8-2017

Deep Multimodal Fusion Networks for Semantic Segmentation

Jesse Tetreault
Clemson University

Deep Multimodal Fusion Networks for Semantic Segmentation

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Jesse Tetreault
August 2017

Accepted by:
Dr. Melissa C. Smith, Committee Chair
Dr. Robert J. Schalkoff
Dr. Adam W. Hoover
Dr. Harlan B. Russell

Abstract

Deep Convolutional Neural Networks (CNN) have emerged as the dominant approach for solving many problems in computer vision and perception. Most state-of-the-art approaches to these problems are designed to operate using RGB color images as input. However, there are several settings that CNNs have been deployed in where more information about the state of the environment is available. Systems that require real-time perception such as self-driving cars or drones are typically equipped with sensors that can offer several complementary representations of the environment. CNNs designed to take advantage of features extracted from multiple sensor modalities have the potential to increase the perception capability of autonomous systems.

The work in this thesis extends the real-time CNN segmentation model ENet [39] to learn using a multimodal representation of the environment. Namely we investigate learning from disparity images generated by SGM [20] in conjunction with RGB color images from the Cityscapes dataset [10]. To do this we create a network architecture called MM-ENet composed of two symmetric feature extraction branches followed by a modality fusion network. We avoid using depth encoding strategies such as HHA [15] due to their computational cost and instead operate on raw disparity images. We also constrain the resolution of training and testing images to be relatively small, downsampled by a factor of four with respect to the original Cityscapes resolution. This is because a deployed version of this system must also run SGM in real-time, which could become a computational bottleneck if using higher resolution images. To design the best model for this task, we train several architectures that differ with respect to the operation used to combine features: channel-wise concatenation, element-wise multiplication, and element-wise addition.

We evaluate all models using Intersection-over-Union (IoU) as a primary performance metric and the instance-level IoU (iIoU) as a secondary metric. Compared to the baseline ENet model, we achieve comparable segmentation performance and are also able to take advantage of features

that cannot be extracted from RGB images alone. The results show that at this particular fusion location, elementwise multiplication is the best overall modality combination method. Through observing feature activations at different points in the network we show that depth information helps the network reason about object edges and boundaries that are not as salient in color space, particularly with respect to spatially small object classes such as persons. We also present results that suggest that even though each branch learned to extract unique features, these features can have complementary properties.

By extending the ENet model to learn from multimodal data we provide it with a richer representation of the environment. Because this extension simply duplicates layers in the encoder to create two symmetric feature extraction branches, the network also maintains real-time inference performance. Due to the network being trained on smaller resolution images to remain within the constraints of an embedded system, the overall performance is competitive but below state-of-the-art models reported on the Cityscapes leaderboard. When deploying this model in a high-performance system such as an autonomous vehicle that has the ability to generate disparity maps in real-time at a high resolution, MM-ENet can take advantage of unused data modalities to improve overall performance on semantic segmentation.

Table of Contents

List of Figures

Chapter 1

Introduction

The unreasonable effectiveness of deep learning has enabled rapid progress in many fields. The two research areas that have been impacted most by deep learning thus far are natural language processing and computer vision. The success of deep learning in these problem areas compared to their traditional counterparts is due to the robustness and descriptive power that a CNN can achieve through a learned representation that is iteratively constructed by building on top of increasingly abstract features. These characteristics are what sets deep learning algorithms apart from traditional approaches and make them useful for such a wide range of problems. Deep neural networks use hierarchical feature abstraction to complete tasks that seemed out of reach with traditional machine learning algorithms.

Most contributions to the field of deep learning fall in the category of either exploring novel learning methods or applying advances in the field to practical applications. Self-driving cars and other autonomous systems such as drones have become some of the most relevant application areas of this research, especially in computer vision. These systems aim to use bleeding-edge technology but also must adhere to strict constraints concerning the safe operation of the vehicle, as well as the safety of actors in the surrounding environment. Real-time execution of any computer vision algorithm that is deployed on these systems is one of the highest priorities.

It is also important to note the overall objective of software controlling an autonomous vehicle. The objective is not to simply perform image processing, but to reason about the environment and make decisions. This objective takes the combination of an effective vehicle sensor hardware configuration, perception software that can extract, combine, and process information from the sen-

sors, a strategy for controlling the vehicle or path-planning once the environment is perceived, and an actuation system to send signals that control the physical operation of the host vehicle. This complexity is why we are motivated to infuse deep neural networks inside the perception loop with multimodal representations of the environment; while we only focus semantic segmentation in this research, exploring effective ways to learn richer representations of the environment, like one that has structural information, could lead to improved performance on a wide range of perception tasks.

We choose to fuse color and depth information to perform semantic segmentation as opposed to detection or classification. Working at a pixel-level granularity, the boundaries and delineation between objects is very important. When fitting a bounding box around objects of interest in detection, the edges of the bounding box must fit as closely as possible to the furthest points of the object in the horizontal and vertical direction, but tracing the silhouette of the entire object is not necessary. But this is precisely what is required for segmentation and also is why depth information is useful for this task. Disparity images representing the depth of the scene are just that: silhouettes of objects with no information about color or texture, but strong information about structure and geometry. When added to the color and texture features that can be extracted from RGB images, this particular bimodal representation has a large amount of useful information for several different tasks.

This research makes contributions to the field of deep learning in the area of practical segmentation implementations for embedded systems. First, we develop and present a method for extending the ENet architecture to learn from RGB and depth data concurrently. We next investigate what level of feature abstraction is appropriate for fusing depth and color image features as well as the optimal fusion strategy. Further, we directly compare features learned from each individual modality prior to fusion in order to confirm the efficacy of designing individual feature extraction branches.

The remainder of this thesis is organized as follows. Chapter 2 provides background information related to the fields of computer vision and deep learning. Chapter 3 follows with a discussion of recent work in deep CNN architectures for semantic segmentation, multimodal deep learning, and learning from depth information. These two chapters provide the motivation and background for designing a real-time multimodal CNN.

Chapter 4 presents the network architecture and necessary modifications to learn from RGB and depth. Here we discuss extending an existing CNN model to have two input streams and show

methods for joining features from separate modalities. Chapter 5 details the experimental setup including training hyperparameters, training hardware, the Cityscapes dataset, and deploying the network for inference.

Chapter 6 presents the results of the final multimodal CNN model on Cityscapes. Class-level and category-level IoU are presented and compared to the baseline ENet model. We extract the feature activation maps from several versions of the model to show and compare the unique but complementary RGB and depth features learned by the network. Conclusions and suggestions for future work direction in this area are given in Chapter 7.

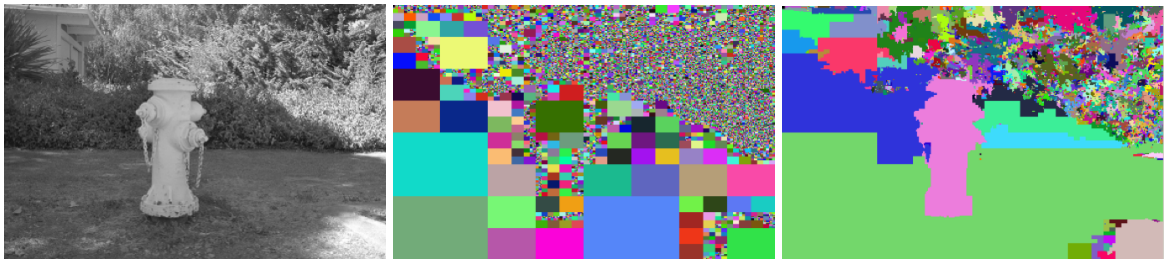
Chapter 2

Background

This chapter provides background information related to this research. We begin with an introduction to different types of computer vision problems and focus on traditional approaches to semantic segmentation. Of importance is the distinction between classification, detection, segmentation, and instance-level segmentation. We follow with the taxonomy and nomenclature used in the field of artificial intelligence in order to distinguish between subfields of research. The final area of background covers artificial neural networks (ANN) and deep learning algorithms, and explains why deep learning algorithms are so powerful for solving computer vision problems.

2.1 Computer Vision

The objective of a computer vision algorithm is to give meaning to information extracted from an image. There are several "semantic granularities" that can be inferred from different computer vision algorithms. The coarsest level of semantic granularity, classification or recognition, provides no localization information. Classification algorithms assign one label or class per input image that either describes the scene or the most salient objects within it. When we are interested in what objects are present in the image as well as their location, the problem becomes object detection. Object detection algorithms regress a 2D bounding box and associated class label to each relevant object in an image. Moving beyond detection to a more powerful level of descriptiveness is the task of semantic segmentation, or assigning a class label to each individual pixel in an image. A more formal definition of semantic segmentation is provided in Section 2.1.1. The finest level of



(a) Original image (b) Regions after splitting step (c) Final regions after merging

Figure 2.1: Split-and-merge algorithm [23]

granularity is instance-level segmentation, where not only is every pixel assigned a class label, but is also assigned an instance ID. For example, if two cars or pedestrians are overlapping in the image plane, we would like to be able to tell them apart by assigning each of them a unique instance ID.

2.1.1 Semantic Segmentation

The goal of a segmentation algorithm is to divide an image into groups of pixels that are determined based on some local property which may include gray level, texture, color, gradient value, shape, or structure from motion. Semantic segmentation algorithms group pixels in the same way but also determine a label for - or assign semantics to - each of these regions. Here we present traditional approaches to segmentation prior to the advent of deep learning.

An intuitive way to approach the segmentation problem is to pose it as partitioning pixels into sets. Given image I we would like to find a collection of sets $S_1, ..., S_N$ such that $I = S_1 \cup S_2 ... \cup S_N$ (sets cover the entire image) and $S_i \cap S_j = 0$ for all $i \neq j$ (sets do not overlap each other). A predicate $H(S_i)$ is used to measure the homogeneity of each region and is designed to capture one of the local properties mentioned above. This method can be implemented in a top-down fashion by treating the entire image as a single region and iteratively splitting it into subregions, or in a bottom-up fashion by treating each pixel as its own region and iteratively merging them together. In practice, merging typically works much better than splitting. The split-and-merge algorithm [23] takes advantage of both approaches by first splitting an image into subregions and then merging them together for a final output. An example of this is shown in Figure 2.1.

Another way to approach segmentation is to treat it as a region growing problem. To create regions, a seed pixel is chosen (possibly at random) and its neighbors are considered for merging based on a cluster or region model. A simple example of a region model is one that keeps track of

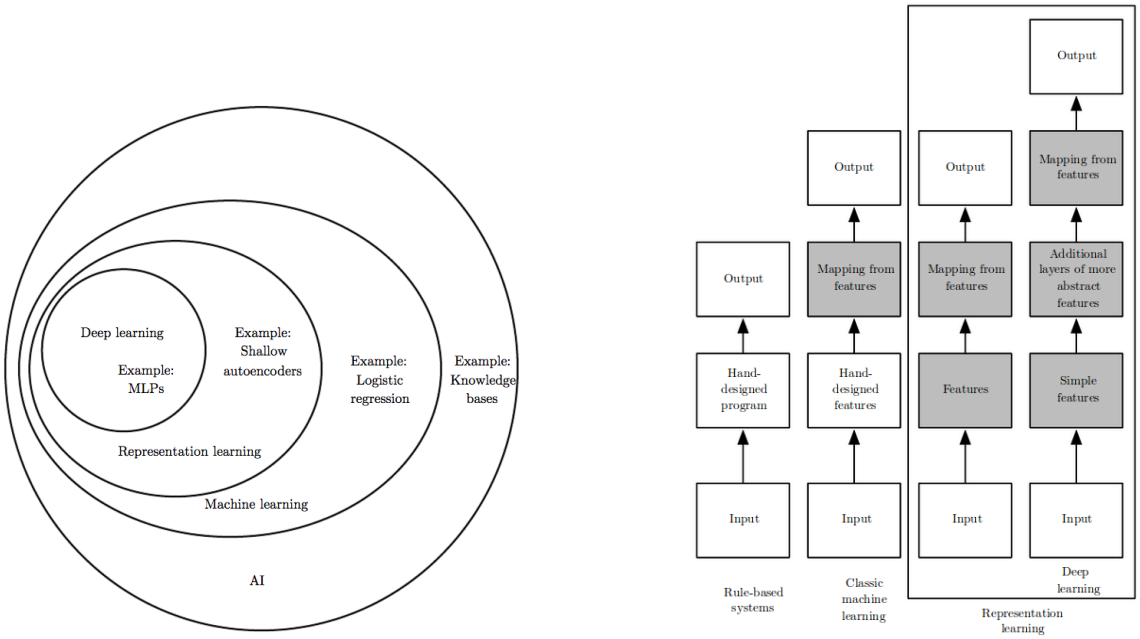
the mean and covariance of a region as it grows, only adding neighboring pixels if their addition keeps these values under a chosen threshold. Region growing can be thought of as a generalization of the flood-fill tool in Microsoft Paint. Some potential drawbacks of using the naive region growing algorithm is the choice of seed pixel may not be optimal and since regions are grown sequentially, early regions may dominate.

