MANCHESTER 1824
The University of Manchester

Real-Time Object Detection with Deep Learning on an Embedded GPU System

Third Year Individual Project – Final Report

April 2020

Márk Antal Csizmadia

Supervisor: Dr. Hujun Yin

Department of Electrical and Electronic Engineering

Acknowledgments

I wish to express my deepest gratitude to my supervisor, Dr. Hujun Yin, for his valuable guidance and continuous assistance that he provided me with throughout the project. His unwavering support helped me make this project a success.

I would also like to extend my appreciation to the Department of Electrical and Electronic Engineering at the University of Manchester for being open to new ideas and accepting my bespoke project.

Abstract

Artificial intelligence is increasingly present in many industries where intelligent digital systems have the potential to improve production and decision-making through visual perception. Machine learning and deep learning help such intelligent systems learn from data and make educated predictions with artificial neural networks. A special class of artificial neural networks, convolutional neural networks, helped researchers achieve unprecedented results in computer vision tasks, such as object detection. Fuelled by these breakthroughs, deep learning, and convolutional neural networks are taking the manufacturing and production industries by storm. Since these application domains require real-time object detection capability, object detection models are increasingly deployed on embedded artificial intelligence computing platforms.

This project investigates the theoretical and practical aspects of training a convolutional neural network-based object detection model and deploying it on an embedded artificial intelligence computing platform for real-time object detection. First, the project explores the mathematics of deep learning, convolutional neural networks, and DetectNet, the object detection model chosen for the project. Then, the model is trained and validated in the NVIDIA DIGITS deep learning system using an object detection dataset specifically created for the project. Finally, the trained model is deployed for real-time object detection on an NVIDIA Jetson Nano embedded artificial intelligence computing platform, and the platform is used as the vision system of a pick-and-place machine. The performance of the trained model is evaluated using the PASCAL VOC evaluation scheme and metrics. The model achieves a mean Average Precision above 90 %, which shows that it learns to efficiently detect the objects of interest in unseen images and videos at test-time. The deep-learning specialized architecture of the embedded artificial intelligence platform runs the trained model at 10 FPS, which enables the pick-and-place machine to perform outstandingly in real-time sorting exercises.

Contents

List of Figures

List of Tables

Total word count: 11468

1. Introduction

1.1. Background and Motivation

The field of artificial intelligence (AI) is concerned with the creation of intelligent computer-based systems that have the capability of decision-making, visual perception, and speech recognition, among others. These capabilities require computers to have knowledge about the world we live in. There are different approaches to provide computers with this knowledge, including the approach of machine learning. Machine learning enables computers to acquire their knowledge in the form of extracting patterns from raw data such as images. The representation of data plays an important role in machine learning. Intelligent systems aim to learn a simplified representation of the input data that can be used to efficiently map the input to the output. As described in [1], deep learning solves this central problem by "learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones".

The most essential deep learning models are deep neural networks (DNNs). DNNs belong to the wider class of feedforward neural networks (FNNs), which are descendants of artificial neural networks (ANNs). Since DNNs belong to the family of supervised machine learning algorithms, they learn through a process termed training. During training, DNNs are shown true examples of the solution to the given task, and their goal is to learn a mapping between the input data and the output via back-propagating the error between the predicted and the gound-truth outputs. Convolutional neural networks (CNNs) are a specialized class of DNNs for processing data with a two-dimensional grid topology, such as pixels in images. The breakthrough of CNNs was brought about by the appearance of a large-scale database of annotated images called ImageNet [2], and the availability of the high-performance computing hardware called Graphics Processing Units (GPUs). As a result, in 2012, Krizhevsky et al. introduced AlexNet in [3], a large, deep CNN model that used an "efficient GPU implementation of the convolution operation" and won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) by outperforming the state-of-the-art.

AlexNet induced an unprecedented interest in deep learning and popularized the Industry 4.0 applications of deep learning-based computer vision in manufacturing. Industry 4.0 is the next phase of the digitalization of processes and technologies within the manufacturing industry that is driven by the rise in data generation, available computational power, and device connectivity. According to a recent study [4], AI will revolutionize many functions of manufacturing such as quality

where ”vision systems [that] use image-recognition technology [will] identify defects and deviations in product features.” Furthermore, in production “robots enhanced with intelligent image-recognition capabilities will be able to pick up unsorted parts in undefined locations, such as from a bin or a conveyor belt.” Such vision-enabled machines will help “producers (…) boost efficiency, improve flexibility, accelerate processes, and even enable self-optimizing operations”. These use cases are driving the adoption of edge computing for embedded AI applications. In edge computing, computation processes are shifted from data centers and cloud infrastructures to edge and Internet of Things (IoT) devices. On the one hand, edge computing comes with advantages such as reduced data communication with servers and the cloud, which makes edge devices suitable for real-time AI applications. On the other hand, they are constrained in terms of computing resources, throughput, and latency, which makes the realization of such real-time AI applications difficult. As a result, there is an increasing demand for edge computing devices that can efficiently deploy embedded real-time AI applications.

