc14634997)

[**2.1** **Interesting Inspiration** 3](#_Toc14634998)

[**2.2** **Problem Definition** 4](#_Toc14634999)

[**2.3** **Scope** 4](#_Toc14635000)

[**2.31** **Key Objectives** 4](#_Toc14635001)

[**2.32** **Architecture Exploration** 4](#_Toc14635002)

[**2.33** **Stretch Scope** 5](#_Toc14635003)

[**3** **Data Exploration** 5](#_Toc14635004)

[**3.1** **Data Context** 5](#_Toc14635005)

[**3.2** **Dataset Composition** 6](#_Toc14635006)

[**3.3** **Data Quality** 7](#_Toc14635007)

[**4** **Technology Exploration & Setup** 8](#_Toc14635008)

[**4.1** **Technology and Tools** 8](#_Toc14635009)

[**4.2** **Assumptions and Risks** 8](#_Toc14635010)

[**5** **Development: Analytics & Algorithms** 9](#_Toc14635011)

[**5.1** **Neural Network Architectures** 9](#_Toc14635012)

[**5.2** **Deep Learning Research** 9](#_Toc14635013)

[**5.21** **Activation Functions** 9](#_Toc14635014)

[**5.22** **Cost Functions** 10](#_Toc14635015)

[**5.23** **Gradient Descent Approach** 10](#_Toc14635016)

[**5.24** **Preventing Overfitting** 10](#_Toc14635017)

[**5.25** **Initialization** 10](#_Toc14635018)

[**5.26** **Layers** 11](#_Toc14635019)

[**5.3** **Convolutional Neural Network (CNN)** 11](#_Toc14635020)

[**5.4** **Region-Based Convolutional Neural Network (R-CNN)** 11](#_Toc14635021)

[**5.5** **You Only Look Once (YOLO)** 12](#_Toc14635022)

[**5.6** **Single Shot Multi-Box Detection (SSD)** 14](#_Toc14635023)

[**6** **Visualizations** 16](#_Toc14635024)

[**7** **Stretch Goals** 16](#_Toc14635025)

[**7.1 Simpson character’s search in Image dataset (can we change title to indicate which stretch goal this implements?)** 16](#_Toc14635026)

[**7.2 Object detection in videos** 17](#_Toc14635027)

[**7.3 Cloud Computing** 17](#_Toc14635028)

[**8** **Acknowledgement** 18](#_Toc14635029)

[**9** **References** 18](#_Toc14635030)

### **1 Abstract**

Object detection and image classification are at the forefront of computer vision technologies with varied applications in Society today. Recent advancements in facial detection based surveillance in the Security industry, pedestrian/object detection in self-driving cars built by the Automobile industry, automated valuation of properties in the Real Estate industry, are just some examples of the varied types of applications for this cutting edge technology.

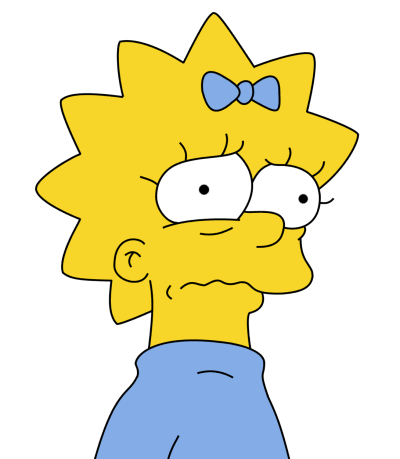
Utilizing the popular 30-season show The Simpsons, this project implements object detection and image classification for characters from the Simpsons series. Various Deep Learning architectures like Convolutional Neural Network (CNN), Faster Region-based Convolutional Neural Network (R-CNN), You Only Look Once (YOLO), and Single Shot Multi-box Detection (SSD) have been explored. As a secondary focus, this project has applied one of the image detection models to videos and explored the use of Cloud computing to train and execute models. Visualizations and a User Interface (UI) to enable a real-time data feed to model execution have been included.

The underlying research and concepts can be extended for use in a commercial enterprise as described above.

### **2 Problem Definition**

### **2.1 Interesting Inspiration**

Can you tell the difference between Maggie Simpson and Lisa Simpson? Was that Lenny Leonard or was that Carl Carlson?

### **Problem Definition**

Our first goal is to classify Simpson characters from images and from videos. Secondly, we aim to extend upon this functionality to successfully identify Simpsons characters when multiple characters are in an image. The model that we develop should be able to consistently identify characters and successfully draw a bounding box around each. Doing so will require a quality interface to perform the classification.

