IAR: Intelligent Augmented Reality Framework

Jahoon Koo
Tobby Lie
Kevin Macfarlane
Devin Piner
Drake Young

April 2020

Abstract:

The field of Artificial Intelligence (AI) studies approaches to utilize computers to perform human tasks. On the other hand, Augmented Reality (AR) is a technology field where computer graphics is harnessed in a user's view of the real world. On their own, each individual computer science field mentioned above provides powerful aid and utility in improving everyday life. In this project, our ultimate goal is to seek out a framework (i.e., a general-purpose software that can be customized to implement different applications) that would yield the optimal marriage of these two aforementioned fields. Our framework, dubbed IAR (Intelligent Augmented Reality), consists of modules for data preprocessing, data management and organization, data modeling (with machine learning and deep learning models), and data visualization.

There are a plethora of applications that would see great benefit from the integration of AI and AR using IAR. For demonstration purposes, we have implemented three specific applications from diverse areas based on the IAR framework: 1) a tool that enables intelligent lane detection on road networks under adverse weather conditions and/or for visually impaired, 2) a tool that serves as therapeutic aid for patients with strabismus, and 3) a tool that allows for automated classification of garbage to educate the user on the differences between recyclable and non-recyclable trash.

Acknowledgements:

We would like to thank Assistant Professor Farnoush Banaei-Kashani and Professor Min Hyung Choi for their dedicated and insightful guidance throughout the duration of this project, and for providing us the opportunity to research under them for the duration of this project.

We would like to thank PhD Student Breawn Schoun for her continued support and assistance in development for virtual reality technologies throughout the duration of this project. We would also like to thank her for sharing her contacts and research in the field of strabismus therapy and providing us the opportunity to collaborate with her on her existing project in the field.

We would like to thank the Undergraduate Research Opportunity Program for providing us with financial assistance through their "Mini-grant". Their financial assistance allowed us to research to greater extents than we otherwise would have been able to.

We would like to thank the University of Colorado Denver: College of Engineering, Design and Computing for the opportunity to dedicate a year of research with access to the faculty, staff, students, and labs, as well as for providing us with the any additional resources we required throughout the research and development processes.

Table of Contents

Table of Figures

1 Introduction

Applications which utilize Artificial Intelligence enable computers to perform human tasks. Likewise, the field of Augmented Reality utilizes computer graphics to enhance a user's view or experience of the real world. Separately, both of these computer science fields provide powerful tools to all fields of study, as well as the day-to-day use by individuals outside academia. In this project, we intend to design a framework (i.e., a general-purpose software or workflow that can be customized to implement different applications) that would yield the optimal marriage of these two fields and pipeline the development of applications which marry them.

For demonstration purposes, we have implemented three specific applications from diverse areas to serve as proof-of-concepts for our IAR framework: 1) a tool that enables intelligent lane detection on road networks under adverse weather conditions and/or for visually impaired -- "Lane Detection", 2) a tool that serves as therapeutic aid for patients with strabismus -- "Strabismus Therapy", and 3) a tool that allows for automated classification of garbage to educate the user on the differences between recyclable and non-recyclable trash -- "TrashNet".

2 Background

2.1 Framework Background

Edge Computing: A method that distributes computing and data storage in a way that brings both closer to the location where it is required in an effort to reduce response times and bandwidth usage. In our case we desired a means to deliver our trained neural network models straight to our end devices for inference rather than having a stored model on the cloud which would increase our inference times greatly. By training our model with our entire dataset away from the edge and deploying a fully trained model to the edge we hoped to alleviate our inference times.

Neural Networks: Neural networks are algorithms that are inspired by the human brain designed to recognize patterns. Through pre-processing of data into a machine understandable format, various types of data such as text or image can be leveraged to cluster or classify (supervised vs. unsupervised learning). Neural networks are the focus in Deep Learning because they focus on defining network structures that are many layers deep. In order to train neural networks, there must be a set of classes that pose as ground truth. With these ground truth samples, the network is given example data that point to what these ground truth values are. They may either predict correctly or incorrectly based and based on these results they will slowly correct themselves over time through a cost optimization function. Through many training periods or epochs, the neural network gains a certain confidence over how it is able to predict and we decide a cut off point to stop training and use this trained model to use for predicting on never before seen data.

Augmented Reality: Augmented reality is a means to provide an interactive experience via a real world environment. Virtual objects through an AR device provide augmentation to what lies

in the real world. In doing this, objects that reside in the real world can be enhanced by computer-generated visual information. For an app to be considered as augmented reality it should deliver virtual information via a real world setting in a way that enhances user experience that would not be possible in the absence of AR.

2.2 Lane Detection Demo Background

For the purposes of this application, we defined Lane Detection as the use of computer software to analyze images or videos of road scenes in order to determine where the lanes are located relative to the vehicle or driver's position. For the purposes of this proof-of-concept application, lane detection is performed on images of road scenes from the point-of-view of the vehicle's driver.

The original intent when designing this application is to be used on an Augmented Windshield. We defined an Augmented Windshield to be hardware which is either embedded-in or mounted-on the windshield of a vehicle, which is to be used to augment the driver and/or passenger's experience while driving.

2.3 Strabismus Therapy Demo Background

The application designed for vision therapy has a focus on treating and dealing with strabismus. Strabismus is a condition where a person's eyes are not aligned on the same visual axis (i.e. they should be parallel to one another). This means that the eyes could be diverging or converging. This results in the brain relying more on one eye than the other. A common approach to treating this condition is to exercise the weaker eye.

This application uses pass-through virtual reality in order to implement various exercise techniques. A ZED Mini camera is used to capture the immediate environment from a first-person point of view and acts as the user's "eyes". This video stream is sent to an eye-tracking FOVE headset to display the real-world video feed to the user. The system was built in the Unity engine for manipulating the 3-dimensional objects shown to the user.

We use ArUco markers in order to track cubes in the real world. These will be tracked and augmented in the headset in order to perform various tests. After speaking with professional vision therapists, they revealed a variety of tests we could construct using the application. This is possible because of the complete control we have over the video feed and augmentation to each eye.