1.2. Aims and Objectives

The project aims to detect select objects in real-time with a CNN-based object detection model on an embedded AI computing platform and adapt the platform to serve as the vision system of a pick-and-place machine. The objectives for achieving this aim are listed below.

- Study and discuss the mathematics of deep learning, CNNs, and object detection.
- Discuss the features of DetectNet, the object detection model selected for the project.
- Create an annotated object detection image dataset of toy cars, dices, batteries, and a few other objects that serve as negative examples.
- Train a pre-trained DetectNet object detection model on the custom dataset in the NVIDIA Deep Learning GPU Training System (DIGITS).
- Conduct a hyperparameter search to find the best performing model.
- Evaluate the object detection performance of the model using the PASCAL VOC object detection evaluation scheme and metrics.
- Deploy the object detection model on an NVIDIA Jetson Nano embedded AI computing platform for real-time object detection.
- Create a pick-and-place machine by connecting the AI computing platform with a robotic arm. Adapt the platform to serve as the vision system of the pick-and-place machine.
- Evaluate the performance of the pick-and-place machine in live sorting tests.

The novelty of the project lies in two of its elements. Firstly, DetectNet is trained on a custom object detection dataset for multi-object detection, the documentation of which is scarcely available to the public. Secondly, the DetecNet-based vision system equips the robotic arm of the pick-and-place machine with the visual perception that could potentially be used in an industrial application similar to the ones mentioned in section 1.1.

1.3. Project Organization

The knowledge necessary for the realization of the project is not part of the syllabus of the BEng Electronic Engineering programme at the University of Manchester. Therefore, the relevant topics were studied through self-directed learning, online courses, and with the aid of the weekly meetings with Dr. Hujun Yin, the supervisor of the project. The certificates of the completed online courses are included in Appendix A: Online Course Certificates. In preparation for the main project work, several practical exercises were completed, which are included in Appendix B: Preparatory Exercises. The final project progress report is included in Appendix C: Progress Report. At the beginning of the project, a risk assessment was conducted, which is included in Appendix D: Risk Assessment. Furthermore, a statement about the effects of the coronavirus (COVID-19) on the project is included in Appendix E: The Effects of Coronavirus (COVID-19) on the Project.

2. Literature Review

2.1. Convolutional Neural Networks

2.1.1. Universal Approximators

DNNs aim to approximate a mapping function as shown in Equation 1 [1].

$$\hat{\boldsymbol{y}} = \hat{f}(\boldsymbol{x}; \boldsymbol{\theta}) \tag{1}$$

where $\hat{f}$ is the approximated mapping function, $\boldsymbol{x}$ is the input data represented as a vector of features, $\hat{\boldsymbol{y}}$ is the vector of predicted labels, and $\boldsymbol{\theta}$ are the parameters of the mapping.

The goal is to adjust the parameters $\boldsymbol{\theta}$ such that $\hat{f}$ closely approximates the true mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ for every $\boldsymbol{x}$**.** The input $\boldsymbol{x}$ is mapped to the output $\hat{\boldsymbol{y}}$ via a sequence of linear and non-linear transformations that are characterized by the parameters $\boldsymbol{\theta}$. The transformations are abstracted into layers of neuron-like computational nodes that constitute a network. Figure 1 shows a DNN with two hidden layers where x_1, x_2, and x_3 represent $\boldsymbol{x}$.