### **Scope**

### **2.31 Key Objectives**

* Prepare data through data cleaning and preprocessing
* Explore classification techniques for image classification using Neural Network algorithm techniques, such as a Convolutional Neural Network (CNN)
* Employ techniques, such as a Faster Region-Based Convolutional Neural Network (Faster R-CNN), You Only Look Once (YOLO), and Single-Shot Multi-Box Detection (SSD) for image tagging in frames with multiple Simpsons characters
* Visualize model performance metrics possibly using tools like TensorBoard.
* Explore the use of open-source tools and libraries like GitHub, Jupyter notebook, Keras, and TensorFlow to aid development of image detection model.

### **2.32 Architecture Exploration**

|  |  |
| --- | --- |
| **Problem** | **Architectures Explored** |
| Image Classification | * Convolutional Neural Network (CNN) * Hybrid RCNN |
| Object Detection | * Faster Region-based Convolutional Neural Network (R-CNN) with different hyperparameters * You Only Look Once (YOLO) * Single Shot multi-box Detection (SSD) |

**INSERT EXPLANATION for why Hybrid CNN-ELM (for Image Classification) and Mask R-CNN (for Image Segmentation) are not being used.**

### **2.33 Stretch Scope**

* Construct a user interface (UI) to execute the model and perform Simpsons character detection/classification.
* Standardize our framework to support image classification/object detection of any object, not just Simpsons characters, when a dataset for that object is provided.
* Identify Simpsons characters in a video at pre-defined intervals.
* Explore the use of cloud computing – specifically, AWS Sagemaker – to aid in development of ML model. Because using AWS costs money, usage of AWS will depend on whether AWS Educate credits are sufficient for the scope of this project.