2.4 TrashNet Demo Background

The ultimate goal of TrashNet was to be a proof of concept of our intelligent AR (augmented reality) framework. It was important for us to be able to deliver an end to end workflow that contained both elements of AI (artificial intelligence)/AR. TrashNet was able to achieve this. TrashNet combines neural networks with AR through an iPhone as a device. The application was trained on classes of waste including: glass, plastic, paper, cardboard and general trash.

After training the model to a confidence we were happy with, we were able to then ship this model to our mobile app for inference. The application is able to take in frames from an iPhone camera and identify which class of waste the user is pointing the camera at. After this inference is performed, AR virtual information is displayed to the user to assist in understanding what they are looking at. We believe this is a powerful aid for users in educating them on the various types of waste and how to dispose of them.

3 Specification

3.1 Framework Specifications

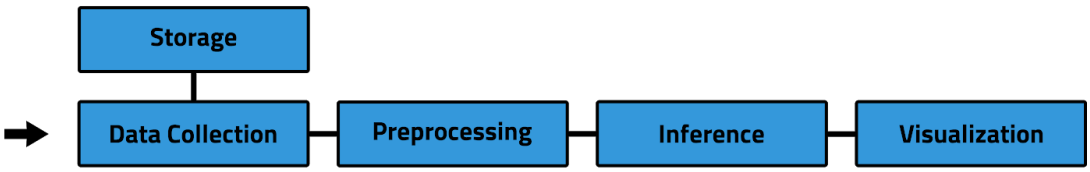

Figure 3.1.A. Framework Workflow Diagram

Figure 3.1.A above depicts the workflow we have designed for our framework. There are five nodes present at the general-level of this workflow that each application the workflow is applied to will have. There is some degree of variability within this workflow because the specific technologies and implementations can be fully customized to the need of the application, and in some instances, removed entirely.

The workflow begins with Data Collection. This node encapsulates the definition of technologies and methods used for taking in the source data. This includes both the training dataset for the Artificial Intelligence models, as well as input data which the model will be applied to. This node is mandatory for all applications, since both Artificial Intelligence and Augmented Reality require input data at some point during runtime.

The Data Collection node branches upward into the Storage node. This node encapsulates the location and technology used for storage of two major components for Intelligent Augmented Reality applications -- the data and the model. Some of the options we have considered are local and cloud storage. The general guideline our framework provides is that “smaller and local is faster, but bigger should be offloaded.”

The Data Collection node also passes forward into the Preprocessing node. This node encapsulates the technologies and methods used for transforming the input data and the data used for training models before the application and model respectively are able to operate. This step may sometimes be optional depending on the application, as sometimes, the raw input data is sufficient. If the raw input data is not, the application must preprocess it into a state which it can be operated upon. Such transformations and assisting technologies are specified in this node.

The Preprocessing node then flows forward, passing its completed resources into the Inference node. This node specifies the "brains" of the application. It encapsulates the Artificial Intelligence trained model information, as well as any supporting technologies needed for the model to operate. Within this node is where the technical aspects of the application's brain are used. As demonstrated by the Strabismus Therapy Demo application, this node can be used to define algorithms and other complex internal processes used in the runtime of the application -- any operations performed that affect output based on the input data.

Finally, the Inference node flows into the Visualization node. This node encapsulates the Augmented Reality aspects of the applications, including any underlying technologies utilized by the application. Within this node, the methodology for displaying the relevant information, as well as the hardware on which the information is displayed should be specified.

3.2 Lane Detection Demo Specifications

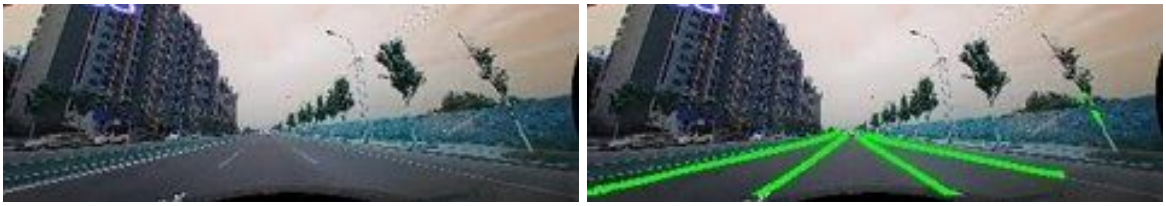

Figure 3.2.A. Before (left) and After (right) Lane Detection on a Cloudy Road Scene

The original intended use of the Lane Detection Demo Application was to augment the driver's visibility, by displaying current road lanes on an augmented windshield while driving. This was planned to be done by taking video feed from a mounted dashboard camera, and projecting the lane predictions onto a windshield, or by using the camera of a mounted smartphone, and displaying lane detections on top of the video feed.

However, we deemed that an application on such a large scale would be out-of-the-scope of a single proof-of-concept application for our framework. Instead, the application operates on a desktop/laptop computer. The application takes in image files of road scenes, and outputs an augmented version with the lane predictions drawn in green.

Figure 3.2.A. demonstrates the use of the Lane Detection Demo application. The input image is shown on the left. The application expects a 1640 pixel wide by 590 pixel tall image of a road scene (shown on the left), and it outputs a 328 pixel wide by 118 pixel tall image with the lanes drawn (shown on the right).

3.3 Strabismus Therapy Demo Specifications

The vision specialists asked us to recreate three fusional techniques. These techniques include displaying different augmentation information to each eye. This allows the vision therapists to determine a variety of different issues based on feedback they get from their patients.

It was also important for us to bridge the gap between the virtual and physical world. The vision specialists said that the current methods are either virtual (using a 3D TV), or physical (using specific objects and tools).

The application is built in such a way that the vision therapists can adjust the settings of the environment on a case by case basis. This allows them to adjust camera angles to be comfortable for the patient's eyes, it allows them to limit how much content is displayed to each eye, and is customizable enough for the creation of new tests.

The system works by finding the ArUco markers on our cube, then displaying information in augmented reality (in our case, another cube with some images on it) at the location of the physical cube. Figure 3.3.A shows how the user will see the cube while using the application.

Figure 3.3.A. The user's view of the augmented object

3.4 TrashNet Demo Specifications

Our current use case for this app is for those that are unfamiliar with the various types of waste. Our target audience is the general public in an effort to assist them educationally and develop positive habits when disposing of trash.