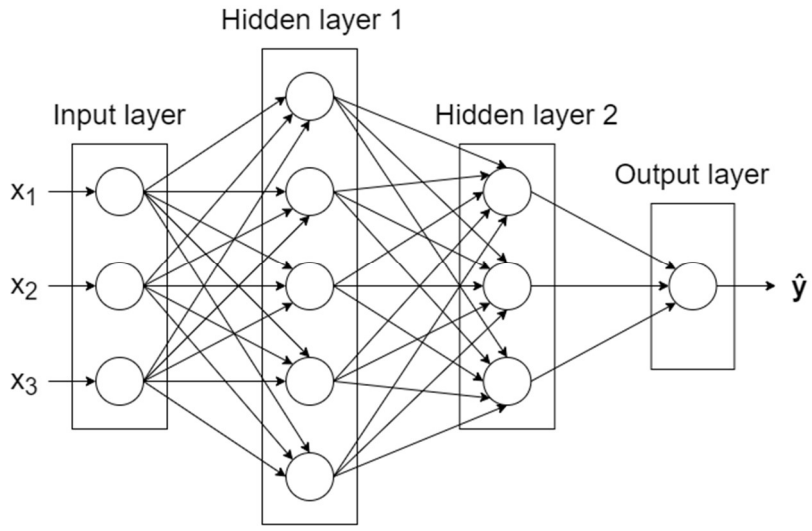


Figure 1: A deep neural network with two hidden layers.

In Figure 1, the neurons and the layers are denoted by the circles and vertical rectangles, respectively. The signal propagates forward and backward across the network via the connections between neurons and the signal strength is determined by the weight of these connections. In Equation 1 in terms of CNNs, $\boldsymbol{x}$ contains the image pixels, $\boldsymbol{y}$ contains the ground-truth image or object class labels, and $\hat{\boldsymbol{y}}$ contains the predicted image or object class labels. Therefore, CNNs aim to learn a representation of the input image data that is then mapped to the output image or object class labels by a classifier in the last layer. The mapping is defined by fully connected, convolutional,

pooling, and normalization layers, among others. As proved in [5], DNNs and CNNs with at least one hidden layer are universal approximators of any continuous function, which means that CNNs can theoretically learn to solve any image classification or object detection task.

2.1.2. Forward-Propagation and Fully-Connected Layers

Each neuron in a layer is characterized by a weight vector $\boldsymbol{w}$, a constant b, and an activation function, g. Neurons are grouped into layers and their weight vectors and constants constitute matrices that define a linear transformation followed by a non-linearity. Equation 2 [6] shows the linear transformation and the element-wise non-linearity in layer l with n_l neurons.

$$\boldsymbol{x}^{[l]} = g^{[l]}\left(W^{[l]}\boldsymbol{x}^{[l-1]} + \boldsymbol{b}^{[l]}\right) \tag{2}$$

where l and $l-1$ are layer indices, $\boldsymbol{x}^{[l-1]}$ ($\in \mathbb{R}^{\boldsymbol{n_{l-1}\times 1}}$) is a vector of features before passing through layer l, $g^{[l]}$ is the activation function that defines the element-wise non-linearity in layer l, $\boldsymbol{x}^{[l]}$ ($\in \mathbb{R}^{\boldsymbol{n_l\times 1}}$) is the resultant feature vector of the transformations, $W^{[l]}$ ($\in \mathbb{R}^{\boldsymbol{n_l\times n_{l-1}}}$) is the weight matrix in layer l, and $\boldsymbol{b}^{[l]}$ ($\in \mathbb{R}^{\boldsymbol{n_l\times 1}}$) is the constant vector in layer l.

The result of the transformation in layer l is fed into layer $l+1$, and so on. This iterative process is referred to as forward-propagation. In CNNs, fully-connected layers, which are usually found near the end of the network, perform the exact same transformations on flattened feature vectors as shown in Equation 2. The convolutional, pooling, and normalization layers, which are discussed later in the report, perform forward propagation in a slightly different, but similar manner.

2.1.3. Convolutional and Pooling Layers

Convolutional layers utilize the 2-dimensional discrete convolution operation shown in Equation 3 [1]. More precisely, Equation 3 shows the cross-correlation operation that is equivalent to convolution without kernel flipping.

$$O(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{3}$$

where O is the output feature map, I is the input feature map, K is the convolution kernel, and m, n, i, and j are indices for the convolution kernel sliding procedure across the feature maps.