The Watershed algorithm, first introduced by Beucher [5] and made computationally feasible by Vincent and Soille [47], addresses some of these issues. The Watershed algorithm computes the gradient magnitude image and interprets it as a topological surface. Interpreting the gradient as a surface maps regions of the image with strong local similarity into valleys and regions with weak local similarity to peaks. Watershed proceeds by flooding the valleys or basins at their lowest point and gradually increasing the grayscale level (or water level) until every pixel belongs to one of these basins. The resulting regions associated with each basin are segmented objects. To reduce oversegmentation, it is also possible to use the chamfer distance image to determine markers that restrict the points where new basins may be started.

2.2 Deep Learning and Artificial Intelligence

Artificial intelligence (AI), machine learning, and deep learning are often used interchangeably but each imply different classes of models and algorithms. Here we chart the space of AI algorithms and differentiate each of these areas of research from each other.

Figure 2.2 provides a Venn diagram and flowchart of the sub-fields of artificial intelligence. A system is said to have artificial intelligence if it exhibits intelligent behavior with respect to tasks that would be considered intelligent for humans. However, this does not imply that the behavior must be learned. Some examples of artificial intelligence that are not required to use learning are knowledge bases or expert systems. These systems have a database of facts about the state of the world or environment and rules for manipulating these facts. When given a goal, a set of facts about the environment, and a set of rules, an expert system can exhibit intelligent behavior. While this functionality is considered intelligent, the system does not learn from the data; it only manipulates the data according to the set of rules. These systems can be combined with machine learning techniques so that the rules it uses to manipulate the database are learned, but this is not a requirement.

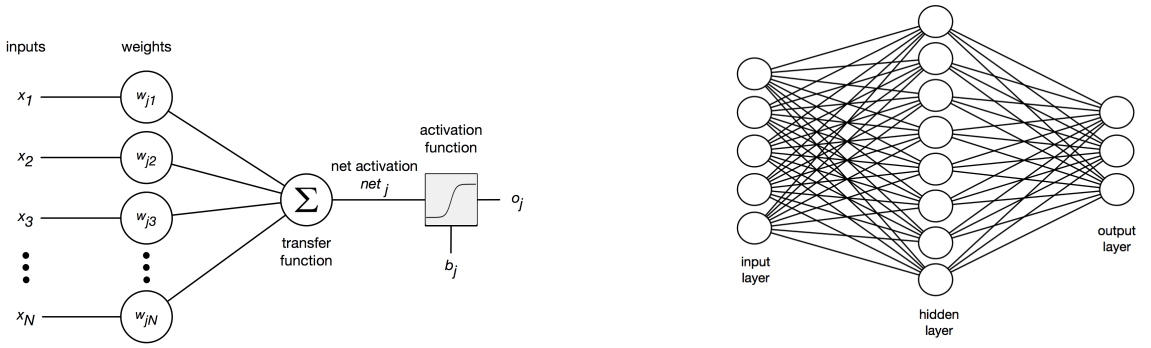


(a) Venn diagram of AI

(b) Flowchart of AI system components

Figure 2.2: Taxonomy and nomenclature of Artificial Intelligence [14]

The first sub-field of artificial intelligence is machine learning. According to Mitchell [36], "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." A less formal definition of machine learning is, "a computer program that improves in performance when provided with better data." While we do not expound on this point here, it should be noted that better data is not equivalent to simply providing more data. Machine learning can replace algorithms and techniques that previously had to be designed by engineers because of their complexity. They can also help with tasks that are difficult to formalize mathematically by learning relationships and representations that are otherwise impossible to visualize in high-dimensional space. Regression is the prototypical example of machine learning algorithms. Regression is a machine learning task where the goal is to predict a value when presented with some vector of inputs. Linear regression is the simplest example where the output scalar y is predicted using $\hat{y} = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w}$ is a learned set of weights and $\mathbf{x}$ is an input feature vector from the training set. The normal equations can be used to solve for a weight vector $\mathbf{w}$ in closed form when choosing to minimize the mean squared error as the cost function.



(a) Computational model of a neuron

(b) Fully-connected shallow architecture

Figure 2.3: Modeling an artificial neural network

The next sub-field of artificial intelligence is representation learning. Representation learning extends classic machine learning in that it still learns a mapping from a set of features to an output, but these features can be learned instead of having to be hand-designed. A simple example is a shallow autoencoder network. When presented with an input example, an autoencoder learns another representation of the example, usually compressed or with reduced dimensionality. This new representation could be used by another machine learning algorithm to perform a second task like regression or classification. The motivation behind representation learning is that learned representations can yield higher performance than hand-designed feature vectors. Many unsupervised learning algorithms fall in this category as their goal is to learn useful representations when there is a lack of labeled data.

The most contemporary sub-field of artificial intelligence is deep learning. We further define deep learning and the artificial neural networks that make them possible in the next section.

2.2.1 Artificial Neural Networks

Figure 2.3a shows a generic model of a computational neuron. Most models of a neuron can be functionally abstracted into two stages of computation: an affine transformation with learned parameters followed by a fixed nonlinear activation function. Given input vector x and weights w_i and temporarily ignoring the activation function, the output of a linear neuron i is:

$$\boldsymbol{o_i} = f(\boldsymbol{x}; \boldsymbol{w}, b) = \sum_j w_{ij} x_j + b = \boldsymbol{w}^T \boldsymbol{x} + b \tag{2.1}$$

Equation 2.1 can be conveniently represented as an inner product between vectors $\boldsymbol{w}$ and $\boldsymbol{x}$ added to the bias term b. Bias b is an added parameter that gives the neuron more flexibility; the bias can be used to selectively inhibit certain neurons or cause $f(0) \neq 0$ for that unit. The set of all $\boldsymbol{w}$ and b for a given network are typically represented together in the parameter vector $\boldsymbol{\theta}$. We can extend this model to give it nonlinear dynamics by sending this value through a differentiable activation function, such as the hyperbolic tangent:

$$\boldsymbol{o_i} = f(\boldsymbol{x}; \boldsymbol{\theta}) = \tanh(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{2.2}$$

The optimal weights for a linear neuron can be estimated in closed form using the pseudoinverse method, as in linear regression. Another approach that can be used to train both linear and nonlinear neurons is to use an iterative gradient descent procedure. Gradient descent approaches take the derivative of a cost (criterion, loss, objective) function used to quantify model error with respect to the network parameters and follow it to a local minimum. At each iteration we step in the direction of the negative gradient of the cost function until some stopping criteria is met. If we select the sum of squared error (SSE) between target vector $\boldsymbol{t}^p$ and output vector $\boldsymbol{o}^p$ given pattern p as a cost function for units with a sigmoid activation function, we can use the delta rule to update the weights at each time step. This method is shown below in Equations 2.3 - 2.5 for a 1-layer network where net_j is used to denote the pre-synaptic activation $net_j = \boldsymbol{w}^T \boldsymbol{x}$ for unit j and ϵ is the learning rate. We use notation w_{ji} to represent the weight of the connection between neuron j and the i^{th} output from the previous layer, which in a 1-layer network is simply the input vector x.

$$o_j = f(net_j) = \frac{1}{1 + e^{net_j}} \tag{2.3}$$

$$f'(net_j) = o_j(1 - o_j) \tag{2.4}$$

$$\Delta w_{ji} = \epsilon f'_j(net^p_j)(t^p_j - o^p_j)x_i \tag{2.5}$$

Constructing layers consisting of multiple neurons and arranging them in a sequential fashion creates a feedforward ANN, or Multi-layer Perceptron (MLP). Feedforward networks are used to approximate a function f^*. More formally, a feedforward network defines a mapping $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$. An example of a simple ANN with one hidden layer and one output layer is shown in Figure 2.3b.

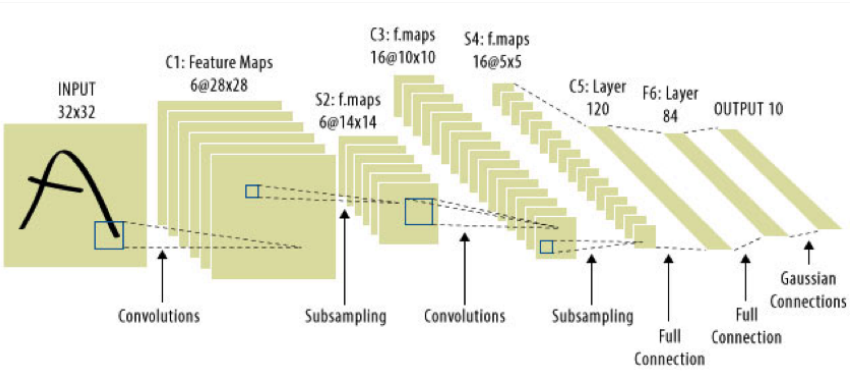


Figure 2.4: LeNet-5 architecture [34]

This 2-layer network models and learns the function $f^{(2)}(f^{(1)}(\boldsymbol{x};\boldsymbol{\theta}))$. Note that the layers in this model are fully-connected, i.e. the output of every neuron in the hidden layer has a weighted connection to every neuron in the output layer.

Due to the dependency each neuron in a MLP has on the activations from the previous layer, the delta rule must be generalized such that it may be used to update all parameters of a network beginning with the output layer. Now known as the backpropogation algorithm, in 1986 Rumelhart et al. [40] showed it was possible to recursively back-propogate errors through a neural network using the chain rule, which leads to the use of δ to represent the error signal unique to each neuron. For the units of the 1-layer network mentioned previously and for the output units of a MLP, $\delta = f_i'(net_i^p)(t_i^p - o_i^p)$. For hidden layers, we compute δ_j by taking the inner product of the succeeding layer k errors δ_k and the weights between the hidden layer and succeeding layer w_{kj}. In the 1-layer network example each weight was also updated using the dimension of the feature vector $\boldsymbol{x}$ it received as input. Deeper layers receive a previous layer's output as opposed to the input feature vector and so we instead represent this as h_i where $h_i = x_i$ for the input layer and is the output of the previous hidden layer for all other layers. This leads to a generalized form of the delta rule that can be used to back-propogate errors needed to adjust the weights of each individual neuron at each training step.

$$\Delta w_{ji} = \epsilon \delta_j^p h_i \quad \text{where} \quad \delta_j^p = \begin{cases} f_j'(net_j^p)(t_j^p - o_j^p) & \text{if output layer} \\ f_j'(net_j^p) \sum_k \delta_k^p w_{kj} & \text{otherwise} \end{cases}$$

The success of the backpropogation algorithm made it feasible to begin training MLPs

for interesting tasks, but due to the limitations of computational resources at the time, training MLPs was very time-consuming. Further, the dynamics of an MLP with fully-connected layers were difficult to interpret. Motivated by these two factors, Bengio and LeCun worked together in the 90's to produce Convolutional Neural Networks (CNN). Over the course of four years they published a series of papers that eventually culminated in the network architecture known as LeNet [33, 6, 32, 34], shown in Figure 2.4. The success of this network was significant because it proved using the backpropagation algorithm could teach an ANN how to extract features with convolutional layers as well as classify them with fully connected layers. Derivatives of the networks produced by Bengio and LeCun were widely adopted by the US Postal System to recognize letters and digits on envelopes. The well-known MNIST database [34] of handwritten digits was an outcome of this research and still serves as the "hello world" dataset for many computer vision and machine learning tasks.