The app from the user's perspective is to be utilized in a simple and easy-to-use way. All they have to do is open the app, point it at trash they wish to identify and then let the app do the rest of the work. Names of the various classes will appear in AR space to help identify what the waste is as well as a confidence level the application is able to achieve and deliver to the user.

Figure 3.4.A demonstrates the use of this application as it is aimed at three classes of waste: metal, cardboard and trash. Our model is quickly able to pick up what the camera feed is displaying to it and with a certain confidence, convey to the user what class it predicts the waste to be.

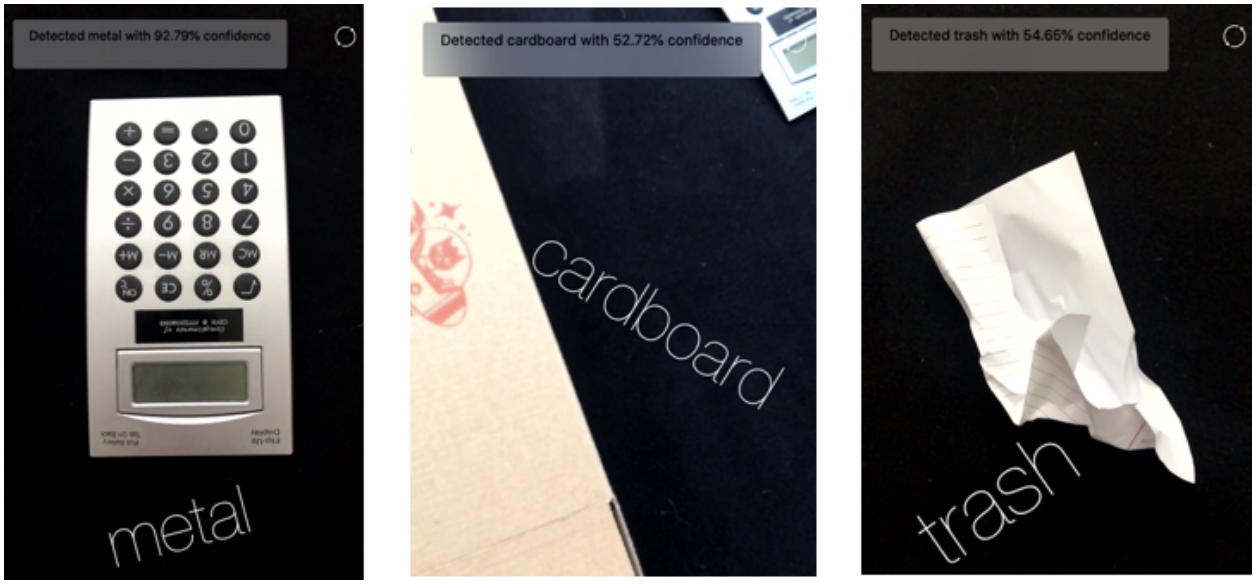

Figure 3.4.A. Demonstration of three of the waste classes being identified in the application

REMOVE THIS WHEN ALL SPECIFICATIONS DONE: High level description of what the system does from a client/user/business perspective. I think we should have a subsection for each of our three projects on the poster, as well as the framework as a whole. Alternately, we could JUST talk about the framework, and use a future section to show the separate projects as "implementations" of the framework. "If you are analyzing an algorithm or researching a hypothesis, the design and specifications sections may not be required, and instead become a problem statement, and descriptions of the suggested/tested approaches to solve it".

4 Design

4.1 Framework Design

Referencing Figure 3.1.A again, we also highlighted more detailed examples of technologies and methodologies to be used as reference when developing Intelligent Augmented Reality applications around our framework.

For the Data Collection node, we specify the details on both input data at runtime, as well as training data for the models. For input data, it must be specified the hardware which will be performing the input operation -- such as a cell phone camera, mounted dash cam in an automobile, or a Virtual Reality headset camera. For the training data, if the application uses an existing dataset, it must be properly mentioned within this node, as well as the structure of that dataset. If the application constructs its own dataset, it must still specify the structure, as well as how the dataset is collected -- the specifics vary depending on the type of data used, but we recommend specifying the features present (and their data types), the class labels used, and the number of data items.

For the Storage node, it must first be specified where the data and model will be stored -- whether the storage is local, on a cloud service, or elsewhere. It is recommended that when local storage is used, the hardware specifications are also provided. It is recommended that when using cloud storage, the service specifications are also provided. Once the data storage location is defined, the method for accessing the stored data must be defined -- such as wired BUS transmission, wireless API calls, etc. Finally, the structure and filetype/filetypes of the stored model and data must be defined; this includes filesystem structure of data directories and subdirectories, file names of relevant data informations, and any supplemental technologies used for I/O communications with the storage medium.

The Preprocessing node must specify all transformations performed on the data prior to use, as well as the data structure/data structures that encapsulate the data during use. This includes any libraries that may have been used to perform the transformations. An example of such a case would be the use of OpenCV to transform the shape of an image collected in the Data Collection node. In such a case, the use of OpenCV would be indicated, the shape of the input data prior to transformation, the shape of the input data after the transformation, and the data structure used for storing that image data.

The inference node must specify all information relevant to the model. Such information includes the hardware and software used for training the model, the structure of the model, the expected input of the model, the output of the model, and any libraries or other technologies which enable the use, definition, or training of the model. We recommend that this node be defined as verbosely as possible, ensuring that all operations relevant to the model are defined. Thoroughly.

Finally, the Visualization node defines all information relevant to the Augmented Reality portion of the application. This information includes all hardware, software, and external frameworks for augmented output, the hardware of the output device, and specifications on how the data received from the inference node are to be displayed.