Convolutional layers operate on data volumes rather than on flattened feature vectors as fully-connected layers do. However, 2-dimensional convolution can be formulated in terms of matrix multiplications similarly to Equation 2. The operation of convolutional layers is defined by the following parameters: the number of input feature maps similar to I (c_1), the height and width

of the input feature maps (h, w), the number of convolution kernels similar to K (c_2), the kernel size along the vertical and horizontal dimensions (f_1, f_2), the convolution stride size along the vertical and horizontal dimensions (s_1, s_2), and the amount of zero-padding along the vertical and horizontal dimensions (p_1, p_2). In forward propagation, the input data volume is of size $h_1 \times w_1 \times c_1$, the convolutional layer has c_2 convolution kernels, each of which has c_1 weight matrices of size $f_1 \times f_2$, and the output data volume is of size $[(h - f_1 + 2p_1)/s_1 + 1] \times [(w - f_2 + 2p_2)/s_2 + 1] \times c_2$. The convolutional layer has $f_1 \times f_2 \times c_1 \times c_2$ weights and c_2 constants, all of which are learnable parameters. Figure 2 shows forward propagation in a convolutional layer.

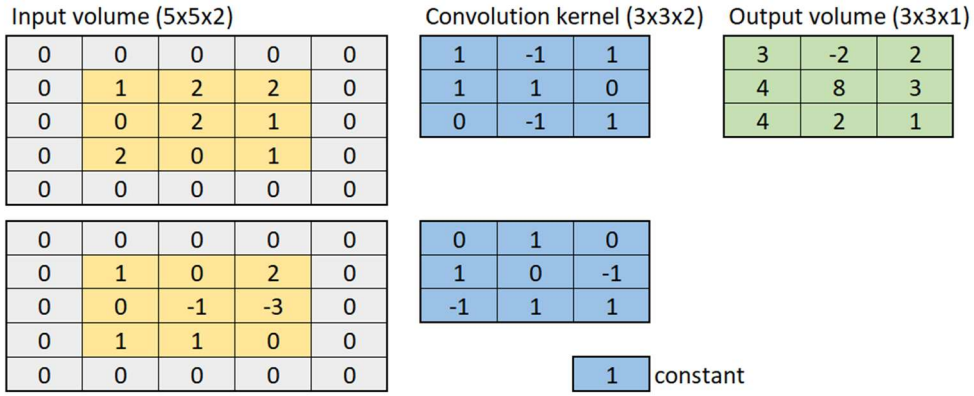


Figure 2: Forward propagation in a convolutional layer.

In Figure 2, $c_1 = 2$, $c_2 = 1$, $h = 3$, $w = 3$, $f_1 = f_2 = f = 3$, $s_1 = s_2 = s = 1$, and $p_1 = p_2 = p = 1$. The input volume is of size $5 \times 5 \times 2$ after zero-padding, there is one convolution kernel that has 2 weight matrices of size 3×3, and the resultant output volume is of size $3 \times 3 \times 1$. The upper weight matrix is convolved with the upper input feature map, and the lower weight matrix is convolved with the lower input feature map. The results are summed and the convolution kernel constant is added. For instance, for $i = 1$ and $j = 1$, Equation 3 gives the value of 8 in the output feature map as shown in Appendix F: Example of Forward-Propagation in Convolutional Layers. The convolutional layer in Figure 2 has 19 learnable parameters. Since the input and the output feature maps have the same size $(h = 3, w = 3)$, this type of convolution is called same convolution. Conversely, valid convolution produces different output feature map sizes.

Another building block of CNNs is the pooling layer that summarizes a subregion in the input feature map and outputs the summary statistic in the output feature map. Pooling helps CNNs reduce the feature map sizes and build up invariance against object translations in images. The pooling operation is commonly either average or maximum pooling. Pooling layers have no learnable parameters. Using the previously established notation, the input data volume to a pooling

layer is of size $h_1 \times w_1 \times c_1$, and the output data volume is of size $[(h - f_1)/s_1 + 1] \times [(w - f_2)/s_2 + 1] \times c_2$. Figure 3 shows the forward-propagation in a pooling layer with $c_1 = 2$, $c_2 = 2, h = 4, w = 4, f_1 = f_2 = f = 2, s_1 = s_2 = s = 2$.

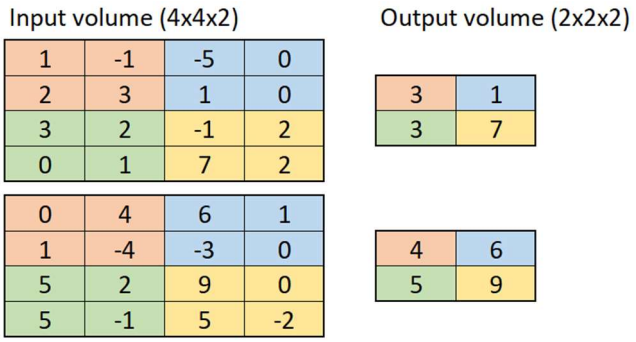


Figure 3: Forward propagation in a pooling layer.