The key feature of LeNet is that it makes use of convolutional layers followed by subsampling layers, each of which are much more sparsely connected than a fully-connected layer. Instead of directly learning a function mapping from the previous layer's output, convolutional layers can be thought of as "filter banks." Each filter is a convolution kernal with learned parameters that is then applied to every valid location in the input. The output of convolutional layers are referred to as feature maps, as they encode the presence of a particular feature at every spatial location. In practice, this is implemented as several units whose weights are constrained to be identical but each have weighted connections to a different location of the input feature map or image. Subsampling or "pooling" after each convolutional layer gives the model shift, scale, and distortion invariance while also reducing the dimensionality of feature maps.

Convolutional layers have unique parameters that must be determined to make them useful. First is the receptive field, or size. Like many traditional convolution kernels, this is typically a 3x3, 5x5, or 7x7 matrix. It is also possible to use 1x1 convolutional layers for feature expansion or reduction. The receptive field controls how much spatial information each neuron sees. A second parameter is the stride S. We can adjust the stride - or how many pixels should be between each application of the convolution - to control whether or not the receptive fields overlap each other. The final parameter P helps control the size of the output feature map. Again, as in traditional convolution, the pixels on the edge of the image cause issues as they do not have neighbors to convolve with. This can be addressed by padding the input with zeros around the border pixels

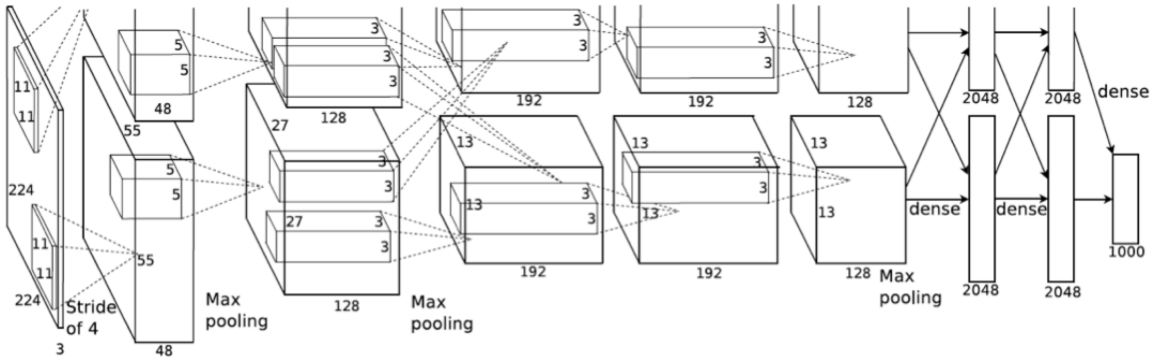


Figure 2.5: AlexNet architecture [30]

with P if desired so that there are valid values for a full convolution operation at each location. If a convolutional layer is not padded, it will slightly reduce the size of the output feature map. After these parameters have been determined, the output of convolutional neuron j with activation function σ and a receptive field of 5x5 at input location x, y can be written as follows:

$$o_j = \sigma(\sum_{l=-2}^{2} \sum_{m=-2}^{2} w_{l,m} x_{x+l,y+m} \ + b) \tag{2.6}$$

2.2.2 Deep Learning

The success of LeNet built a solid foundation for research in the subsequent decade, but most progress was incremental and still somewhat hindered by computing resources available at the time. The ANN architecture now known as AlexNet [30] gave birth to the field now known as deep learning by outperforming several state-of-the-art classification algorithms by a wide margin to win the ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC12) [11]. AlexNet draws inspiration from LeNet, and is composed of 5 carefully designed convolutional layers followed by 3 fully-connected layers whose output is fed to a 1000-way softmax function for classification. Deep learning algorithms have dominated ILSVRC and almost all other image recognition challenges since the initial success of AlexNet.

A "deep" neural network (DNN), in the technical sense, is any ANN architecture that has more than one stage of non-linear feature transformation. With this definition, a simple 2-layer ANN could be considered a DNN. This characterization, however, does not capture the full meaning behind deep learning. The power of deep learning lies in its ability perform hierarchical feature abstraction. Unlike representation learning, where it can be difficult to extract high-level, abstract

useful features from the data, deep learning "learns" representations that can be expressed in terms of other, simpler representations [14]. Just as an MLP or feedforward ANN can be thought of a series of mathematical function mappings from input to output values, DNNs can be thought of as a series of feature extraction stages that are typically then followed by a function mapping to the final output.

While the choice of optimization algorithm, cost function, dataset, regularization techniques, hyperparameters, and model design are crucial to properly implement a deep learning algorithm, in a simple sense, architecturally, this can be accomplished by stacking more of the right type of layers on a network. While this seems like an obvious research direction to follow, it should be noted that the advent of deep learning was in tandem with the rise of GPGPU (general-purpose graphics processing units) computing. GPGPUs are processors that can perform many computations in parallel, which lends itself to accelerating ANN training. Looking closely at the backpropogation algorithm, we can observe that while all units in a particular layer depend on the outputs from the previous layer, they do not depend on values from any adjacent neurons within the same layer. This type of problem is referred to as "embarassingly parallel" meaning that it is possible, with enough compute power, to calculate the activations for an entire layer simultaneously. This feature is useful both in the forward pass and backpropogation update stage. Much of the success of deep learning has arisen simply because we now have the capability to explore these types of architectures and problems.

Chapter 3

Related Work

In this chapter we map the space of solutions using deep learning for semantic segmentation. We first focus on well-studied DNN architectures that have been used for this task. As MM-ENet is designed to run in an embedded system, models are evaluated and compared particularly with respect to inference speed and size in memory. This discussion is followed by an overview of multimodal neural network architectures and approaches to learning from more than one form of input data. Finally we present solutions that learn specifically from color and depth images to perform object detection and semantic segmentation.

3.1 Semantic Segmentation

State-of-the-art CNNs for semantic segmentation are typically composed of two sub-networks: an encoder and a decoder. The objective of the encoder network is to extract features from an image while also reducing the dimensionality of the representation. This intermediate representation is used as input to a decoder that upsamples these features back to the original image resolution. A softmax function is typically applied to each pixel of the decoder output to produce a categorical distribution over all object classes. Therefore, the output of the decoder network is typically a CxWxH tensor with each channel encoding the probability if its corresponding object class at a particular pixel location.

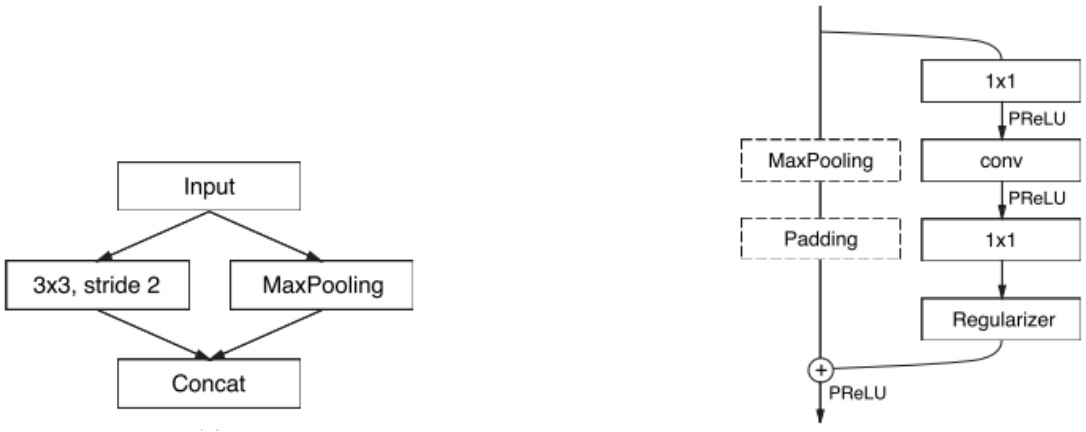
The Fully Convolutional Network (FCN) [35] was one of the first successful approaches using a contemporary CNN architecture to perform dense segmentation. The authors create an fully-

convolutional encoder by "convolutionalizing" the fully-connected layers of pre-trained classification networks such as AlexNet [30], VGG-16 [42], and GoogLeNet [45]. The FCN decoder is composed of several deconvolutional layers that are learned during training followed by a final deconvolutional layer that is fixed to bilinear interpolation. One of the key insights of FCN is to equip the network with skip connections that upsample features from lower layers and concatenate them with the decoder output to provide the classifier with both course semantic information and local appearance information.

Noe et al. propose DeconvNet [37] with a more extensive decoder than FCN. The decoder is symmetric with respect to the number and feature sizes of the encoder. In addition to deconvolution, the DeconvNet decoder network uses unpooling layers. Unpooling layers save the pooling indices of the corresponding pooling layer in the encoder to directly upsample feature activations to the spatial location they were extracted from. Unpooling produces a sparse feature map and therefore is followed by a deconvolutional layer to densify the activations. The authors claim that the unpooling operation captures *example-specific* structures while the deconvolutional layers tend to capture *class-specific* shapes; unpooling traces strong activations back to their original location while deconvolution amplifies activations related to the target class and suppresses others. Training is further improved through the addition of batch normalization [24], which reduces the internal covariate shift by normalizing input feature activations to a Gaussian distribution between layers. Because DeconvNet uses two fully-connected layers in its encoder, it is relatively large in memory compared to FCN.

Motivated to reduce the number of parameters and amount of memory required by segmentation networks, Badrinarayanan et al. propose the SegNet [4, 3] architecture. The SegNet encoder is topologically identical to the 13 convolutional layers of VGG-16, but in contrast to FCN and DeconvNet, the decoder contains only upsampling (unpooling) operations and convolution therefore eliminating deconvolution altogether. Architectures that store and use feature maps from an encoder during classification outperform SegNet but require more memory during inference. SegNet also uses batch normalization after every convolutional layer to improve training. Kendell et al. [27] build on the SegNet model and improve performance through adding dropout [44] at inference time.

Paszke et al. iterate on the SegNet architecture to create an even more efficient model called ENet [39] that is 18x faster and has 79x less parameters than SegNet. They construct their network using "bottleneck" modules that use skip connections inspired by ResNet [19] in parallel with varying forms of convolution. The initial bottleneck module used at the front of the network



(a) Initial bottleneck module

(b) Regular bottleneck module

Figure 3.1: ENet bottleneck modules [39]

and the bottleneck modules used throughout the rest of the network are shown in figure 3.1. The convolution operation for each module is either a dilated, 3x3, or an "asymmetric" 5x5 convolution. An asymmetric or flattened convolution, originally proposed in [46, 26], refers to an nxn convolution decomposed into two smaller filters that succeed each other; one nx1 convolution followed by one 1xn convolution. The cost of executing a 5x5 asymmetric convolution is comparable to a single 3x3 convolution while also increasing the receptive field. While ENet currently has less than state-of-the-art segmentation performance on the Cityscapes [10] dataset, it has the fastest reported inference time at just 13 ms.

3.2 Multimodal Feature Learning

The most common types of modalities that are combined are from the image space, but there are also approaches that combine other forms of data with images to learn different tasks. Nojavanasghari et al. [38] use visual, acoustic, and text features in a multimodal fusion network to predict persuasiveness using the Persuasive Opinion Multimedia (POM) dataset. Kim et al. [28] learn from vision and language information to achieve state-of-the-art on open-ended as well as multiple-choice visual QA tasks. Christoudias et al. [8] explore the problem of exploiting multiple modalities during inference time when they were not present in the training set including color and grayscale images, multi-resolution imagery, and color images and text.