4.2 Lane Detection Demo Design

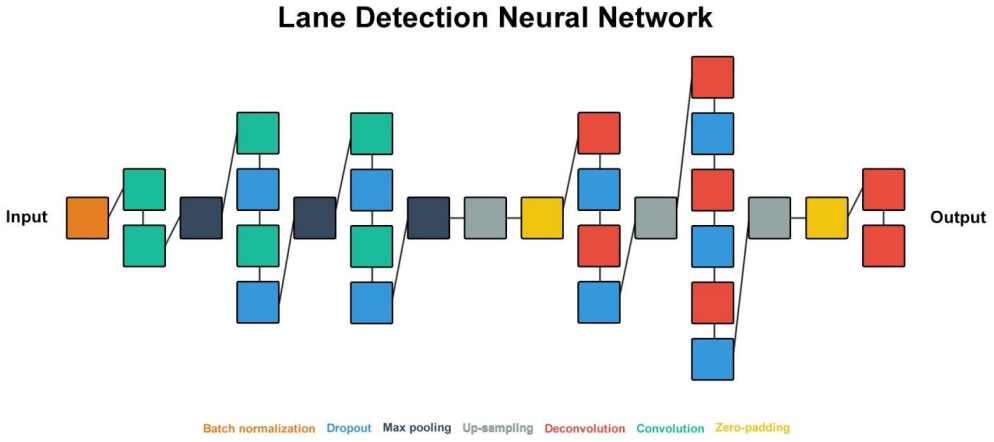

Figure 4.2.A. Model Structure for the Lane Detection Demo App

For the Lane Detection Demo application, we use a customized VGG16 model structure. The changes we make to the model structure allow us to work with images of the appropriate landscape orientation, rather than square images. We take in an input RGP image of 328p x 118p, pass it through a combination of Batch Normalization, Convolution, Dropout, Max-Pooling, Up-Sampling, Zero-Padding, and Deconvolution to generate an a 328x118 matrix of floating point values, where the greater the value, the greater the certainty that the pixel contains a lane. The appropriate ordering for the model structure is shown in Figure 4.2.A above.

In order to act as a proof-of-concept, this application was constructed following the modules described by our framework. For the Data Collection module, we use the existing dataset "CULane," which consists of 133,235 image frames of driving scenes in Beijing, China -- 88,880 training, 9,675 validation, and 34,680 judgement (Pan et al. 2018). Due to the size of the dataset, we opted to use a desktop computer's local file-system (1Terabyte Hard Disk Drive) for the Data Storage module of our application. Note that the full dataset needn't be stored once a trained model is constructed, but 35 Gigabytes of storage is necessary for the training process. The Preprocessing module of the application utilizes the Python version of OpenCV to extract the relevant image data from the image files given for either training or judgement. The Inference module of the application utilizes a Keras model with the VGG16 structure shown in figure 4.2.A. Finally, the Visualization module of this application utilizes Python's Pillow library for construction of the lane prediction images.

4.3 Strabismus Therapy Demo Design

This application is heavily focused on augmented reality. In order to effectively implement that, we used the Unity engine. It has a lot of useful features that make designing and building AR applications a streamlined process.

The main idea is to take in video feed from the ZED Mini cameras. In Unity, we use certain libraries (discussed in section 5) to read in the video feed, detect the location of a cube, augment the cube to perform certain tests, then display the result to the user. Figures 4.3.A and 4.3.B show the ZED Mini camera system taking in video before and after we apply augmentation.

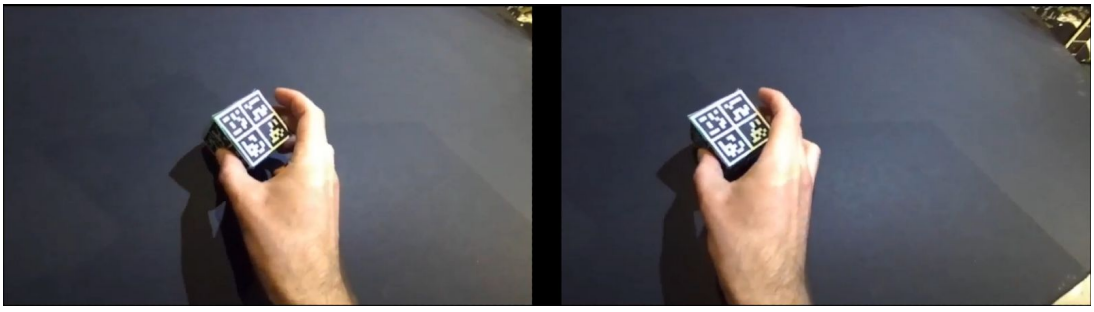

Figure 4.3.A. Video feed for each eye before augmentation

Figure 4.3.B. Video feed for each eye after augmentation

This system fits well in three of the IAR framework modules - those being data collection, preprocessing, and visualization. Data collection in terms of video feed, preprocessing in terms of altering the video to be the correct resolution to detect the ArUco markers, and visualization in terms of creating the augmented objects.

4.4 TrashNet Demo Design

The neural net we used for this application was Resnet-50 which is a convolutional neural network that consists of 50 layers. The input size of the images Resnet-50 takes is 224 x 224 so we utilized open-cv with Python in order to resize our input images appropriately. Resnet-50 focuses on something called Residual Learning which aims to resolve the issue of degrading accuracy. With very large networks, the vanishing gradient problem becomes an issue as the gradient is back-propagated to earlier layers and repeated multiplication may make the gradient infinitely small so as a very large network goes deeper its performance may degrade rapidly (He et al. 2015).