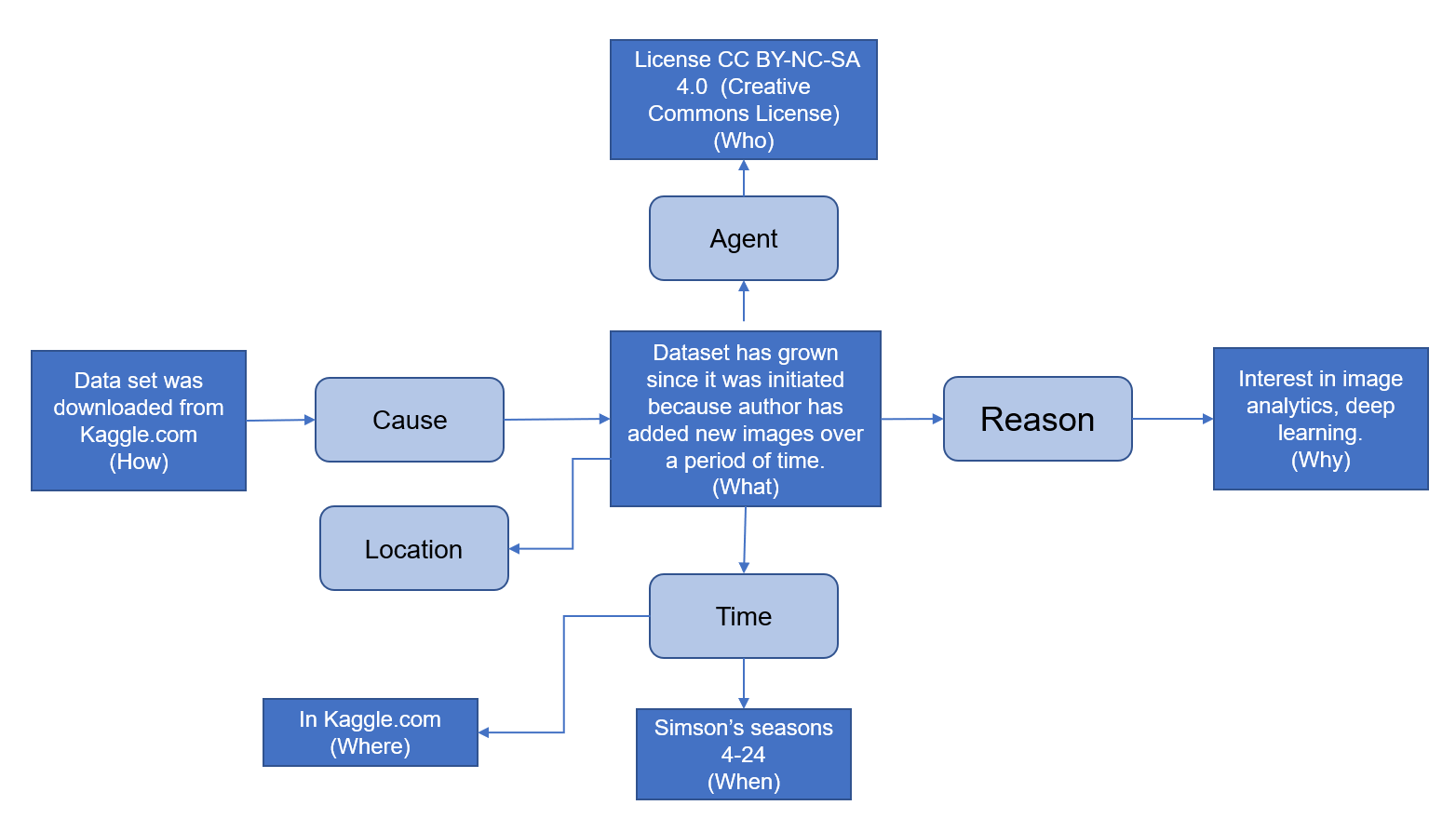
### **3 Data Exploration**

### **3.1 Data Context**

Our team was interested in working on an image analytics project. This aligned strongly with our Professor’s goals for the Capstone project in DAEN 690, for such an image analytics project was outlined in Professor’s Suggested Projects. Meanwhile, in searching for an appropriate dataset for this image analytics project, we sought a large collection of images organized around a common theme, yet clearly diversified into at least several different groups. The Simpsons is a very popular, televised cartoon that has aired 30 seasons of content. It consists of upwards of 40 characters, including many commonly featured characters, like Bart Simpson, Marge Simpson, and Krusty the Clown, among others. These characters are hallmarked by their distinctive appearances and eccentric behaviors, making them optimal for an image analytics project.

The website [Kaggle.com](https://www.kaggle.com/alexattia/the-simpsons-characters-dataset) offers an extensive dataset related to image detection of Simpson’s characters. Their image dataset has 20 character-delineated folders with anywhere between 400 and 2,000 pictures in each and credibility as evidenced by its 53,700 views, 7,274 downloads. It contains frames spanning from season four to season 24. All of the images are a collection of video frame captures from Simpson’s episodes, and some dataset images even feature multiple characters in a single frame. We have used this dataset build a model to distinguish between Simpsons characters.

Below is a Data Context representation of our data:



Number images

### **3.2 Dataset Composition**

Three discrete files in Simpson’s characters dataset exist as follows…

1. File simpson-set.tar.gz: an image dataset consisting of the 20 aforementioned folders
2. File simpson-test-set.zip.: a preview of the image dataset
3. File annotation.txt: an annotation file responsible for bounding boxes for each character

**General Dataset Information**

URL: <https://www.kaggle.com/alexattia/the-simpsons-characters-dataset>

Title: The Simpsons Characters Data

Authors: alexiattia

Publication Date: June 6, 2017

File License Number: "CC BY-NC-SA 4.0"

Text: image data

Tags: deep learning, image data

### **3.3 Data Quality**

Prior to proceeding on data conditioning, we want to provide an overview of steps taken to ensure high data quality.

To ensure completeness, characters are only being classified when there are more than 100 images of that character available in the dataset. However, it is worth noting that due to the organization of the Kaggle dataset, images could have multiple characters, meaning that the character of interest could be either prominently featured in the frame or solely present in the background.

Meanwhile, as discussed under Data Context and Dataset Composition, each image set consists of a mixture of hundreds to thousands of images. Due to the total count of images, it virtually impossible to visually identify uniqueness.

With regards to accuracy, this has only been determined in the aftermath of algorithm execution and is discussed under the description of each individual algorithm.

We guarantee atomicity because each training set contains images that are classified and grouped by character while having a provided image level annotation that properly applies boxes to characters in each image.

In relation to conformity, the dataset provided conforms to one image classification model that is required to run.

Upon considering the combination of the above, we believe that the data quality is good.

**WE LIKELY NEED A LITTLE MORE EXPLANATION IN THIS SECTION.**

**3.4 Data Conditioning**

For existing data provided through Kaggle, we employed existing functions, such as **INSERT FUNCTIONS HERE**, in the Keras and PyTorch libraries to read images and preprocess them for classification and detection. This has been sufficient to accomplish all core image classification and object detection tasks. However, in relation to project stretch goals, there may be insufficient images with multiple characters for tagging. Hence, the team may have to generate additional frames with multiple characters.