3.2.1 Deep Learning using RGB-D Data

Several multimodal CNN implementations learn vision tasks specifically by combining features from color and depth images. We explore work in this area applied to both object detection and segmentation as they are the most common RGB-D vision applications. Multimodal RGB-D approaches differ in several ways including choice of depth representation, method of modality combination, location or depth of modality fusion, and whether the considered modalities are available at train time, test time, or both.

Some methods use the gradients from a pre-trained and fixed RGB network to guide training of a depth network. Gupta et al. call this "cross-modal distillation" and use the representations learned from RGB images as a supervisory signal for learning depth representations [17, 16]. Hoffman et al. [21] train CNNs on color and depth images separately such that they can use depth information during inference to boost performance of classes that had no labeled depth training data. In all of these methods, the HHA [15] representation was applied to depth information with each pixel encoding the horizontal disparity, height above ground, and angle with respect to gravity of the surface. This approach has advantages but imposes overhead for using this representation during inference. Hoffman [22] addresses this by developing an object detection framework where at test time a color image is presented to a traditional CNN as well as an additional "hallucination network" that has been trained to extract depth features from RGB images. In this way they benefit from using features unique to depth but do not bear the computational cost of computing the HHA representation during inference. All of the depth networks used in these implementations are initialized from pre-trained RGB weights. While this helps the network learn initially, it may not give the depth network the chance to learn features as unique as it would when training from scratch.

There are also several methods for combining independent modality features at different points in a CNN. Wang et al. [48] explore multimodal architectures with a carefully designed layer that learns both discriminative and complementary features from each modality. This layer is at the end of the two networks and attempts to project the high-order features into a shared feature space before performing object detection. Eitel et al. [13] process depth and color modalities in separate CNN streams and joins them in a late fusion network to perform object detection on Washington's RGB-D dataset [31]. This method does intentionally keep features from each modality separate

before fusing them in fully-connected layers, which is an effective method for combination but the inference speed of this approach is slower than fully convolutional architectures. Another method of combination is channel-wise concatenation of the modalities before presenting them to the network, used in [15].

The motivation behind the recently proposed architecture FuseNet [18] is closely related to this research. The authors use a SegNet-style architecture with two encoder branches: one for depth and one for RGB. The authors use fusion layers, which are implemented as element-wise sums between the two encoder branch feature maps at various points in the network. They refer to this as either sparse or dense fusion depending on how many times this layer is embedded between the two encoders. Importantly, the feature maps from the depth encoder are summed and applied to the fusion layers of the RGB encoder, but not the other way around. Using this strategy the RGB encoder fusion layers learn from features embedded with depth information. As the shared representation moves through the RGB encoder, it is continually being supplemented by unique and increasingly abstract depth features. They achieve competitive results on the indoor SUN RGB-D segmentation dataset [43] but again do not allow the network to learn completely separate features before combining them into a joint representation.

3.3 Summary

There are many approaches that use a CNN to perform semantic segmentation, and a smaller subset that use RGB-D data. While using depth information during training alone can improve segmentation performance in various ways, approaches that use depth during both training and inference have shown better performance. Further, many segmentation models have a large memory footprint and are not suitable for real-time execution. ENet is the fastest segmentation architecture tested on the Cityscapes dataset and is built on the sturdy foundation of SegNet, but does not exploit depth information during training or inference. This gap creates the need for a model architecture that can execute in real-time and also use a multimodal representation of the scene to improve segmentation performance, which is a contribution of this research.

Chapter 4

Research Design

This chapter presents the overall research approach of this thesis. First we formalize learning the task of semantic segmentation using a CNN. MM-ENet is presented next with an approach to extending RGB segmentation models to operate on multimodal data. Different methods of feature extraction and concatenation are presented followed by a closer look at how these different operations affect the network architecture.

4.1 Learning Semantic Segmentation

Since most modern neural networks are trained using maximum likelihood, the cost function is in the form of a negative log-likelihood, or cross-entropy between the training data and the model distribution [14]. While generative models estimate the joint probability distribution $p(\boldsymbol{y}, \boldsymbol{x})$, discriminative models used for semantic segmentation use conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$. Qualitatively, the conditional probablity is estimating the probability of label $\boldsymbol{y}$ when presented with feature vector $\boldsymbol{x}$. The negative log-likelihood as a cost function is formalized in Equation 4.1.

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y}|\boldsymbol{x}) \tag{4.1}$$

When training for the task of semantic segmentation, we follow FCN by computing the spatial loss over the dimensions of the output layer $\ell(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{ij} \ell'(\mathbf{x}_{i,j};\boldsymbol{\theta})$. Using this formulation, gradient descent becomes a sum over the spatial gradients. This formulation is useful for many

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
$4\times$ bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

Figure 4.1: ENet blob dimensions for 512x512 input [39]

reasons. One reason is that it can be viewed as taking $\ell^{'}$ for every receptive field in the final layer as a mini-batch instead of only having the means to compute one loss value at every training step. Formulating the loss in this manner means that accumulating gradients from multiple images works well, and can implicitly help correct class imbalances. This sum over the spatial gradients also implies that patchwise training, or only taking a subset of all available $\ell^{'}$, is equivalent to sampling the actual loss.

Another direct way of addressing class imbalances in the training set is to weight each pixel of the softmax output corresponding to a particular ground truth class by some variant of its inverse pixel-level frequency across the training set. Although there are different variants of this approach, we use median frequency balancing [12]. The median frequency balancing approach weights each pixel by $\alpha_c = median\_freq/freq(c)$, where $freq(c)$ is the number of pixels of class c divided by the total number of pixels in images where c is present, and $median\_freq$ is the median of these frequencies.

4.2 MM-ENet: Extension to A Multimodal Architecture

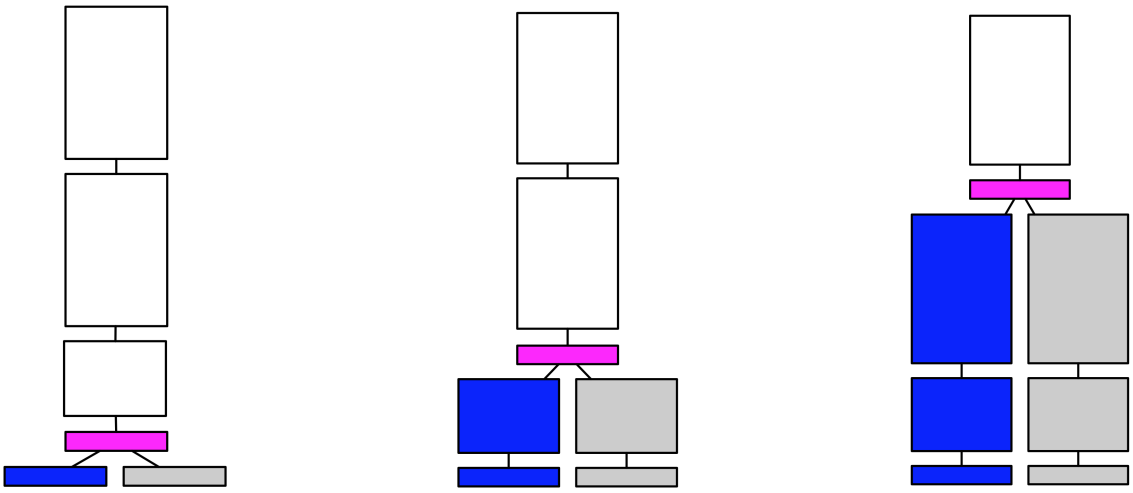
Before introducing the MM-ENet architectures, we look at the ENet architecture in more detail to motivate the choice of split location. Figure 4.1 shows the dimensions of a feature blob

as it moves through an ENet model. This particular example is for an input image of 512x512. From this table it can be seen that the resolution of the feature maps at the output of the encoder are downsampled by a factor of 8 with respect to the input resolution. At the beginning of each phase of bottleneck modules in the encoder there is a downsampling operation, with the exception of bottleneck 3.x, which is the last phase of the encoder. The beginning of each of these phases - before the downsampling operation - is a potential point of interest for fusion. These locations are where the MM-ENet architectures combine features from individual modalities.

4.2.1 Split Architecture

In order to adapt a network for multimodal learning, we duplicate all layers up until a fusion point so that each branch becomes an individual modality feature extraction stream. From the point of fusion to the end of the encoder is the "fusion" network, which retains it's architecture but now operates on a joint representation of the data. When duplicating a layer, all of its parameters and dimensions are kept constant. This model extension approach means that the feature extraction streams will be symmetric. While it may be possible to tune the depth extraction branch to achieve higher performance by exploiting certain aspects of depth data, e.g. modifications to the receptive field of the first couple layers or number of filters in the convolutional layers, these variations were not explored here. Symmetric feature extraction branches make the network easier to train and more amenable for comparison to the baseline model.

An exhaustive approach to exploring multimodal architectures would be to instantiate and train a network at every possible point of duplication in the encoder, making a total of 24 different architectures to study. While this may produce interesting results at some unintuitive fusion locations, it is likely that the best results will come from fusion at points prior to a downsampling operation. This is because downsampling implies that some information is lost, such as the exact shape of an edge. It is well known that one major advantage of downsampling is that it allows neurons in the subsequent convolutional layer to have a wider effective receptive field, enabling them to draw more context from the image. This context could be the difference between recognizing a rider on a bike versus a pedestrian on the sidewalk. While channelwise concatenation of features before or after a downsampling layer has the same effect, the elementwise operations are sensitive to the operation resolution. It is for this reason fusion should occur prior to a downsampling layer, resulting in three possible architectures.



(a) MM-ENet (b) MM-ENet-Split1 (c) MM-ENet-Split2

Figure 4.2: Modifying ENet to operate on multimodal data

These three architectures are enumerated as MM-ENet, MM-ENet-Split1, MM-ENet-Split2. MM-ENet, showin in Figure 4.2a, is the earliest possible fusion location and MM-ENet-Split2, shown in Figure 4.2c, fuses the modalities just before the last downsampling location. Figure 4.2b shows the MM-ENet-Split1 architecture that fuses the two inputs prior to the second downsampling point in the middle of the encoder network. Another possible variant of combining disparity and color data would be to feed it to the network as a 4-D image, where the first three channels are RGB channels and the fourth channel is the disparity image. Concatenating the inputs this early is the naive approach to modality combination and does not allow the network to take advantages of features from individual modalities, but instead combines them immediately. This approach was not implemented as does not learn from each modality separately at any point.

4.2.2 Feature Extraction and Fusion

To explore the best way to fuse features from both modalities, three different fusion operations were implemented: channelwise concatenation, elementwise addition, and elementwise multiplication. Here we discuss what each of them represent from a qualitative perspective as well as their effect on the network size and structure. These three combination methods are shown in Figure 4.3.

The first and perhaps most obvious way to fuse features from the two modality streams is to simply concatenate them, doubling the number of channels in the feature blob at this point in

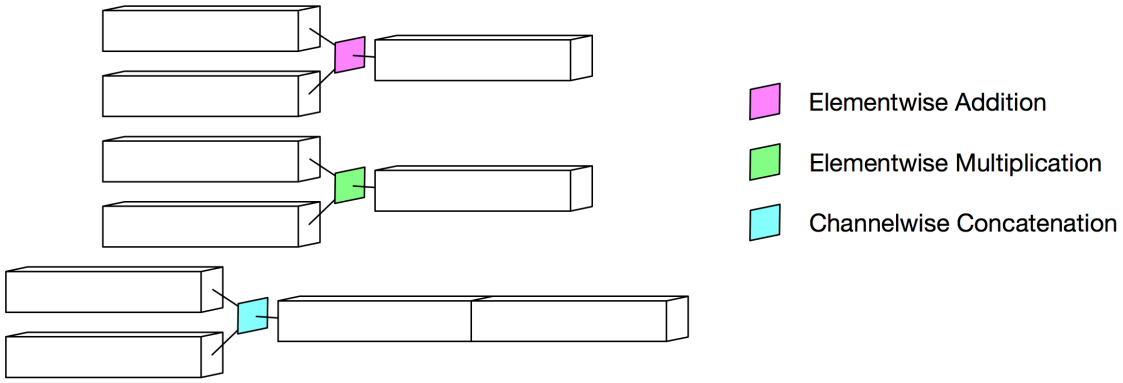


Figure 4.3: Modality fusion approaches

the network. Concatenation in effect is providing the fusion network with twice as many features to learn from. Convolutional layers operate across all channels, which means that the number of blob channels prior to entering the fusion network is doubled. This increase in blob size is mainly due to the skip connection in each ENet bottleneck module that keeps the original number of input channels regardless of the number of outputs from the parallel convolution branch. The number of feature maps remains doubled until the blob must be upsampled in the decoder. Since the decoder uses the SegNet strategy of unpooling based on pooling indices of layers in the encoder, it must upsample to the same amount of feature channels that were present in the corresponding encoder layer. This constraint means that when using concatenation as a fusion operation, the dimension of blobs moving through the network is doubled at almost every location. This architecture increases the amount of memory needed for training and slightly increases the amount of memory needed for inference, but does not affect other parameters of the network such as the number of filters in convolutional layers. It only means that every convolution operation must operate across twice as many feature channels, slowing down training. The unavoidable increase in memory requirements is one drawback to using concatenation for fusion.