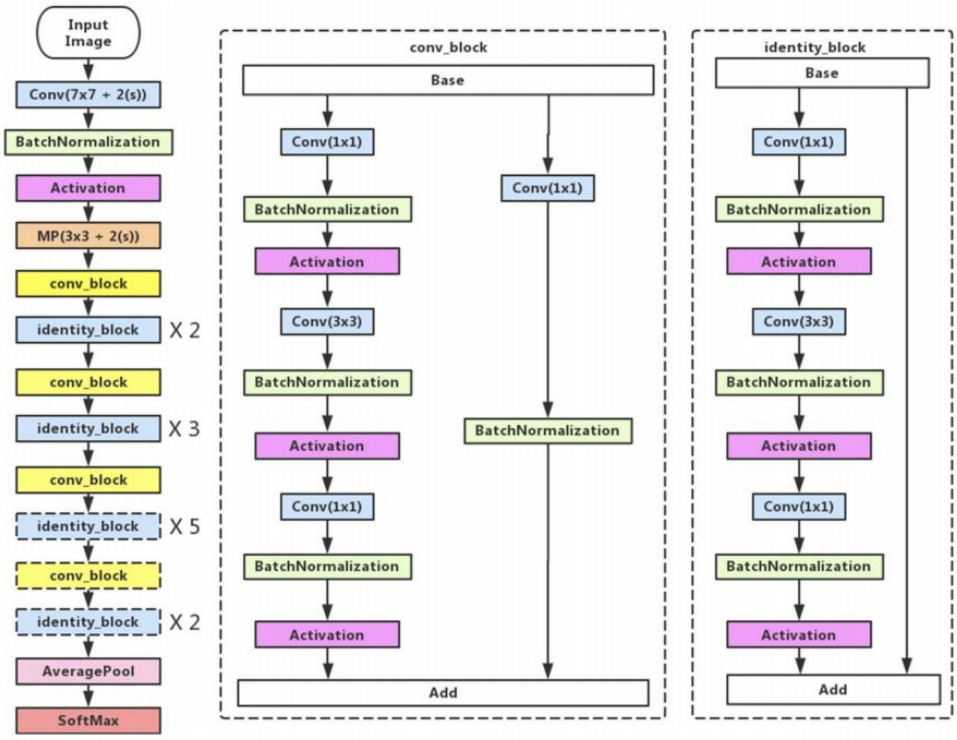

Figure 4.4.A Condensed structure of Resnet-50 (Ji et al. 2019)

The overview of TrashNet is through training via Resnet-50, we are able to develop a confident model when provided image data of waste classes. We plug this model into our mobile application on an iPhone and through CoreML and ARKit, two SDKs of iOS development, we are able to infer on the camera feed of the iPhone as well as deliver AR information.

To form an end-to-end workflow that fit within the parameters of our framework we had to align with the various modules of our framework. For Data Collection we utilized a Kaggle dataset that contained 6 classes of waste: cardboard (393), glass (491), metal (400), paper (584), plastic (472) and trash (127) (Cchangcs 2019). We stored this dataset in a Google Drive repository since we utilized Google Colaboratory for training. The Preprocessing was done via open-cv with Python in order to resize our images for our model and to convert our images to the appropriate array format for the model input. The Inference utilized a Keras model based on the Resnet-50 architecture above. For Visualization we utilized the iOS SDKs CoreML and ARKit. CoreML allows Keras models to be converted for iOS friendly models to be used in apps and ARKit was used to display AR information to the user's display.

5 Implementations

5.1 Framework Implementation

Our goal is to have each application fit into as many modules of the framework as possible. Of the three implementations, the TrashNet performed best as it incorporated the most modules. We

needed to keep these modules in mind as we implemented the various applications. This ensured that our progress would help us understand and find the aspects of the framework that worked best - and give us a sense of how to develop further applications.

The following subsections under the Section 5 Implementation header will define both the application's own implementations, as well as how those specific applications implemented themselves with respect to our framework and the defined nodes therein. Due to the modular, plug-and-play nature of the framework, some of the demonstration applications may not appear to include one or more of the nodes specified in our framework. This is not an issue, as our framework will treat undefined/unused nodes as empty, and pass the data from previous nodes along as-is, treating the missing nodes as an empty intermediary between its endpoints.

5.2 Lane Detection Demo Implementation

As mentioned in earlier sections, this application was developed, tested, and run using a Python 3.8 environment within an Anaconda 3 shell. The training environment used a Tensorflow-GPU environment to take advantage of an NVIDIA RTX 2070 GPU. We trained our Keras model on the CULane dataset for 100 epochs. The final training metrics will be discussed in greater detail in later sections. When training the dataset, we utilize a batch size of 32 -- trial and error deemed that any batch size larger than this would not fit in system memory (16 Gigabytes of Random Access Memory) for the training process. The program implementation was based and built upon a github repository, with modified use to implement our workflow (khaledmohammed 2019).

For this implementation, we wanted to test the robustness of our framework when using large datasets that require hardware more powerful than edge devices like Virtual Reality Headsets and Smartphones. Due to the plug-and-play nature of our framework's modules, we were able to scale-up the implementation of our framework on this resource-intensive application. As such, the greatest difficulty to overcome with the implementation of this application within the scope of our framework was the hardware and training-time limitations. The full training time took 115 hours to complete with the parameters we had. We would have liked to train using Google's cloud-based training, however, the size of the dataset prevented us from freely uploading to Google Drive, requiring us to train locally on the most powerful machines we could access. We addressed these hardware limitations by tuning the training parameters enough to fit on the most capable computer we had access to -- successfully pushing the resource demand of the application to a level suitable for testing the scalability of our framework.

5.3 Strabismus Therapy Demo Implementation

The decision to use Unity game engine made the augmentation aspect of this project easier. One of us had a lot of prior knowledge and experience using this engine, which also made the decision to utilize it easier. It has a lot of libraries that can be downloaded and used for applications including AR. Everything is coded in C#.

The version 2.8.5b ZED SDK was used, as well as version 2.8.2 of the ZED Unity Plugin. Version 0.17 FOVE SDK was used. A custom ArUco Unity Plugin (created by PhD student

Breawn Schoun) was used. These all allowed the system to communicate with Unity and to detect the specific objects. It also allowed us to focus more on the augmented reality aspects of this application.

One of the biggest setbacks was trying to implement this on the Microsoft Hololens V1. There were missing portions of libraries, as well as it only working with older versions of Unity. Trying to fix this and get it working (as well as testing with different IDE's and engines) wasted a lot of time and we should have just moved on to a different headset. Eventually we used a combination of devices - the ZED Mini camera and the eye-tracking FOVE headset.

5.4 TrashNet Demo Implementation

TrashNet was written in Python 3.7 within Google Colaboratory which is in the style of Jupyter Notebooks. Google Colaboratory enables the use of a free Tesla K80 GPU to be used with Keras. We mounted our Google Drive repository with our trash data and this allowed us to load our data into our Colaboratory environment for training. We trained our model for 100 epochs on our entire dataset. After training was complete we had a Python script that converted our Keras model to a CoreML model which could then be plugged into our iOS app. Through ARKit we were able to interact with the predictions made by our CoreML model and display corresponding information.

Some of the major frameworks/libraries we used consisted of Keras, open-cv, seaborn, CoreML and ARKit. Keras was utilized to train our neural network over our image data. Open-cv was used inorder to manipulate our image data in the appropriate ways for input into our model. Seaborn was utilized to display our results and metrics. CoreML and ARKit as mentioned above were used for developing our iOS application.