Figure 4 illustrates the forward-propagation in some of the convolutional and pooling layers in AlexNet [3] where the left-most data volume of size $227 \times 227 \times 3$ is the RGB input image.

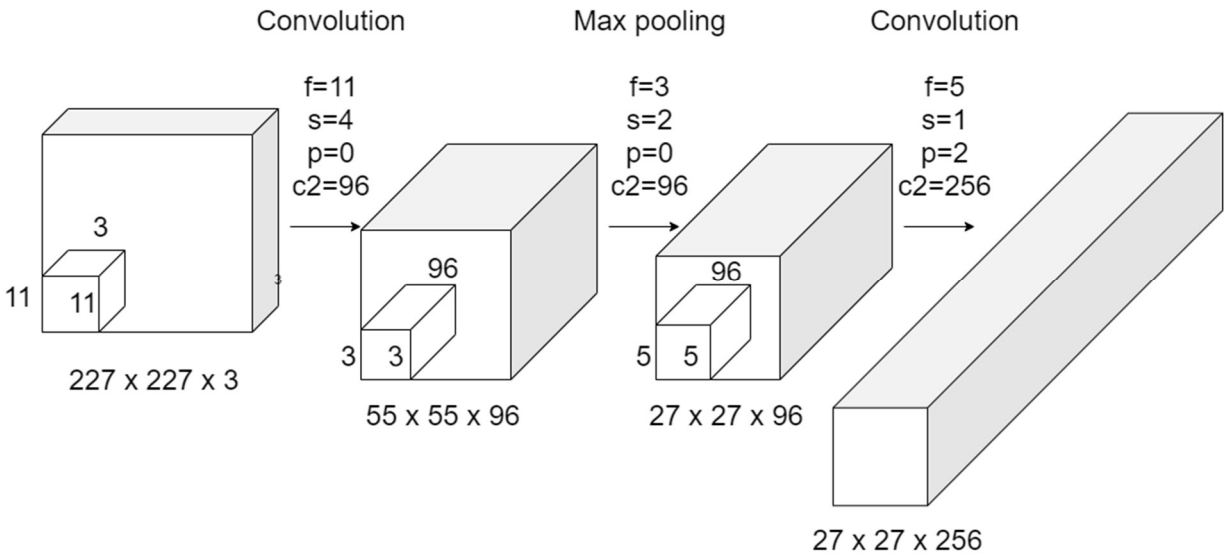


Figure 4: Convolution and pooling in AlexNet.

As described in [1], convolutional layers enable CNNs to efficiently process image data through sparse interactions, parameter sharing, and equivariant representations. Unlike in fully-connected layers, in convolutional layers not each entry in the input feature maps interacts with each entry in the output feature maps. As a result of these sparse interactions, the same set of parameters is used to convolve patches of the input feature maps with the convolution kernels. This parameter sharing phenomenon results in the convolutional layer being equivariant to translation in incoming feature maps. That is, the convolution kernel can extract the same information across the input feature maps regardless of its location. Parameter sharing and sparse interactions enable

deeper layers in the network to interact with greater patches of an input image while learning more abstract identities regardless of their original location. Consequently, the depth of CNNs is proportional to their representational power. Following this logic, 1x1 convolutions are proposed in [7] to increase the depth of the network, introduce extra non-linearity, and reduce the number of feature maps of the next convolutional layer. As a result, the representational power of the network is increased, and the computational cost of subsequent convolutional layers is reduced. However, deeper models are more prone to over-fitting and vanishing gradients. In [8], Szegedy et al. proposed a new deep CNN architecture by the name Inception that aimed to deepen networks while avoiding over-fitting. GoogLeNet, a network model based on the Inception v1 architecture shown in Figure 5 [8], outperformed the state of the art at the ILSVRC-2014. The idea behind the Inception module is that information in images has a sparse structure since the salient parts of the image vary in size and location due to occlusion or distance from the camera. This sparsity can be exploited by clustering correlated outputs as argued in [9]. This follows the Hebbian learning rule [10] that states that the synaptic strength of neurons that activate together should increase. Following this logic, the Inception module learns the sparse representation by processing the visual and spatial information at various scales. This achieved by a mixture of 1x1, 3x3, and 5x5 convolutional layers that extract and aggregate different features, and feed them to the next layer. Variants of the Inception v1 module aim to better preserve information in the deep architecture and factorize the convolution filter banks for increased computational efficiency.