A second method of modality fusion is elementwise addition. The ENet bottleneck module uses elementwise addition to combine the convolved and identity feature maps each time it is used, so this operation is already common in the original model. The motivation behind using this layer is that it can act as an OR operation. That is, elementwise addition lets features pass through if they are present in either of the two individual modality's feature maps. Viewing fusion in this way

has some useful properties that can guide optimal placement of this layer. The hypothesis is that since we expect the semantic information present in both of these modalities to be related but their noise sources to be mostly independent [48], this layer could end up including both sources of noise if it is too close to the front of the network. Fusion this early may make the model more robust to noise during training, but if the noise is overwhelming it could negatively affect performance.

The final method of modality fusion is elementwise multiplication. Like elementwise addition, this does not increase the dimension of the feature maps but keeps it constant with respect to the original ENet architecture. If elementwise addition can be thought of as an OR operation, multiplication can be thought of as an AND. That is, it will only let features pass through that are present in both modalities. We expect that this would be more appropriately placed towards the front of the network, as it could help filter out some of the independent noise sources. If placed too close to the end of the encoder, it could destroy the ability to take advantage of unique features since strong activations from an individual modality may be zeroed out.

4.2.3 Training Using Multimodal Data

Training ENet and therefore subsequently MM-ENet happens in two phases: pre-training the encoder and then using those weights to initialize and train the entire network, including the decoder. While this is not an uncommon strategy for training segmentation networks, it particularly lends itself to training with multimodal data. Using this pre-training approach, the encoder has a chance to learn how to optimally combine features before having to learn how to upsample them to the original spatial resolution. The pre-training process uses the same cost function as the full network, but since the input image has been downsampled by a factor of eight by the end of the encoder, we take the loss with respect to a ground truth image that has also been downsampled by a factor of eight. All other training and network hyperparameters are identical to those used in the full network.

4.3 Summary

Taking all three fusion locations as well as all three fusion operations into account produces nine different multimodal CNN architectures. These are enumerated as MM-ENet, MM-ENet-ADD, MM-ENet-MUL, MM-ENet-Split1, MM-ENet-Split1-ADD, MM-ENet-Split1-MUL, MM-ENet-Split2,

MM-ENet-Split2-ADD, MM-ENet-Split2-MUL. In this research we focus on the three Split1 architectures to choose a location appropriate for investigating the effect of the fusion operation.

Chapter 5

Experimental Setup

Deep learning algorithms can be abstracted into four components: dataset specification, choice of model, choice of cost function, and optimization or training procedure. Here we discuss the Cityscapes dataset [10] as well as details pertaining to the disparity images used for training. Following this is the software stack and the full pipeline a model must go through from initial training to deployment. This chapter is also where we discuss the hardware configuration and the advantage of distributed computing systems with multi-GPU nodes. Finally we submit details of the solver used for training and relevant hyperparameters such as learning rate and momentum.

5.1 Cityscapes Dataset

The Cityscapes dataset is known for being one of the most challenging datasets for semantic segmentation. Cityscapes focuses on daytime urban street scenes taken from 50 cities in

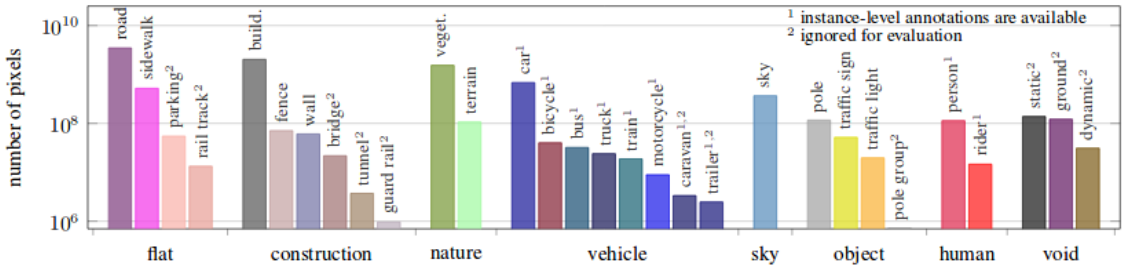


Figure 5.1: Cityscapes class distribution [10]

good/medium weather conditions. Scenes are selected from several hours of raw video to maximize variance in the scene layout and background. There are 30 labeled classes in 5000 fine-grain annotated images of which 2975 are used for training, 500 are used for validation, and the remaining 1525 are used for testing. Cityscapes does not release the annotations for the test set in order to ensure that they cannot be used for supplemental training. To be evaluated on the test set, a submission must be made to an evaluation server hosted by the dataset curators. Many of the benchmarked solutions on Cityscapes operate on full resolution images. For this reason we use the validation set to compare the performance of MM-ENet models to an ENet model that has also been trained on Cityscapes at this resolution.

Cityscapes uses the standard metric for evaluating the performance of a segmentation algorithm, Intersection-over-Union (IoU). As the name suggests, this is calculated by taking the intersection of the inferred segmentation and the ground truth image and dividing it by their union. Since each class belongs to a category, Cityscapes evaluates both IoU_{class} and $\text{IoU}_{category}$. Additionally, submissions are compared using a second metric called instance-level IoU. "Global" IoU is known for biasing towards object classes that take up a large area in the image. Instance-level IoU (iIoU) weights each pixel by the ratio of the class's average instance size to the size of the ground truth instance. Note that false-positive (FP) pixels do not require this weighting as they do not belong to any instance. This second metric is not to be confused with differentiating between separate object instances, which is a finer level of semantic granularity. Equations for computing these two metrics are shown in Equations 5.1 and 5.2.

$$\text{IoU} = \frac{A \cap B}{A \cup B} = \frac{TP}{TP + FP + FN} \tag{5.1}$$

$$\text{iIoU} = \frac{iTP}{iTP + FP + iFN} \tag{5.2}$$

5.1.1 Disparity Images

Cityscapes provides disparity images paired with each image in the training set, taken from a stereo camera and computed using the semi-global block matching (SGM) algorithm [20]. SGM, like many stereo vision algorithms, consists of multiple and sometimes iterative operations that produce a refined and acceptable estimation of depth for a given application. Scharstein and

Szeliski [41] decompose two-frame stereo methods into four steps: (1) matching cost computation, (2) cost aggregation, (3) disparity computation/optimization, and (4) disparity refinement. SGM is a widely used approach and has several parameters that can be tuned to produce a representation suitable for a specific task.

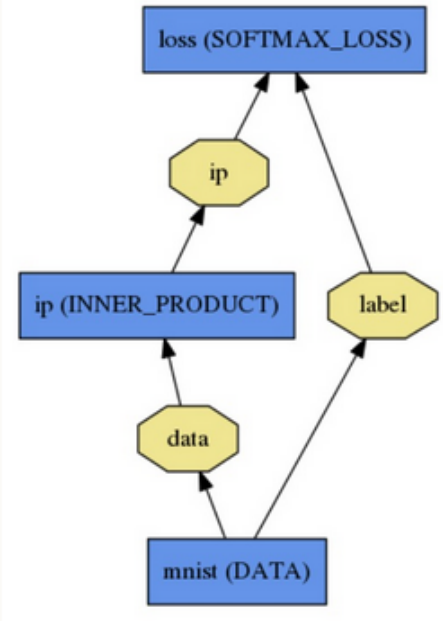
It is worth noting that the method of disparity calculation has a strong impact on what the network is able to learn. Artifacts related to particular stereo algorithms embed themselves in the representation and the network sometimes learns to use these artifacts as features. Learning to use artifcats as features could improve recognition of certain objects, but of course certain artifacts, such as heavily distorted edges on particular object classes due to a sub-optimal SGM parameter, could prevent it from learning consistent representations. Cityscapes provides pre-generated disparity images so that methods using depth information can directly compare their results. They do not submit any other details related to the generation of the disparity images. It is not clear that representations learned using a particular stereo vision algorithm would generalize to all forms of depth information, and in fact it is very likely that they would not.

5.2 Training Pipeline

5.2.1 Software Stack

As the field of deep learning is still a fairly new research area, there are several software frameworks with varying levels of support and capabilities that can be used to train and deploy these networks. Some of the most widely used frameworks are Caffe [25] from BVLC, Torch [9] by Facebook, Tensorflow [2] by Google, and MXNet [7] from Amazon, to name a few.

In this work we adopt one of the first and still widley-used deep learning frameworks, Caffe. Caffe was developed by a PhD student at UC Berekeley in 2014 and is now an open-source project with many active contributors. Caffe defines a network layer-by-layer in a custom schema that includes the entire architecture as well as loss functions and terms. A network solver is specified separately to intentionally decouple modeling and optimization. The two fundamental structures in the Caffe framework are the *layer* and the *blob*. A layer is a node on the graph and defines a form of computation. Example layer types are data layers, convolutional layers, loss layers, pooling layers, and batch normalization layers. Blobs are how Caffe represents intermediate computations as they move through the graph. A blob is a tensor, or an n-dimensionsal array, that has four dimensions:



(a) Caffe computation graph

```
name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

(b) Caffe protobuffer file

Figure 5.2: Building a model with Caffe

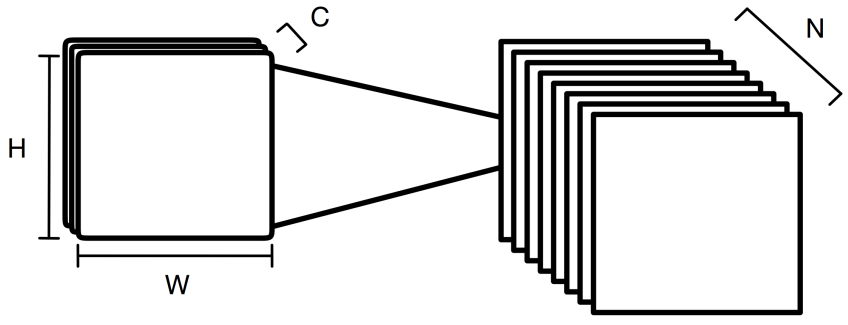


Figure 5.3: Caffe blob data structure

NxCxHxW where N is the number of images, C is the number of channels, H is image height, and W is image width. For example, a training batch of eight 512x512 RGB (3-channel) images is fed into the network as an 8x3x512x512 dimensional blob. A convolutional layer that takes this blob as input and has 16 learnable filters creates an output blob with dimensions 8x16x512x512 (assuming stride and padding were configured appropriately). An example of a simple Caffe network to perform logistic regression using the MNIST dataset is shown in Figure 5.2 and a visualization of the blob data structure is shown in Figure 5.3. In this model there is one fully-connected, or inner product layer, containing two neurons.

It was necessary to develop specialized layers in Caffe to train multimodal networks. The Caffe data layer is built to have many different features such as resizing or cropping input images on the fly. It also has the capability to shuffle the dataset before each epoch to help avoid overfitting the training set. However, a Caffe data layer is restricted to two output blobs: the data and the label. If each modality stream was fed by an independent data layer, the shuffling function would produce a different ordering for each input, therefore throwing off the synchronization between images. To retain the ability to shuffle the training images but still ensure synchronization between inputs, we develop a multimodal data layer capable of producing three outputs including two data blobs coupled with one label and integrate it into the Caffe framework.

The second specialized layer developed for Caffe was the spatial dropout layer. In the open-source version of ENet in caffe, the spatial dropout layer was written in python. This design choice causes two issues during training: 1. Python layers can only be implemented on the CPU therefore causing a memory transaction every time a blob must pass through one in the network and 2. Python layers can not be used during multi-gpu training. These are both limitations of the Caffe framework. We developed a C++ implementation of this layer and added it to Caffe to address both of these limitations.

5.2.2 Model Deployment

When deploying a trained model, several steps must be taken to ensure that it functions properly. Of significant importance is configuring the batch normalization layers for inference. During training, all activations from an input blob to the batch normalization layer are normalized to have a mean of 0 and variance of 1. The values needed to shift the distribution within these constraints are recomputed for each batch, and therefore at the end of training still contain values

from the most recent image batch. To calculate parameters suitable for inference, it is necessary to run batch normalization across the entire training set, setting the final parameters so that the activations over the entire training set fall within this distribution. Shifting the distribution in this way could inhibit a model's ability to generalize across datasets, but usually improves performance within a dataset.

5.3 Hardware Configuration

It is essential to use GPUs to accelerate DNN training. The Palmetto Cluster [1] was the main training platform used for all experiments. In addition, networks were trained on local machines containing either an Nvidia GTX 1080 or an Nvidia Titan X (Pascal). The latest additions to Palmetto include a 40-node phase with two Nvidia P100 GPUs per node. When training on different platforms, it is necessary to modify both the batch size and training iterations to achieve the same number of epochs on the dataset. Palmetto additionally supports the use of multiple GPUs on a single node. While Caffe does support multi-GPU training, it does not currently support distributed training across multiple nodes.

Caffe uses a tree-reduction strategy when training on multiple GPUs. For example, when using 4 GPUs on a single node for training, the network is fully duplicated across all GPUs and at every iteration each GPU receives a unique batch of images from the training set. A forward pass is done on each GPU and then GPU 0:1 and 2:3 exchange gradients followed by GPU 0:2. GPU 0 will compute the updated model and send it back out $0 \rightarrow 2$, and then $0 \rightarrow 1$ and $2 \rightarrow 3$. The amount of acceleration gained from using multiple GPUs is highly dependent on the interconnection strategy, i.e. whether or not GPUs are connected through a P2P connection, a PCIe host connection, or have to communicate through the CPU. On the Palmetto Cluster phase 16 with Nvidia P100s, the primary phase used during training, the GPUs are connected through a PCIe host connection. While this is not the best case scenario, it does make multi-GPU training feasible and faster than using a single GPU.

5.4 Hyperparameters

Following ENet and many other recent CNN networks, we use the Adam [29] solver as the model optimization algorithm. Adam is a first-order gradient-based optimizer for stochastic objective functions. The Adam solver computes adaptive learning rates for different parameters using estimates of the first and second moment of the gradient. Adam typically requires little tuning and is efficient in memory.

Hyperparemeters for the solver are based off those reported in ENet. We use a learning rate of 5×10^{-4} with momentum of 0.9 and train for close to 100 epochs. L2 regularization is used in the form of weight decay with a decay value of 2×10^{-4} . To calculate the number of iterations needed to train for a certain amount of epochs, the batch size, number of GPUs, dataset size, and iteration size must be determined. The iteration size is how many forward passes to accumulate gradients for before applying them with a backpropogation step. Accumulating more gradients before applying an update can help smooth the training loss. The formula for computing training iterations on a particular platform is shown in Equation 5.3.

$$\#\ \text{Epochs} = \text{Iterations} * \frac{\text{Batch Size} * \text{Iteration Size} * \#\ \text{GPUs}}{\text{Dataset Size}} \tag{5.3}$$

5.5 Summary

In this section information related to the hardware and software used to train and deploy these models was presented along with details needed to replicate all experiments. These details covered the solver used for optimization and hyperparameters including the learning rate, momentum, and regularization approach. The Cityscapes dataset as well as how the additional training modality, disparity images, were generated is also discussed as well as the implications of using different disparity generation algorithms.

Chapter 6

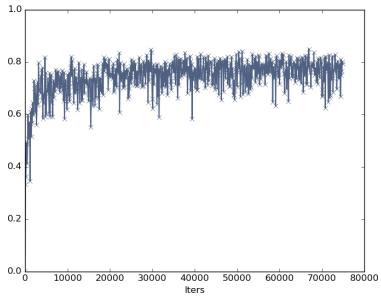
Results

In this chapter we present results from combining features using three modality fusion approaches with the MM-ENet-Split1 architecture. The focus of these results is on analyzing the difference between the three approaches to modality fusion. We leave analyzing the effect of fusion location to future work.

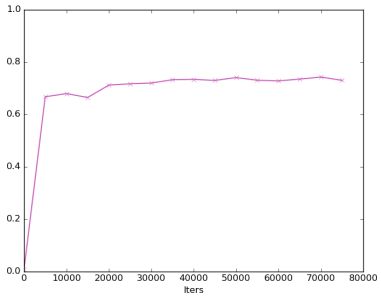
First, accuracy and loss curves during training are shown and compared to confirm that all models converged to a valid solution. Next, a quantitative analysis on the Cityscapes validation set of 500 512x256 images is given. Models are compared to the baseline ENet model using four different metrics: class-level IoU, class-level iIoU, category-level IoU, and category iIoU. Equations for calculating these metrics are presented in Chapter 5. Following this evaluation is a qualitative analysis that compares the inference results of all models. Feature activation maps from convolutional layers are directly extracted from different points in the network after being presented with a test image. These feature maps are presented visually to investigate the differences in features learned by each modality extraction branch.

6.1 Training Results

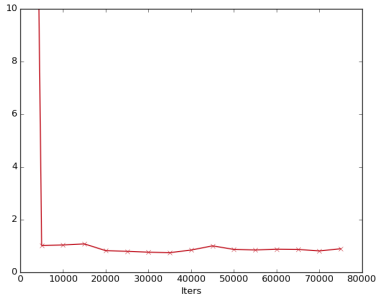
Figure 6.1 shows the training curves for each architecture. For each model the training accuracy, test (validation) accuracy, and test (validation) loss is shown. These are the most relevant metrics for showing that the network trained properly, and most importantly, did not overfit the dataset. If the training accuracy continues to rise but the test loss also begins to increase, that



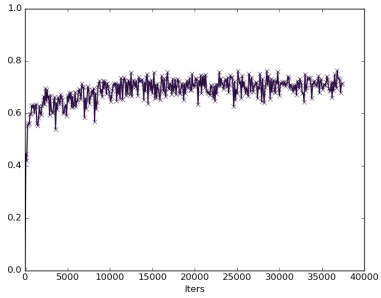
(a) MM-ENet-Split1 Train Accuracy



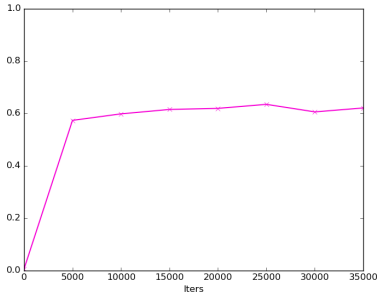
(b) MM-ENet-Split1 Test Accuracy



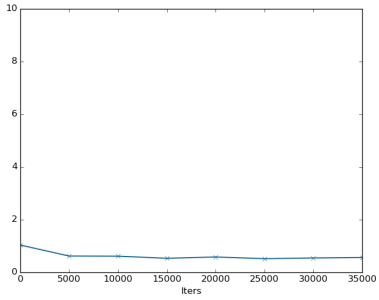
(c) MM-ENet-Split1 Test Loss



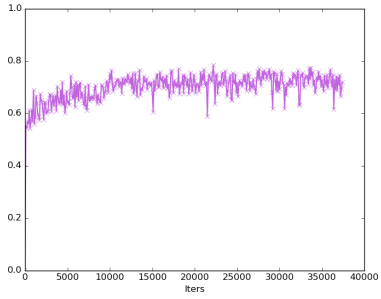
(d) MM-ENet-Split1-ADD Train Accuracy



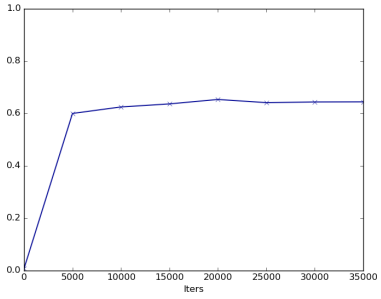
(e) MM-ENet-Split1-ADD Test Accuracy



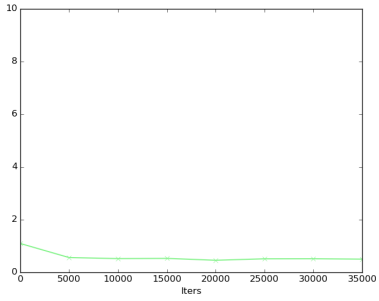
(f) MM-ENet-Split1-ADD Test Loss



(g) MM-ENet-Split1-MUL Train Accuracy



(h) MM-ENet-Split1-MUL Test Accuracy



(i) MM-ENet-Split1-MUL Test Loss

Figure 6.1: Training curves for full networks

implies that the model is memorizing the dataset instead of learning representations of the different classes. Test accuracy can also be used against train accuracy to prove this point, but the loss value is usually more helpful. When presenting these three graphs for all of the architectures, it can be seen that all architectures both converge and avoid overfitting the training set.

6.2 Segmentation Results on Cityscapes

Figure 6.2 shows the class-level IoU results on the Cityscapes validation set. All 19 classes are considered for evaluation. The baseline ENet model outperforms all multimodal networks on average, but does not have the highest IoU in every category. MM-ENet-Split1, the model that uses concatenation to fuse features, is the highest performing model on the building, pole, vegetation, sky, and car classes. Interestingly, this same network has a 0 score on the traffic light, traffic sign, person, rider, and motorcycle classes. Every model effectively has a 0 score on the bus and train classes as they are not frequent enough in the validation set. More instances of these classes exist in the test set kept by Cityscapes and so are more useful when evaluating there. On almost every class MM-ENet-Split1-MUL outperforms MM-ENet-Split1-ADD, and also outperforms it on average. MM-ENet-Split1-MUL has the highest performance out of all the multimodal Split1 architectures.