The main initial challenge was trying to train our model on just a Macbook Pro 2019. This imposed limited GPU resource difficulties and training times that took too long. After we switched over to our Google Collaboratory environment we were able to successfully cut down our training time significantly and test our model training more effectively.

6 Results and Evaluation

6.1 Framework Results & Evaluation

Reflecting on the performance of our framework with respect to the three demonstration applications, we discovered the strengths and weaknesses of our framework. Our three applications stressed three of the main test cases of our concern -- an application which utilizes all nodes of the framework to test standard use (TrashNet Demo), an application which pushes the hardware limits to test scalability (Lane Detection Demo), and an application where nodes were omitted to test robustness (Strabismus Therapy Demo).

In each of the three demonstration applications, our framework proves its ability to pipeline the workflow, showing its usability in the average case, its scalability for resource intensive applications, and its robustness for plug-and-play applications which may not utilize a subset of the nodes. We believe that the framework was proven to be a successful structure for design and workflow pipelining of Intelligent Augmented Reality applications.

In terms of weaknesses, however, we found that in our attempt to maintain abstraction, we fell into a pitfall of vagueness. Oftentimes, the specifics of what must be defined for each node were too value and subject to interpretation. This resulted in issues where the demo applications differed on the general points that needed to be defined in each framework. Though the framework was successful at the abstract level, at the lower, defined levels of implementation, we found the framework to need further definition for standardized use.

6.2 Lane Detection Demo Results & Evaluation

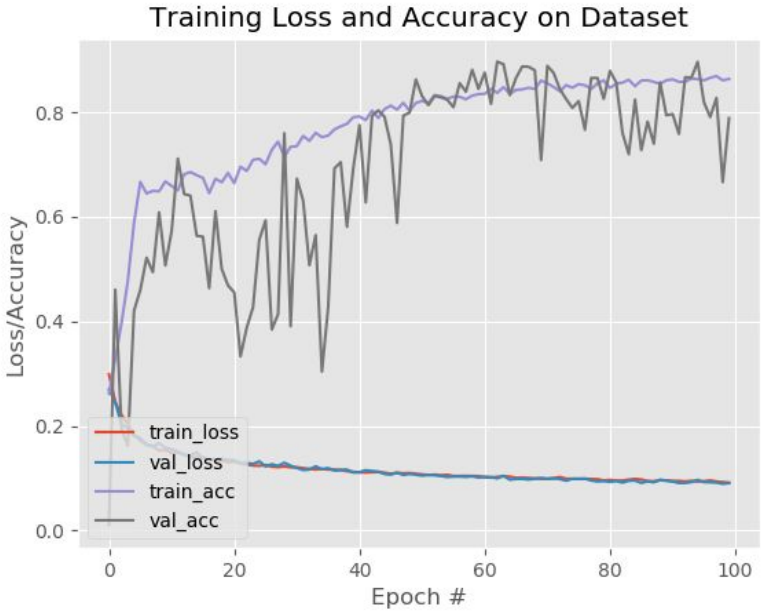

Figure 6.2.A. Training and Validation Metric for the Lane Detection Demo App

As mentioned in previous sections, the model used in this demo application was trained for 100 epochs. The final training metrics after those 100 epochs were: 0.9831 Accuracy, 0.2689 Dice-Loss, and an F1 Score of 0.6900. Likewise, the final validation metrics after the 100 epochs were: 0.9784 Accuracy, 0.2623 Dice-Loss, and an F1 Score of 0.5583.

These metrics are "good". As we see in Figure 6.2.A, throughout the 100 epochs, training accuracy was continuing an upward trend, while training loss was continuing a downward trend. This indicates that the model had not begun to overfit. We have decided to call this model a "well behaved model", as it trained without overfitting, and showed the desirable trends from the training process. However, the overall performance of the model has great room for

improvement. With such a large loss and low F1 scores, the model itself is not in a state where it would be viable for real-life use.

Figure 6.2.B. Demonstrations of Successful Lane Detection
(left: day with light traffic, center: day with no traffic, right: night with lights and no traffic)

Figure 6.2.C. Demonstrations of Lane Detection Weaknesses
(Left: high traffic, Center: night with low visibility, Right: irregular roads/intersections)

When running the application for demonstration purposes on the judge dataset, we found that there were noticeable distinctions between areas of success and areas of weakness. As shown in Figure 6.2.B, the application found its greatest success in scenes with little-to-no traffic, on straight, well-lit roads. The successes showed that nighttime lane detection was possible, as long as the detection was performed on highways with enough street-lighting. However, the weaknesses shown in Figure 6.2.C illustrate that the application struggles to determine lane locations in areas of high traffic, low visibility, and when the roads are irregularly shaped -- such as during turns and intersections.

Since the model hadn't overfit during the training process, an initial suggestion would be to re-train the model for a greater amount of time in order to improve such performance, or to prune the dataset to include a greater proportion of the scenes in which the model struggled.

Operation of the model itself aside, the application's intended purpose was to test the scalability of the framework we designed. Subjectively speaking, the framework used was robust enough to plug-and-play with the resource-intensive lane detection application, on more powerful hardware. Use of the framework's modules for this application streamlined the integration of large-scale Artificial Intelligence with Augmented Reality.

6.3 Strabismus Therapy Demo Results & Evaluation

To measure the success of this application, we wanted to see how well it fit into the framework, as well as to see if we could implement the three degrees of fusion (vision therapy diagnosis techniques). These techniques help the vision therapist evaluate the patient's eyes and determine what course of action needs to be taken.

Due to the travel restrictions during the COVID-19 crisis, we were unable to gather data and test these techniques at the vision therapy lab. We did, however, give a demo of the application to the vision therapist. He approved of the techniques and is ready to start testing.

This system utilized three of the modules from the IAR framework - with a focus on the visualization module. This application demonstrated a more augmented-reality approach to the framework and it succeeded in achieving that goal. This is because it has the best and most fleshed out implementation of augmented reality of the three applications.