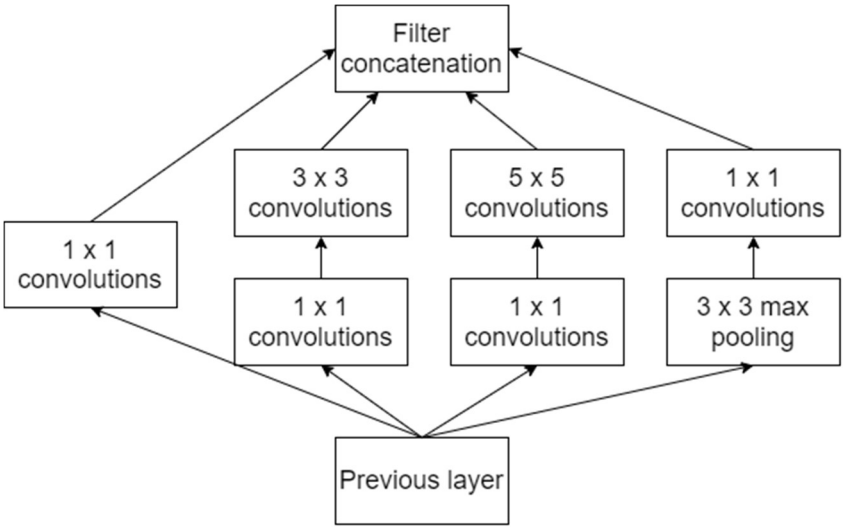


Figure 5: Inception v1 module in GoogLeNet.

2.1.4. Activation Functions

Activation functions introduce non-linearity that allows CNNs to learn highly non-linear representations of the input data. Examples of activation functions include the sigmoid function and the Rectified Linear Unit (ReLU) function. The sigmoid function is discussed later in the report. The ReLU function is defined in Equation 4 [11].

$$g^{[l]}\left(\boldsymbol{x}^{[l-1]}, W^{[l]}, \boldsymbol{b}^{[l]}\right) = \max\left(0, W^{[l]}\boldsymbol{x}^{[l-1]} + \boldsymbol{b}^{[l]}\right) \quad (4)$$

where the g is the ReLU function and the rest of the notation corresponds to that of in Equation 2.

It was shown in [11] that the signal traveling through the ReLU rectifier amplifies the response of neurons that have activation values above zero or some other small value, and eliminates others. This helps the network avoid vanishing gradients and learn the sparse data representation that was discussed in the case of the Inception v1 module in section 2.1.3.

2.1.5. Classifier, Loss, and Cost Function

The last layer of CNNs is a classifier that approximates the probability distribution of the input data with maximum likelihood estimation. In the case of multi-class classification, the Multinoulli output distribution is approximated with the softmax classifier. The softmax classifier is mentioned for completeness but it is not used in the project. In the case of binary classification, the Bernoulli output distribution is approximated with the sigmoid classifier shown in Equation 5 [6].

$$\hat{y} = \hat{f}(\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{1 + e^{-\left(W^{[\mathrm{L}]}\boldsymbol{x}^{[\mathrm{L}-1]} + \boldsymbol{b}^{[\mathrm{L}]}\right)}} \quad (5)$$

where $\hat{\boldsymbol{y}}$ is the vector of predicted labels, $\hat{f}(\boldsymbol{x}; \boldsymbol{\theta})$ is the approximated mapping from Equation 1, $W^{[\mathrm{L}]}$ and $\boldsymbol{b}^{[\mathrm{L}]}$ are the parameters of the sigmoid classifier, and $\boldsymbol{x}^{[\mathrm{L}-1]}$ is the representation vector learned by the network.

The measure of dissimilarity, or loss, between the model's estimated probability distribution and the true data distribution, is expressed as the negative log-likelihood, or equivalently, the cross-entropy. The per-example sigmoid cross-entropy loss of binary classification is shown in Equation 6 [6].

$$L\left(\hat{f}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right) = -\left(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right) \quad (6)$$

where $L\left(\hat{f}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$ is the per-example loss between the predicted label $\hat{f}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$ and the ground-truth label $y^{(i)}$, $\boldsymbol{\theta}$ denotes the parameters in each layer of the network, and $\hat{y}^{(i)}$ is the

prediction for the i^{th}observation that is identical to $\hat{f}(\boldsymbol{x}^{(i)};\boldsymbol{\theta})$.