Should it be necessary for new data or data obtained from outside of Kaggle, we will use the label\_data.py function, obtain cropped pictures, and then classify each image when applying algorithms. When preprocessing these images, we first adjusted the sample size to ensure that the entire image collection is in the same sample size. The labels for the various characters are then converted from names to numbers, and the dataset is divided into training set and test set. For the core project scope, no other data sources have needed to be considered. This procedure will become more pertinent for the stretch goals section of the project.

### **4 Technology Exploration & Setup**

### **4.1 Technology and Tools**

|  |  |
| --- | --- |
| **Function** | **Technology/Tool** |
| File Repository | GitHub, GitHub Desktop, GitLab, OneDrive |
| Development | Jupyter Notebook |
| Computing | ARGO, AWS |
| Primary Development Language | Python 3.7.2 |
| ML Libraries | TensorFlow, TensorBoard, Keras, PyTorch |
| Agile Board | YouTrack |
| Collaboration | Blackboard, Slack, WhatsApp, Cisco Webex |

### **Assumptions and Risks**

**Assumptions**

* Images in the dataset are distinct enough to train an image recognition model.
* Quantity of images in each dataset is sufficient to meet the minimal threshold to train a model that can predict with good accuracy.

**Risks**

* Computing power required to run image detection code can be high. There is a risk that the programs will take too long to run. In many cases, ARGO has been unable to perform image recognition on more than a few characters due to limits on memory consumption when running. This impacts the amount of time the team has to test and implement, as well as the success of the model in correctly identifying many of the Simpsons characters. This could maybe be mitigated by using GMU’s ARGO cluster.
* Scope includes a few aspirational items whose completeness will be directly dependent on available time. Prioritizing tasks and communicating with team members and professors is in full swing to mitigate this risk as much as possible.

### **5 Development: Analytics & Algorithms**

This represents the core project scope.

### **5.1 Neural Network Architectures**

As part of our core project development scope, an understanding of six neural network architectures is necessary. This understanding allows us to compare and contrast the accuracy, performance and other metrics associated with these various architectures. These six neural network architectures are…

1. Convolutional Neural Network (CNN)
2. Faster R-CNN (with different hyper parameters)
3. YOLO (You only look once)
4. Single shot Multi-box detection (SSD).

### **5.2 Deep Learning Research**

Before diving into these architectures, there are some basic concepts of deep learning with which we familiarized ourselves.

### **5.21 Activation Functions**

Activation functions are responsible for transforming the weighted inputs to a neuron and determining if the resulting output is “activated” or not. The simplest type of an activation function is the step function (Perceptron), which can input and output binary values only. A level above the step function is a Sigmoid, which can output any value between 0 and 1. Third, tanh-based activation functions are similar to Sigmoids, but faster and can return values between -1 and 1 (scaled sigmoids). And finally, the Rectified Linear Unit (ReLU) returns 0 if the output is negative and the output itself in all other cases.

### **5.22 Cost Functions**

Cost functions are used to quantify difference between output received from a neuron versus output expected to be received. The quadratic cost function is similar to the Mean Squared Error metric used in Linear Regression models. Alternatively, there is Cross-Entropy cost function. This log-based function enables faster learning when difference between received and expected values is high.

### **5.23 Gradient Descent Approach**

To learn from a cost function and make changes to rectify the error, a gradient descent approach can be used to reduce or eliminate the magnitude of error. Learning rates determine the size of step to correct the error. Batch sizes sample input data to feed one run of a network. Second-order calculations use the acceleration or momentum of previous steps to adjust the size of next step. In essence, learning rates, batch sizes, and second-order calculations are all Gradient descent related features that can be used to tune a model.

### **5.24 Preventing Overfitting**

Meanwhile, to prevent overfitting of the model, techniques like L1/L2 based normalization, Dropout (where a subset of neurons are dropped), and artificially expansion of data can be used.

**THIS EXPLANATION LIKELY NEEDS TO BE EXPANDED**

### **5.25 Initialization**

With regards to initialize weights, bias and other outputs, Glorot-normal and Glorot-uniform values can be used.

**THIS EXPLANATION LIKELY NEEDS TO BE EXPANDED**

### **5.26 Layers**

Various types of layers can be used to build a neural network. In a Dense layer, all neurons in one layer are fully connected to neurons in the next layer. Alternatively, in a Softmax layer, a class probability score is output based on generated weights. Meanwhile, in a Max Pooling layer, the size of an image is greatly reduced by grouping pixels, hereby flattening the image. In this light, a Max Pooling layer can consolidate a two-dimensional or three-dimensional array into a single dimensional structure. Lastly, convolutional layer, the basis for Convolutional Neural Networks, learn to identify shapes independent of location in the image.