The fact that MM-ENet-Split1 outperformed the baseline ENet model on some classes and scored 0 on others is interesting. It appears that when this model concatenates the outputs from each of the feature extraction branches it is able to reason about large object classes like sky and road very well but is completely unable to learn features to segment spatially small object classes. This behavior could arise for a variety of reasons. One plausible reason is that since both sets of features are provided to the fusion network, large object classes that are likely to have low variance produce homogeneous activation regions in both branches. The succeeding convolutional layers may be learning to look for consistent features instead of complementary features. This conjecture would also imply the converse; when using a low resolution like 512x256, spatially small classes, especially ones that have a very high variance like a person, produce features that are complementary but not very consistent. We look at this in more detail in the qualitative analysis.

Figure 6.3 shows the class-level iIoU results. Classes that do not have object instance labels are not included. Fusion via multiplication still appears to be the highest-performing multimodal architecture on average and ENet maintains the highest average score. In contrast to the non-

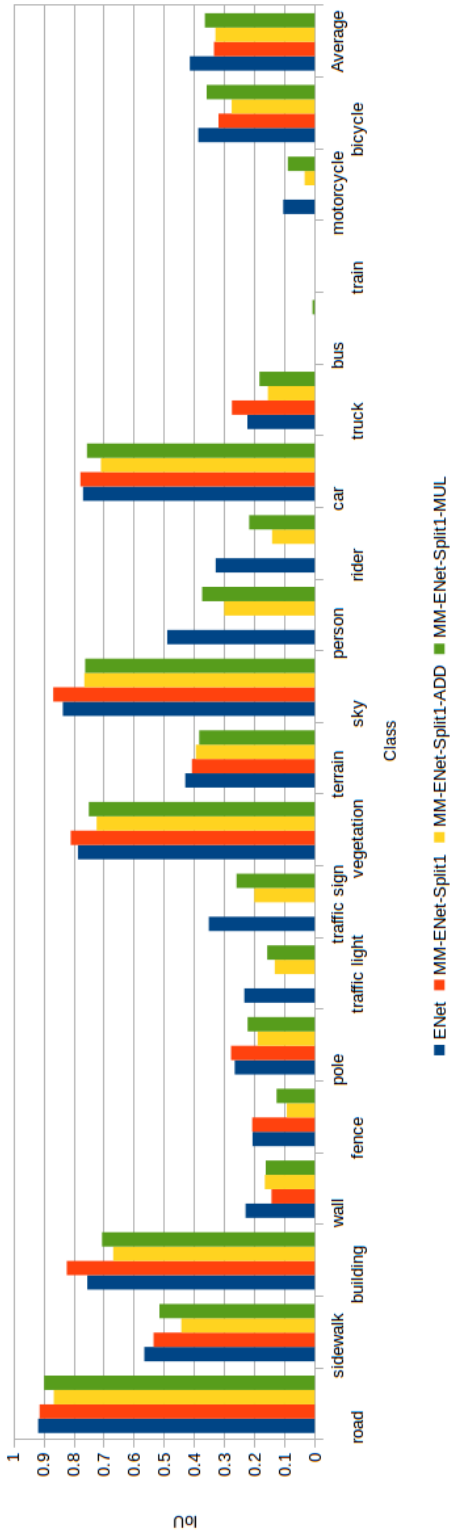


Figure 6.2: IoU comparison of 19 classes

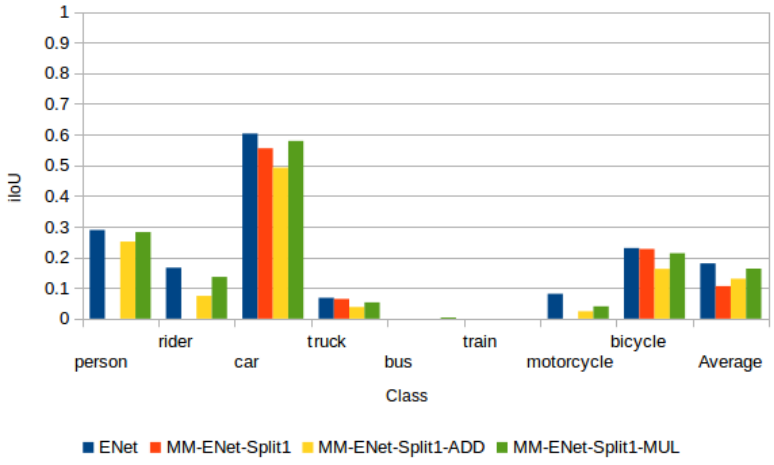


Figure 6.3: iIoU comparison of 8 classes

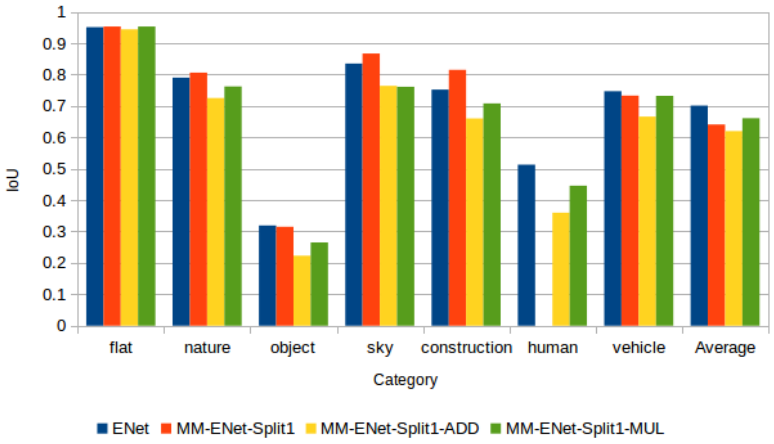


Figure 6.4: IoU comparison of 7 categories

weighted IoU results, MM-ENet-Split1 does not hold the highest score on any individual class. These scores are further confirmation that this model is biased towards large, spatially consistent classes and was not able to learn to segment objects with large inter-class variance.

We now move forward to present category-level segmentation results. The mapping from class to category can be found in section 5.1. Figure 6.4 shows the category-level IoU results. These scores are consistent with the class-level results in that ENet has the best overall performance, MM-ENet-Split1-MUL is the best performing multimodal architecture, and while MM-ENet-Split1 attains high performance on many categories, its lack of ability to identify and segment humans gives it a low average performance. While MM-ENet-Split1-ADD had the lowest average class-level

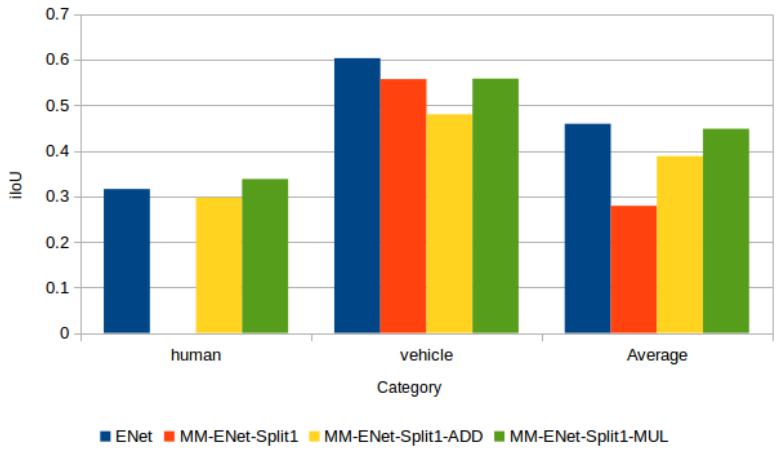


Figure 6.5: iIoU comparison of 2 categories

IoU score, here it slightly outperforms MM-ENet-Split1.

Figure 6.5 presents category-level iIoU results. This metric provides results that allow for a closer comparison of what each model learns. Note that humans and vehicles are the only categories for which there are labeled instances. While MM-ENet-Split1-MUL does not have the higest average iIoU score, it outperforms the baseline ENet model in the weighted human segmentation category. Looking at Equation 5.2, it can be seen that false positives are not weighted as they do not belong to a particular object instance. This equation suggests that when using multiplication as a fusion operation, there are less false positives than when using RGB alone. The multiplication operation was expected to act as an AND gate and therefore discard features that were not present in both modalities. It appears that MM-ENet-Split1-MUL has effectively learned to use more than one modality to improve performance with respect to the category-level iIoU metric on humans.

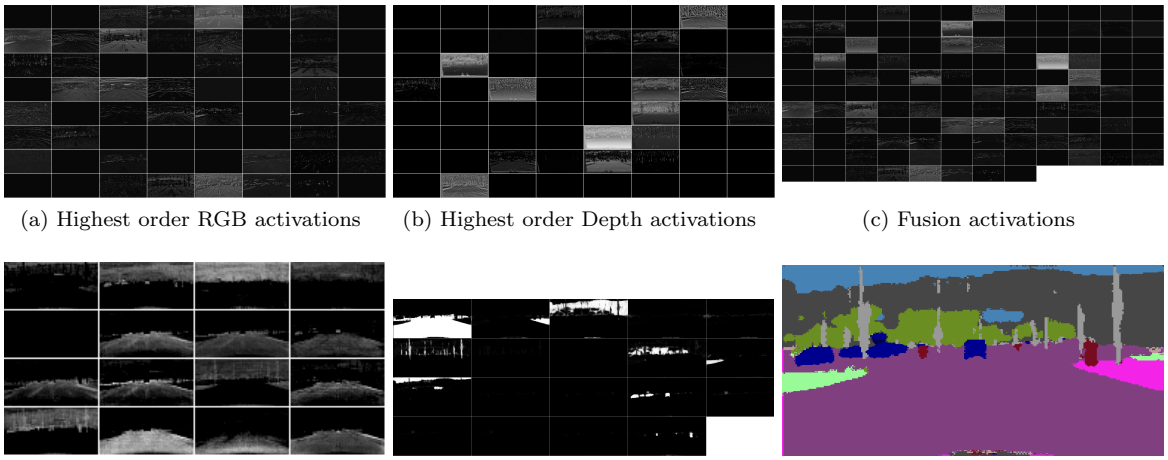
While ENet is the best performing segmentation model on average across all metrics used in this evaluation, MM-ENet-Split1-MUL had interesting properties that helped it perform segmentation on the very important human category. Addition did not turn out to be very useful at this point in the network, but may have better performance if used at a deeper location. Fusion with concatenation helped the network reason about large, consistent classes with low variance but either added too much noise during fusion or was too biased against spatially small object classes to produce competitive iIoU and IoU scores.

Figure 6.6: Image from test set (Munich)

6.3 Multimodal Feature Extraction and Fusion

This section focuses on a qualitative analysis of the models used in this research. To compare different architectures and fusion operations, we extract activation tensors from convolutional layers at strategic points in the network to directly observe what types of features they detect. The first location of feature map extraction is at the final output of each individual extraction branch. Activations at this level should be the most different, and contain features that are complementary if trained properly. Directly after this point, on the other side of the fusion layer, activations are again extracted to show the result of the fusion operation. Finally, the activations at the end of the encoder that represent the joint feature representation and the output feature maps of the decoder are extracted to display the direct probability the network infers for each of the 19 classes. A colored inference image is paired with these activations that is produced by taking the highest probable class at each pixel location and mapping it to a unique color. An image from the test set (not validation set) for which no label exists is used to produce these features. This image is shown in Figure 6.6, which is a challenging scene that contains almost every object class multiple times at different angles, orientations, and occlusion levels.

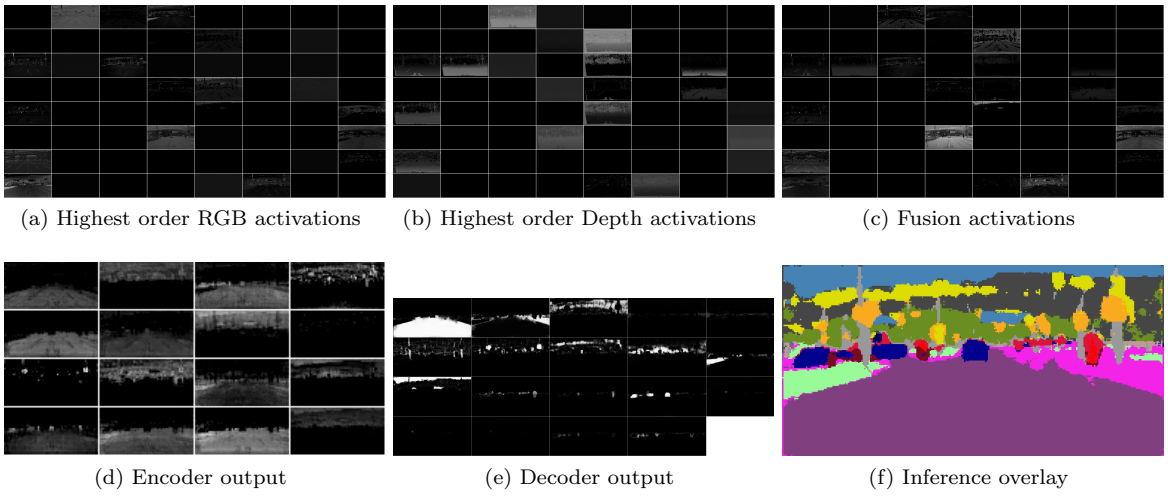
Figure 6.7 visualizes the activations produced by MM-ENet-Split1 when running inference on the aforementioned test image. The most important activations to consider are those at the output of the encoder and decoder. While there are strong activations for areas that likely correspond to the ground or sky, there is no presence of concentrated activations in areas where there would be smaller objects. The decoder probabilities are the most telling of this phenomenon. When fully



(a) Highest order RGB activations (b) Highest order Depth activations (c) Fusion activations

(d) Encoder output (e) Decoder output (f) Inference overlay

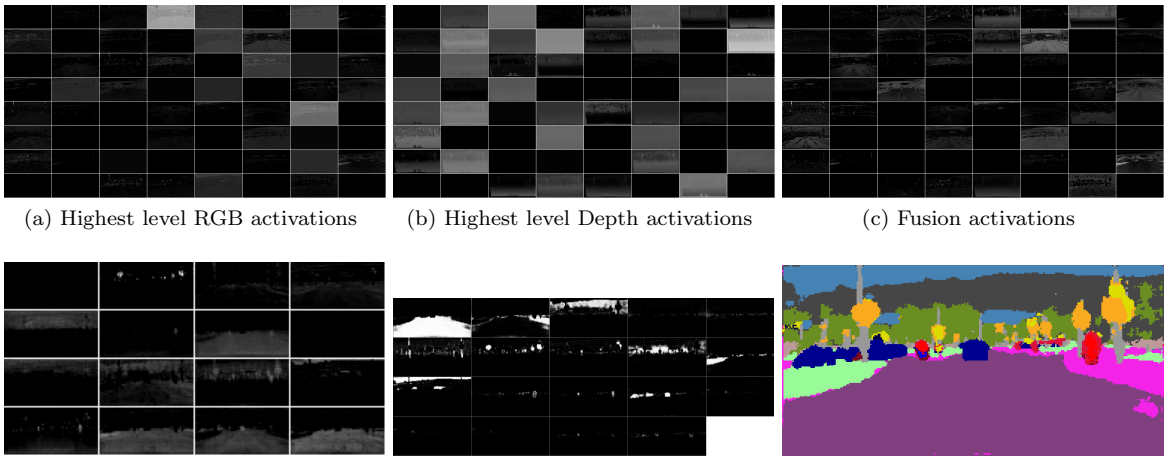
Figure 6.7: Split1 architecture fused using channelwise concatenation



(a) Highest order RGB activations (b) Highest order Depth activations (c) Fusion activations

(d) Encoder output (e) Decoder output (f) Inference overlay

Figure 6.8: Split1 architecture fused using elementwise addition



(a) Highest level RGB activations (b) Highest level Depth activations (c) Fusion activations

(d) Encoder output (e) Decoder output (f) Inference overlay

Figure 6.9: Split1 architecture fused using elementwise multiplication

upsampled back to the original resolution, the channels corresponding to many of the small object classes appear completely blank. The output of the decoder is a 19x512x256 blob and each channel is arranged top to bottom, left to right in the same order as the 19 classes in Figure 6.2. Mapping this to the inference output, it is easy to see that the segmentation performs well on classes that the network learned but is absent from any detection of humans, traffic signs and lights, and bikes and motorcycles.

The more interesting comparison is between the networks using addition or multiplication to fuse features. Figure 6.8 shows the feature maps extracted from MM-ENet-Split1-ADD. While the output of the feature extraction branches are difficult to directly compare, looking at the output of the fusion layer indicates that the activations are relatively small compared to MM-ENet-Split1. While comparing relative activation strength is not a definitive way to compare networks, it does seem to indicate that the fusion activations after elementwise addition are not as "confident" as those at this point in the concatentation network. This could also be a byproduct of the more noisy activations that are present after the fusion layer in the concatenation model. The network does produce some form of sparse activation at the output of the decoder, and when mapped to an inference image does indeed indicate that the network can reason about all object classes to some degree.

The most interesting network to visually analyze is MM-ENet-Split1-MUL, shown in Figure 6.9. The first point of interest is that the output of the depth encoder produces feature maps that

seem to be much more densely populated, which normally may or may not be advantageous. But from the perspective of elementwise multiplication for fusion, the feature maps from this extraction branch appear to be acting as thresholds for letting the RGB features pass through. Another way to interpret this is that the depth branch has learned which feature maps it should amplify as they come from the RGB branch and therefore produces an activation map on each channel that it finds useful. The output of the encoder and decoder also both show that stronger activations are attained for almost every object class. Finally mapping this to the inference image and comparing it to the elementwise addition network, it is possible to see that it performs much better segmentation. Specifically, it seems to more tightly conform to object boundaries, suggesting that the multiplication operation effectively got rid of some of the noise present in each of the two modalities.

6.4 Summary