Using Equation 6, the cost function of a CNN with a sigmoid classifier is defined in Equation 7 [1]. The cost is the sum of the per-example losses normalized by the number of observations.

$$J(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} L\left(\hat{f}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)}\right) \tag{7}$$

where $J(\boldsymbol{\theta})$ is the cost, M is the number of observations, and $L\left(\hat{f}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)}\right)$ is the per-example loss.

In object detection, the cost function incorporates the object label loss as well as the object localization loss. The object label loss is commonly one of the sigmoid and the softmax loss. However, the form of object localization loss often varies across different architectures.

2.1.6. The Back-Propagation Algorithm and Gradient-Based Learning

In [12], Rumelhart et al. introduced a new learning procedure in neural networks that they called back-propagation. The procedure of the back-propagation of errors "repeatedly adjusts the weights of the connections in the network to minimize a measure of the difference between the actual output vector of the net and the desired output vector". In [13], LeCun et al. demonstrated the first commercial use of the back-propagation algorithm in CNNs by recognizing hand-written digits on checks. Paraphrasing the definition of the back-propagation algorithm, the loss is back-propagated across the network and gradient-based optimization techniques are used to adjust the learnable parameters, or neuron connection weights, such that the loss is minimized. In line with the definition, the parameter gradients in an arbitrary layer are derived as shown in Equation 8 [1]. The full mathematical derivation of the back-propagation algorithm in fully-connected, convolutional, pooling, and normalization layers is beyond the scope of the report.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \nabla_{\boldsymbol{\theta}} L\left(\hat{f}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)}\right) \tag{8}$$

where $\nabla_{\boldsymbol{\theta}}$ is the gradient operator, $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the vector of partial derivatives of the cost function $J(\boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{\theta}$, and $\nabla_{\boldsymbol{\theta}} L\left(\hat{f}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)}\right)$ is the vector of partial derivatives of the per-example loss with respect to the parameters $\boldsymbol{\theta}$.

2.1.7. Regularization and Data Augmentation

Regularization and data augmentation are techniques against over-fitting. Over-fitting refers to the phenomenon when a model learns an overly specific representation of the training data and it is then unable to generalize to previously unseen observations. This manifests itself as high training and low test performance metrics. The classical solution to avoid over-fitting is the inclusion of a weight norm penalization term in the loss function such as the L^1 or the L^2 norm. Norm penalization is mentioned here, but it is not used in the project. Introduced in [14], dropout is a different type of regularization method. Dropout layers disregard select connections from previous layers with a probability p called the dropout ratio. By disregarding some connections, the model learns a more relaxed representation of the data and can better generalize to previously unseen observations. Besides these regularization techniques, data augmentation offers an alternative option to avoid over-fitting. Data augmentation is a technique to expand the size of the training dataset in an attempt to show more examples to the model during training. Showing more training examples to the model results in better model generalization. In the context of image classification and object detection, the training dataset is augmented with images that are, for instance, flipped, cropped, scaled, or rotated. There exist offline and online augmentation. In offline augmentation, the augmented data is stored locally before feeding it to the network at training time. In online augmentation, the augmentation is done on the fly during training, and data is not stored locally.

2.1.8. Parameter Optimization Algorithms

Parameter optimization algorithms use the back-propagated error gradients to adjust the learnable parameters in layers across the network. The back-propagated gradient is an expectation, which can be estimated using a small set of observations, called a mini-batch. Averaging the gradient over mini-batches accelerates training. The estimated mini-batch gradient is shown in Equation 9 [1].

$$\boldsymbol{g_t} = \frac{1}{M'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{M'} L\left(\hat{f}\left(\boldsymbol{x^{(i)}}; \boldsymbol{\theta_{t-1}}\right), y^{(i)}\right) \tag{9}$$

where t is the iteration number, $\boldsymbol{g_t}$ is the estimated gradient on iteration $\boldsymbol{t}$, $\boldsymbol{\theta_{t-1}}$ is the parameter vector on iteration $\boldsymbol{t-1}$, M' is the mini-batch size and the rest of the notation is as in Equation 8.