### **5.3 Convolutional Neural Network (CNN)**

A Convolutional Neural Network (CNN) is a multilayer perceptron specially designed for recognizing image contents. This network structure is highly invariant to translation scale slant or covariant deformation. It is the basic algorithm for our other algorithms and hence assumes a foundational role in this project. In this project, the CNN is the mastermind behind image classification. The CNN sifts through the dataset and classifies each input image into a category. The computer then treats each input image as an array of pixels. According to the image resolution, we can see **h** \* **w** \* **d** (h = height, w = width, d = size) (Prabhu, 2018).

For the deep learning CNN model, each image will go through a series of convolution layers, as well as kernel, pooling and full-connected layers (FC). We can add many convolutional layers and flatten the output into the FC Layer. At last, through an activation function, the results can be output to a class, and the images are classified.

**THIS EXPLANATION SHOULD BE CLEANED UP A BIT.**

### **5.4 Region-Based Convolutional Neural Network (R-CNN)**

Our Region-Based Convolutional Neural Network (R-CNN) combines region proposals with CNNs, hereby using high-capacity convolutional neural networks to bottom up region in order to locate key features and perform image segmentation (Girshick, Donahue, Darrell, & Malik, 2014). R-CNN can also be understood as regions with CNN features.

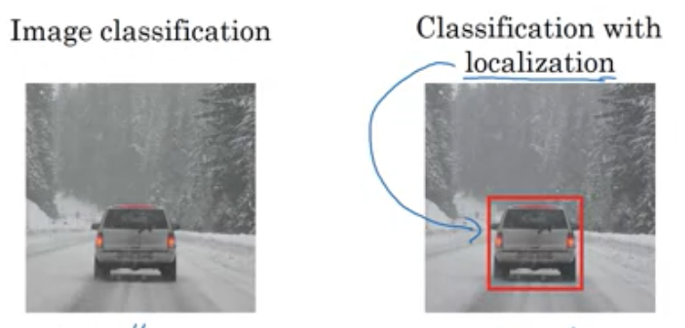
### **5.5 You Only Look Once (YOLO)**

#### **Architecture Description**:

(May want to compress below content and add comparison of different YOLO versions)

This is an architecture that tries to overcome some of the limitation of segmentation-based and region-based architectures for image detection. The algorithm works with convolutional nets that perform image classification and image localization.

A convolutional net can perform image classification. This can, for example, be trained to return a class “car”. Localization is where, in addition to classification, the algorithm returns coordinates associated with where in the image a particular object was detected. This is also referred to as a bounding box. This bounding box consists of four sets of values: bx, by, bh and bw. Bx and by are the cartesian coordinates of the top left corner of box, bh is height, and bw is width of object. The output y in case of localization consists of these elements, the first of which is the probability associated with the classification, the next four pertaining to the bounding box, and the rest being binary class variables that indicate the type of object that has been found.