In this chapter segmentation results across the 19 Cityscapes classes using the MM-ENet-Split1 architecture and all three modality fusion techniques was presented. Qualitative segmentation results were shown in Figures 6.7 - 6.9 and showed that multiplication using elementwise multiplication was most effective at this split location. Results were only presented for the Split1 architecture due to difficulty training with uniform hyperparameters at the other two split locations. Different solvers could have been presented for each split location individually, but comparing the results in this way would not have provided insight directly related to the operation used for modality fusion. In effect, this would have only shown that it was possible to find optimal hyperparameters for each model. Instead, we focus on providing data and answering questions related to the optimal way to fuse RGB and disparity images at the Split1 location.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The goal of this research was to investigate two questions: 1. What is the best level of abstraction to fuse features extracted from depth and RGB color modalities? and 2. What operation for fusing these features into a joint representation best lends itself to performing the task of semantic segmentation? While difficulties related to training the MM-ENet and MM-ENet-Split2 architectures prevented us from directly answering the first question, results from our experiments did help illuminate the efficacy of different fusion operations. When concatenating feature maps prior to the fusion network, large object classes benefit from being reinforced by two different representations. Elementwise addition was capable of learning to recognize all object classes, but when applied to the Split1 architecture, introduced noise that blurred object boundaries in the final segmentation map. Elementwise multiplication turned out to be the best fusion operation for the Split1 architecture and outperformed elementwise addition. While average IoU was still below the baseline ENet model, MM-ENet-Split1-MUL did outperform the baseline ENet model in the weighted iIoU category on humans, which is an important category. Results show that this is because the elementwise multiplication operation likely amplifies some features that are present in the RGB branch while suppressing others.

It is expected that fusion via multiplication will perform better than addition in an early-combination approach such as MM-ENet by supressing noise from each of the modalities, while fusion via addition is expected to increase in performance when used for feature combination deeper

in the network. The results from this experiment also provide guidance when training specifically with disparity images generated using SGM, and can serve as a comparison for other multimodal approaches that either use depth representations other than SGM or other completely separate modalities.

7.2 Contributions

The major contribution of this research was to provide an effective way of adapting real-time segmentation models to operate on more than one image modality. This was accomplished by providing architectures that duplicate the convolutional and bottleneck layers of ENet and join their feature maps prior to one of the three downsampling layers present in the original model. These architectures can be used with any input resolution and addiitonal modalities other than disparity as long as they can be represented as an image and have the same dimension as the RGB input.

To facilitate training on multimodal data, the Caffe framework was extended to support multiple inputs in the same layer through development of a C++ multimodal data layer. This layer preserves the ablity to shuffle the dataset during each epoch, which is crucial to avoiding overfitting. Additionally, to accelerate training time, a spatial dropout layer was designed and integrated into Caffe. The development of this second layer completely eliminates dependency of Caffe on Python and therefore CPU. This makes training on multiple GPUs possible and much more efficient.

7.3 Future Work

Suggestions for future work in this area include finding solvers that are more effective for training the MM-ENet and MM-ENet-Split2 architectures. While separate solvers could have been tuned to work at all three split locations, this would have defeated the purpose of the research. Showing results from networks trained in this way would only prove that it is possible to tune multiple networks for this problem but not answer the question of what the optimal fusion operation to use for combination is. Initializing the depth feature extraction branch from a pre-trained depth-only network would accelerate training and also possibly lead to better overall performance.

A second future work direction would be to repeat this experiment with different disparity generation techniques or more than two image modalities as input. Optical flow is a logical modality

to compare with disparity as it also contains structural information. If the goal is to maintain real-time inference while adapting for multimodal input, the execution time of the algorithm used to generate the additional modality should be considered. While SGM was chosen here due to fast software implementations that are readily available in computer vision libraries, dense optical flow or more sophisticated disparity generation techniques bear a heavy computational cost and may become the bottleneck during inference. It is also for this reason that 512x256 images were used for training and inference as opposed to a higher resolution. While extending this research to use higher-resolution images would be useful, with current hardware limitations it may prevent it from being deployed in a practical system.

A final direction in which this research may be extended is to do more extensive training/validation set exploration, such as k-fold validation. K-fold validation divides all available labeled images in multiple ways so that it can be shown that results are not tied directly to the images used for training, and instead generalize well. The best way to evaluate the performance is to submit results directly the the Cityscapes evaluation server to be compared to the test set images for which no labels exist. While we did not do this here it would surely provide a better comparison to other segmentation models. To be competitive with other state-of-the-art models, the submissions to the evaluation server should also be trained an higher-resolution images.

Bibliography

[1] Palmetto cluster users's guide. `https://www.palmetto.clemson.edu/palmetto/index.html`. Accessed 07-20-17.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for scene segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[4] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

[5] Serge Beucher and Christian Lantuéjoul. Use of watersheds in contour detection. 1979.

[6] Léon Bottou, Yoshua Bengio, and Yann Le Cun. Global training of document processing systems using graph transformer networks. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 489–494. IEEE, 1997.

[7] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[8] C Mario Christoudias, Raquel Urtasun, Mathieu Salzmann, and Trevor Darrell. Learning to recognize objects from unseen modalities. In *European Conference on Computer Vision*, pages 677–691. Springer, 2010.

[9] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[10] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[12] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[13] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust rgb-d object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 681–687. IEEE, 2015.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *European Conference on Computer Vision*, pages 345–360. Springer, 2014.

[16] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. 35-2: Invited paper: Rgb-d image understanding using supervision transfer. *SID Symposium Digest of Technical Papers*, 47(1):444–447, 2016.

[17] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross modal distillation for supervision transfer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[18] Caner Hazirbas, Lingni Ma, Csaba Domokos, and Daniel Cremers. Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *Proc. ACCV*, volume 2, 2016.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[20] Heiko Hirschmüller. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:328–341, 2008.

[21] J. Hoffman, S. Gupta, J. Leong, S. Guadarrama, and T. Darrell. Cross-modal adaptation for rgb-d detection. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5032–5039, May 2016.

[22] Judy Hoffman, Saurabh Gupta, and Trevor Darrell. Learning with side information through modality hallucination. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[23] Steven L Horowitz and Theodosios Pavlidis. Picture segmentation by a tree traversal algorithm. *Journal of the ACM (JACM)*, 23(2):368–388, 1976.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456, 2015.

[25] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[26] Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014.

[27] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

[28] Jin-Hwa Kim, Sang-Woo Lee, Dong-Hyun Kwak, Min-Oh Heo, Jeonghee Kim, Jung-Woo Ha, and Byoung-Tak Zhang. Multimodal Residual Learning for Visual QA. *Advances in Neural Information Processing Systems*, 2016.

[29] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[31] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817–1824. IEEE, 2011.

[32] Yann Le Cun, Leon Bottou, and Yoshua Bengio. Reading checks with multilayer graph transformer networks. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 1, pages 151–154. IEEE, 1997.

[33] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[35] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[36] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[37] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[38] Behnaz Nojavanasghari, Deepak Gopinath, Jayanth Koushik, Tadas Baltrušaitis, and Louis-Philippe Morency. Deep Multimodal Fusion for Persuasiveness Prediction. In *ACM International Conference on Multimodal Interaction*, pages 284–288, 2016.

[39] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. 2016.

[40] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[41] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[43] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015.

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[47] Luc Vincent and Pierre Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6):583–598, 1991.

[48] A. Wang, J. Lu, J. Cai, T. J. Cham, and G. Wang. Large-margin multi-modal deep learning for rgb-d object recognition. *IEEE Transactions on Multimedia*, 17(11):1887–1898, Nov 2015.