6.4 TrashNet Demo Results & Evaluation

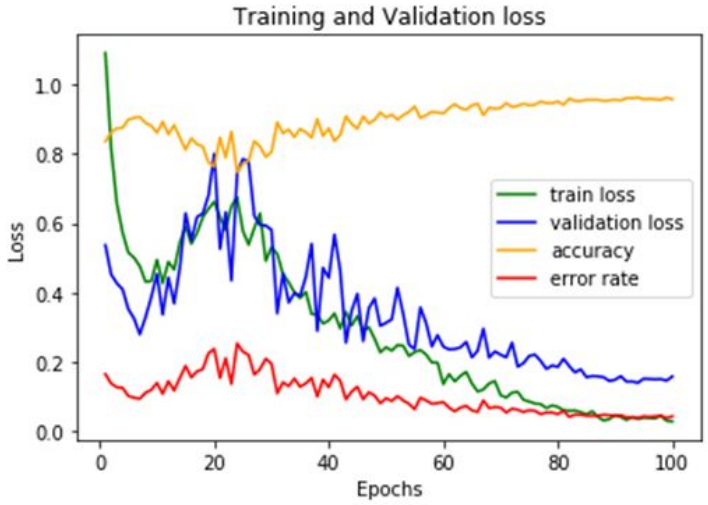

Figure 6.4.A. Training and Validation Metric for the TrashNet Demo App

As mentioned in the previous section, we trained our model for 100 epochs on our entire dataset. After 100 epochs, the model achieved 0.95873 accuracy on the training samples.

As we can see from figure 6.4.A, the overall accuracy of the model constantly increases, and training loss and validation loss both decrease concurrently. Validation loss occasionally goes lower than the training loss. It happens due to image augmentation on the training data and causes the model harder to predict in comparison to the unmodified validation data. At the end of 100 epochs, the validation error is low but slightly higher than the training loss. It indicates that the model is slightly overfitting, but the model is still good.

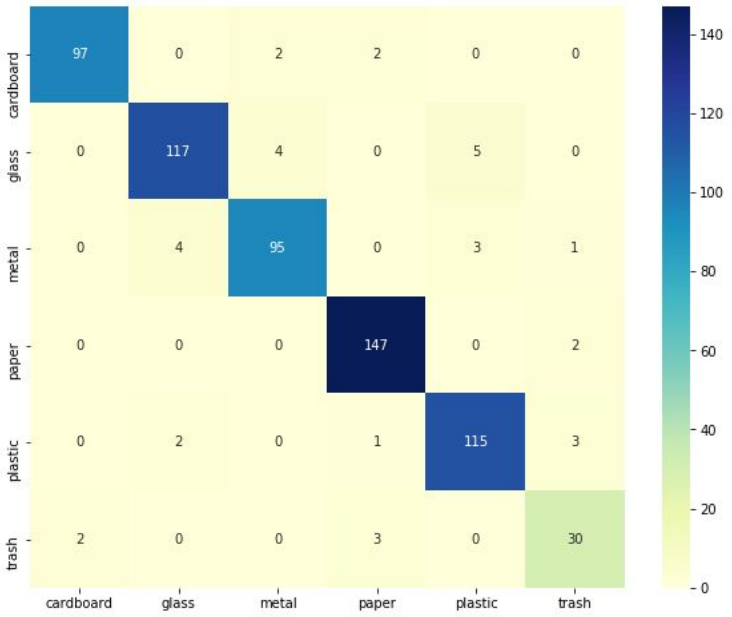

Figure 6.4.B. Confusion Matrix for the TrashNet Demo App

When running the trained model on the test dataset, we found some strengths and weaknesses in the trash classification model. Since each trash type is well distinct, the overall prediction rate of the model is great. The application can distinguish different types of trash and achieved around 93% accuracy on the testing dataset. However, as shown in Figure 6.4. B, the model seems to have confused metal for glass and plastic for glass because glass, plastic, and metal may look similar depending on angle, lighting, and shape.

Even though we had done data preprocessing to filter out bad data, the model is still confused metal for glass and plastic for glass. Also, the model is slightly overfitting due to image augmentation on the training dataset. Since we trained the model with a limited amount of data, training the model with a greater amount of trash dataset would be a good suggestion to avoid the model overfitting and make the model more robust.

This test application was successfully built using the IAR framework. It equally focused on augmented reality and artificial intelligence. Besides, this test application demonstrated the potential of the IAR framework that anyone can easily create an application using the IAR framework if they have proper dataset for the application they want to build and AR filters to display on devices.

7 Future Work

7.1 Framework Future Work

Part of our research led us to explore techniques in optimization and distribution in the context of mobile systems. Our investigation yielded technologies to be utilized at training-time and inference-time along with pre-inference data compression.

Training-time optimization exploits compression approaches that produce models with smaller memory and computation requirements that can then be used for inference directly on the embedded device. The hashing trick (Chen 2015), pruning (Han 2015), and optimal brain damage (LeCun 1989) reduce the number of connections required to describe the network, while the inception module (Szegedy 2015) in the GoogLeNet architecture makes use of the convolution algorithm to compress data passed to subsequent layers.

Inference-time distribution takes advantage of edge- and cloud-devices to lessen the workload of the end-device by sharing the computations relative to device capabilities. With minimal resources on end devices, it is in the interest of developers to carefully weigh the expected resource consumption levels of inference solely on the end device versus the resources needed to ship data to edge and cloud devices. When end device systems are critically limited, horizontal and vertical distribution designs can shift the computation towards the cloud. In the Early Exit configurations (Teerapittayanon 2017), the model is stretched from end- to edge-devices then to cloud-devices. At each device boundary, prediction confidence informs the system whether to return to the user or continue to the next device. Neurosurgeon (Kang 2017) dynamically cuts the model in two pieces, one side to be computed on the end device, and the other to be computed in the cloud. This split is decided by predictions of network speeds and available device resources to minimize inference time. DeepThings (Zhao 2018) designs a fused tile partitioning algorithm to optimally perform convolution functions in parallel by a collection of edge-devices coordinated by a work-stealing and data reuse-aware scheduling paradigm.

When operating on video data, input frames can be compressed before inference using dynamic Region of Interest encoding (Liu 2019). By identifying blocks within an image with high probabilities of containing objects of interest, dynamic compression can be applied to pixels based on their presence in these regions. Pixels outside regions of interest contain data unrelated to our target and can be compressed to lower levels of detail while pixels inside the regions are preserved at the highest level of detail. Region of interest encoding minimizes the transfer size of images over the network for inference in the cloud, reducing latency and consuming less power on end devices.