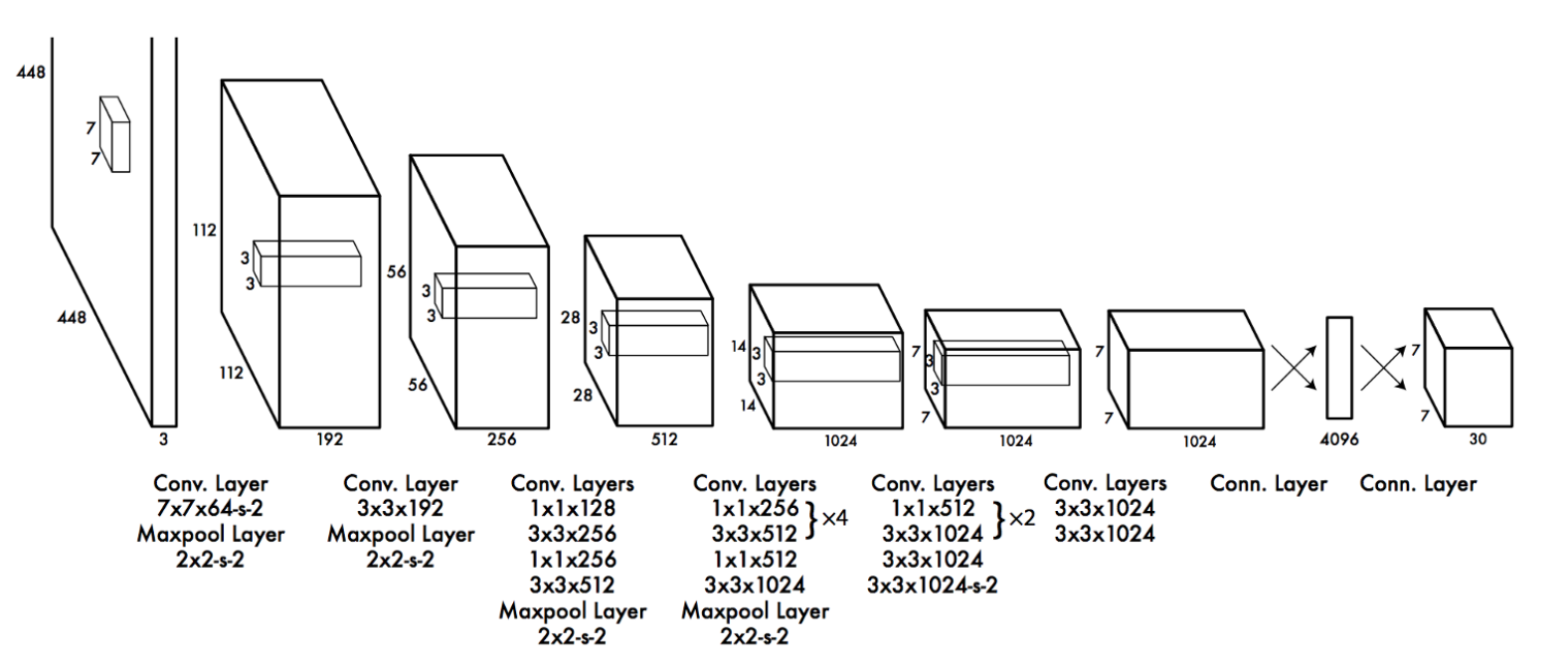
Localization can also be extended for landmark detection, where multiple coordinates associated with a part of an object. For example, the location of the eyes of a person can be returned by the algorithm.

YOLO algorithm works by splitting an image into a grid. The algorithm checks to see whether each cell is the location of the midpoint of an object. If the cell represents an object midpoint, that cell then becomes responsible for arriving at the bounded box for that object. One convolutional layer is used for all cells of the grid, which translates to efficiency and faster detection. As a direct result, YOLO can be used for real time video-based detection. In the following example, the image is of 100 x 100 x 3 size, but the output will be 3 x 3 x 8 (3x3 because we chose that as grid size and 8 to account for probability), four bounding box values, and three classes (because this algorithm tries to classify three types of objects). If the detected image spans beyond current cell, the height and width of detected object can return values greater than one.

Evaluation of object localization is done via an Intersection over Union (IoU) calculation, which tries to determine how much of the image is present in the calculated box boundaries. Typically, IoU values below 0.5 are discarded. This calculation is also used to aid in non-maximum suppression, where multiple detections of the same object are resolved, and the best box is retained.

In scenarios where midpoints for two or more objects exist in the same cell, anchor boxes can be used. Anchor boxes are predefined shapes used by the algorithm in which to fit objects. The ‘y’ output is extended to return all object matches found (number of anchor boxes = number of possible objects that can be detected)

Make sure this is YOLO3 arch. Write about YOLO versions and benefits



(add source: <https://arxiv.org/pdf/1506.02640.pdf>)

#### **Algorithm Implementation**:

An implementation of the YOLOv3 architecture called DarkNet built using C by YOLO authors is available as open source software. A Python implementation of the same is available on GitHub (provide references). This Pythonic implementation converts the original C based model build logic to Python and enables training of new datasets.

Using this GitHub repo as base, the dataset containing Simpson images and annotations was trained. Multiple modifications were made to the codebase to get it running on ARGO. Also, annotations file had to be modified to a format that YOLO could input.

Hyperparameters were adjusted to reduce validation loss. And the final model (with best fit) had the following hyperparameter settings (ADD)

TensorBoard was implemented to provide model performance metrics and model comparison. Since ARGO lacks a UI interface, the TensorBoard event outputs were transferred to a local machine and evaluated there. (add screenshots)

#### **Challenges faced**

OOO errors

Long execution times

OSError – unable to open H5 file (file lock unavailable)

Different nodes – different training times

Multi GPU failure

Accuracy of Image detection and video detection (test) – adequate sample images? Quality of images being trained on? Issues with Homer?

#### **Next Steps (with additional time)**

### **5.6 Single Shot Multi-Box Detection (SSD)**

The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes. This is then followed by a non-maximum suppression step to produce the final detections. The early network layers are based on a standard architecture used for high quality image classification (truncated before any classification layers), which is a base network. We then add auxiliary structure to the network to produce detections based on the following key features:

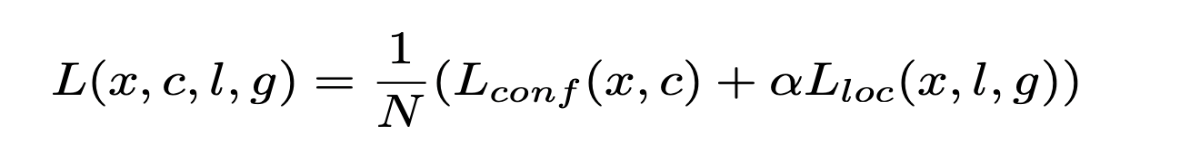
**Multi-scale feature maps for detection**

We add convolutional feature layers to the end of the truncated base network. These layers decrease in size progressively and allow predictions of detections at multiple scales. The convolutional model for predicting detections is different for each feature layer (cf Overfeat[4] and YOLO[5] that operate on a single scale feature map).

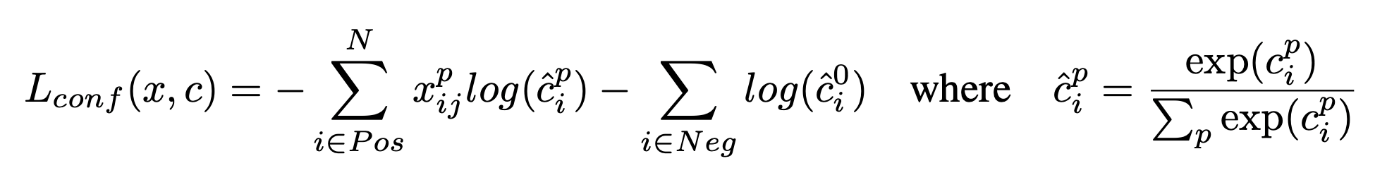
**Convolutional predictors for detection**

Each added feature layer can produce a fixed set of detection predictions using a set of convolutional filters. These are indicated on top of the SSD network architecture.

**Training objective** The SSD training objective is derived from the MultiBox objective but is extended to handle multiple object categories. The overall objective loss function is a weighted sum of the localization loss (loc) and the confidence loss (conf):

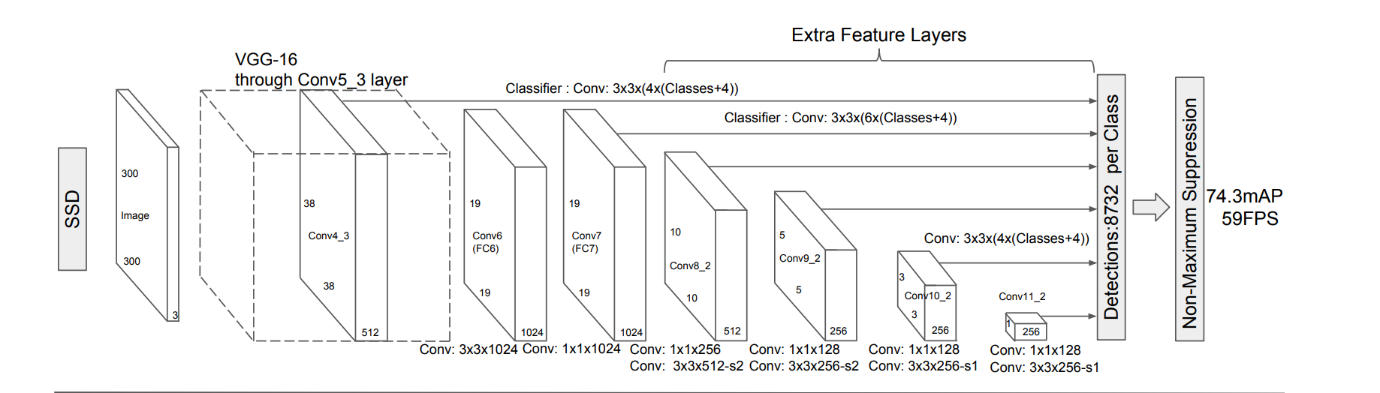


The confidence loss is the softmax loss over multiple classes confidences (c),

****

and the weight term α is set to one by cross validation.