7.2 Lane Detection Demo Future Work

There are several areas we wanted to expand upon for this application if we had the available time resources to do so. As addressed in previous sections, the model struggled with specific types of road scenes, and the model had the potential to be trained further without overfitting. Because of these two observations, we would have liked to prune the dataset to include more training data in the “problem areas” and re-train the model for a greater number of epochs.

Due to the expensive computational resources required for training this model, we were unable to perform any level of hyper-parameter tuning on our model. If we had a powerful enough system, or enough funds to offload this computation onto a cloud service, we would have liked to do so in order to evaluate a more complete model for the purposes of testing our framework.

Finally, the current system of this application is designed for use in a desktop setting, and it runs inference on static images stored in the file system. There is limited practical use for such an application apart from being simply a proof-of-concept. Had we the time to expand upon this application, we would have liked to advance the application for mobile use on live video frames, rather than static system files.

7.3 Strabismus Therapy Demo Future Work

This application could potentially be used as a diagnosis tool. This would broaden the number of areas within vision therapy that it could be of use. In order to achieve this, a lot of eye data is required to train models so we can predict and identify specific conditions. The only way to gather this data is through running experiments - as the amount of data we need is not readily available. This would also allow it to fit into the IAR framework better - we could ensure it utilizes all of the modules.

The system for detecting the ArUco markers is a bit sluggish. This needs to be improved so that it is a smoother experience for the user. To enhance this aspect of the application, different detection and tracking methods must be tested and implemented. We will have to do some research, but perhaps a CNN, as well as some preprocessing, would help speed up the detection and tracking of the cube. This system should also be tested on a newer headset that will have improved and updated hardware.

Even though the system can be modified to perform a wide variety of tests, this all must be done within the Unity engine. This makes it difficult for vision therapists to make any modifications to these tests. An easy to use user-interface is a future goal for this project. That would allow someone with no experience in programming or using Unity to be able to adapt the application to their specific needs.

7.4 TrashNet Demo Future Work

Our two main goals for future work within TrashNet would be to further the educational aspect of the application as well as to gamify it in order to motivate users to utilize it more. If we were to improve these two areas of the app it would improve the usage which would in turn improve the overall education of the public.

We could gamify the application such that a points system is implemented to encourage users to correctly dispose of waste and penalize incorrect disposal. We could display more useful information to the user for each class of waste as well as create filters for certain classes that tie certain faces or 3D assets to each class of waste. This would in turn educate users on the various classes of waste in a painless manner. The points system could result in a game that could further reward the user for reaching certain milestones.

8 Conclusions

Computers utilize Artificial Intelligence to perform human tasks. Augmented Reality utilizes computer graphics to enhance a user's view or experience of the real-world. These separate computer science disciplines provide powerful tools to all types of users, however, a marriage of the two holds an even greater potential. In this project, we designed a framework to provide a generalized, scalable, and robust workflow for pipelining the development of applications which marry the fields of Artificial Intelligence and Augmented Reality -- Intelligent Augmented Reality.

We demonstrated the use of our framework by developing three unique proof-of-concept applications, each covering different fields of study with distinct use cases and needs from the framework we developed -- A tool for detecting and displaying lane-lines in road scenes of various conditions ("Lane Detection"), a tool for use in therapeutic aid of patients suffering from strabismus ("Strabismus Therapy"), and a tool for the classification of various trash and recyclable materials for appropriate disposal ("TrashNet").

Throughout the project we demonstrated the development processes of the applications individually, as well as how we utilized our framework to pipeline the development. We highlighted the strengths and weaknesses of each of our applications, and reflected on how each individually could be improved, and used these strengths and weaknesses to consider areas in which our framework could be further improved in the future, and perhaps become a set of standards to be used at the professional and research level after sufficient refinement.

References

Cchangcs, Garbage Classification, (May 2019), Kaggle Dataset, https://www.kaggle.com/asdasdasasdas/garbage-classification

Chen, Wenlin, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. "Compressing Neural Networks with the Hashing Trick," April 19, 2015. https://arxiv.org/abs/1504.04788.

Han , Song, Jeff Pool, John Tran and William J. Dally. "Learning both Weights and Connections for Efficient Neural Networks". arXiv. (October 2015). https://arxiv.org/abs/1506.02626v3

Kaiming He, Xiangyu Zhang, Shaoquing Ren, Jian Sun. "Deep Residual Learning for Image Recognition." *ArXiv.org*, 10 Dec. 2015, arxiv.org/abs/1512.03385.

Kang, Yiping, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. "Neurosurgeon." ACM SIGOPS Operating Systems Review 51, no. 2 (April 2017): 615–29. https://doi.org/10.1145/3093315.3037698.

khaledmohammed, *CULaneDetection*, (May 2019), GitHub repository, https://github.com/khaledmohammed/CULaneDetection

Le Cun, Yann S, Jason A Denker, and Sara undefined Solla. "Optimal Brain Damage," 1989. http://yann.lecun.com/exdb/publis/pdf/lecun-90b.pdf.

Liu, Luyang, Hongyu Li, and Marco Gruteser. "Edge Assisted Real-Time Object Detection for Mobile Augmented Reality." *The 25th Annual International Conference on Mobile Computing and Networking*, May 2019. https://doi.org/10.1145/3300061.3300116.

Pan, Shi, Lou, Wang, and Tang, *Spatial As Deep: Spatial CNN for Traffic Scene Understanding*, V1 (February 2018), distributed by AAAI Conference on Artificial Intelligence (AAAI). https://xingangpan.github.io/projects/CULane.html

Qingge Ji, Jie Huang, Wenjie He, Yankui Sun. "Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images". 26 Feb. 2019, www.researchgate.net/publication/331364877_Optimized_Deep_Convolutional_Neural_Networks_for_Identification_of_Macular_Diseases_from_Optical_Coherence_Tomography_Images.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions." *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. https://doi.org/10.1109/cvpr.2015.7298594.

Teerapittayanon, Surat, Bradley Mcdanel, and H.t. Kung. “Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices.” *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017. https://doi.org/10.1109/icdcs.2017.226.

Zhao, Zhuoran, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. “DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, no. 11 (2018): 2348–59. https://doi.org/10.1109/tcad.2018.2858384.