**SSD Architecture**



### **6 Visualizations**

Include TensorBoard outputs and any other metrics output (mention this covers a stretch goal)

### **7 Stretch Goals**

### **7.1 Simpson character’s search in Image dataset (can we change title to indicate which stretch goal this implements?)**

**Reference** - Image Based Search Engine Using Deep Learning

**Overall Process**

Simpson Image Dataset

Feature Extraction

Save Features on disk

Feature Extraction

Input Image

Feature distance Comparison

Search Recommendation

**Feature Extraction**: CNN without dense/classification layer is used for feature extraction. Extracted features for all Image dataset are stored on disk. Feature extraction for 3000 images takes ~5 min on Argo.

**Distance Measure**: L1-norm or Manhattan distance is used for finding similarity between two images. The smaller the distance the more similar the images are.

**Steps:**

1. We extract feature for all images in dataset using CNN model and save them in pickle file.
2. Take input image from search input and generate feature map.
3. Compare generated feature map with all the features from pickle file.
4. Find distance using L1-norm for all images in dataset.
5. Sort all distance and recommend top images with minimum distance as matching images.

### **7.2 Object detection in videos**

This has been implemented using the YOLO algorithm. YOLO can parse videos into individual frames and apply image detection on each frame during playback. The speed of playback is highly dependent on processor type and capacity. In slower machines and non-GPU systems, video playback slows down as the computing resources apply image detection to individual frames.

Different YOLO models (trained with different hyperparameters) seemed to differ in ability to detect characters. For e.g. some models would correctly identify Homer Simpson as Homer, whereas others would mis-identify Homer as Crusty the Clown. Though the object detection in video exercise was fairly successful, the team identified various factors to test for improvement in accuracy in object detection.

Some of these include a detailed study of provided dataset to determine quality of annotations and quality of variances in character poses. Some of the other techniques could be to change anchor box presets in YOLO, change image size presets in YOLO and potentially explore running the original Darknet code (written in C)

### **7.3 Cloud Computing**

MLaaS – Machine Learning as a Service is offered by all major Cloud computing providers like AWS, Microsoft, Google and IBM. The team decided to focus on the AWS offering due to familiarity with AWS. AWS Sagemaker is a managed service offering that is advertised as an at scale Machine Learning/Deep Learning service with no infrastructure management, ability to deploy trained models behind endpoints.

At a high level, there are 4 ways to train/run models in Sagemaker. One is to use one of many AWS provided algorithms (for e.g. object-detection, K-means, image classification, among others), another to bring your own training logic and use AWS to train a model and host it, a third is to just host a model and the fourth is a fully custom container with custom libraries, code etc.

The team attempted to use the first option, which is to use an out of the box algorithm for object detection. AWS is internally using Single Shot Multi-Box Detection (SSD) algorithm to implement this. Some of the challenges faced were with Service limits on AWS account and data preparation. Existing service limits required a separate request for each EC2 instance type that was planned for use. This service level expansion exercise is time consuming as it requires an AWS representative’s involvement. The out of the box algorithm expects the data and related annotation to be a particular format (JSON). Simpons’s dataset available via Kaggle would need to go through a major transformation to align to the Sagemaker image detection format.

Alternatively, an existing algorithm that the team developed (for e.g. YOLO) could be ported over to Sagemaker and run as is. Team is exploring this option.

### **8 Acknowledgement**

Tad Berkery – Intern at GMU (add more content)

### **9 References**

* Shoji Kido; Yasusi Hirano; Noriaki Hashimoto Detection and classification of lung abnormalities by use of convolutional neural network (CNN) and regions with CNN features (R-CNN) <https://ieeexplore.ieee.org/document/8369798>
* Suresh Prasad Kannojia; Gaurav Jaiswal Ensemble of Hybrid CNN-ELM Model for Image Classification <https://ieeexplore.ieee.org/document/8474196>

**THE ABOVE SOURCE IS ABOUT HYBRID CNN-ELM – Should it still be included?**

* Faster R-CNN Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun <https://arxiv.org/abs/1506.01497>

|  |  |
| --- | --- |
| [1] | Kaggle.com, "The Simpsons Characters Data," kaggle, 15 June 2017. [Online]. Available: <https://www.kaggle.com/alexattia/the-simpsons-characters-dataset>. [Accessed 8 June 2019]. |

(<https://www.coursera.org/learn/convolutional-neural-networks?specialization=deep-learning>)

Sagemaker:

<https://www.youtube.com/watch?v=ym7NEYEx9x4>

<https://www.youtube.com/watch?v=R0vC31OXt-g>

Cloud

<https://www.kdnuggets.com/2018/01/mlaas-amazon-microsoft-azure-google-cloud-ai.html>