LeCun et al. proposed in [15] that the stochastic gradient descent (SGD) algorithm combined with the back-propagation algorithm was an effective method for training CNNs. The SGD parameter

update rule is shown in Equation 10 [15].

$$\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \alpha_t \boldsymbol{g_t} \tag{10}$$

where t is the iteration number, $\boldsymbol{g_{t-1}}$ is the estimated gradient of the cost function with respect to the parameters $\boldsymbol{\theta_{t-1}}$ and α_t is the learning rate. It is common to apply learning rate decay policies, such as exponential learning rate decay, to ensure the convergence of SDG, which is why α_t denotes the learning rate on iteration t.

Lately, the Adam, or Adaptive Moment Estimation optimizer [16] is the preferred option due to its fast and robust convergence, and computational efficiency. The algorithm computes the first-order moment estimate, or mean, and the second-order moment estimate, or uncentered variance, using the gradient estimate from Equation 9. The first-order and the second-order moment estimates are shown in Equation 11 [16] and Equation 12 [16], respectively.

$$\boldsymbol{m_t} \leftarrow \beta_1 \boldsymbol{m_{t-1}} + (1 - \beta_1)\boldsymbol{g_t} \tag{11}$$

$$\boldsymbol{v_t} \leftarrow \beta_2 \boldsymbol{v_{t-1}} + (1 - \beta_2)\boldsymbol{g_t^2} \tag{12}$$

where $\boldsymbol{v_t}$ and $\boldsymbol{m_t}$ are the exponential moving average estimates of the first-order and second-order moments of the gradient at timestep $\boldsymbol{t}$, and β_1 and β_2 are the exponential decay rate of the moving averages. The authors propose the default values of 0.9 for β_1, and 0.999 for β_2.

Since the moving averages are initialized to zero, the estimates are biased towards zero during the initial timesteps. The bias-corrected estimates $\widehat{\boldsymbol{m_t}}$ and $\widehat{\boldsymbol{v_t}}$ are computed as shown in Equation 13 [16] and Equation 14 [16], respectively.

$$\widehat{\boldsymbol{m_t}} \leftarrow \frac{\boldsymbol{m_t}}{1 - \beta_1^t} \tag{13}$$

$$\widehat{\boldsymbol{v_t}} \leftarrow \frac{\boldsymbol{v_t}}{1 - \beta_2^t} \tag{14}$$

The bias-corrected estimates are used to update the parameters as shown in Equation 15 [16].

$$\boldsymbol{\theta_t} \leftarrow \boldsymbol{\theta_{t-1}} - \alpha_t \frac{\widehat{\boldsymbol{m_t}}}{\sqrt{\widehat{\boldsymbol{v_t}}} + \epsilon} \tag{15}$$

where α_t is the learning rate at timestep t, and the constant ϵ is set to 10^{-8} for numerical stability.

2.1.9. Normalization Strategies

The Local Response Normalization (LRN) is a normalization technique that prioritizes neurons that activate stronger in the same spatial location across adjacent channels. In neuroscience, this phenomenon is referred to as lateral inhibition. LRN accelerates model training and may be applied within or across different feature maps. The two versions of LRN are called intra-channel and inter-channel LRN. Equation 16 [3] shows a type of inter-channel LRN used in AlexNet [3] and GoogLeNet [8].

$$b_{x,y}^{i} = a_{x,y}^{i} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^{j})^2 \right)^{\beta} \tag{16}$$

where $a_{x,y}^{i}$ denotes the neuron activation (after applying the non-linearity) at the spatial position (x, y) in the i^{th} feature map, $b_{x,y}^{i}$ denotes the response-normalized activity at the same position in the same feature map, n denotes the number of adjacent feature maps included in the normalization operation, N is the total number of feature maps in the layer, and k, α, and β are hyper-parameters. The sum runs over n adjacent feature maps and computes the amount by which neuron activities are to be normalized across the different feature maps. The authors of AlexNet propose the following hyperparameter values: $k = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$.

2.1.10. Weight Initialization Techniques

Proper initialization of the learnable parameters ensures that the signal propagating through deep networks assists fast learning and convergence. These techniques tackle exploding and vanishing gradients and break the symmetry of signal propagation in the network.

Introduced in [17], the Xavier initialization maintains the activation variances and back-propagated gradients variance across the layers in the network during training. It is achieved by initializing the weights in layers using the uniform distribution shown in Equation 17 [17].

$$W^{i} \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \right) \tag{17}$$

where W^{i} is the weight matrix in layer i, U is the uniform distribution, n_i is the fan-in, or the number of neurons in the input layer i, and n_{i+1} is the fan-out, or the number of neurons in the output layer $i + 1$. The Xavier initialization results in uniformly distributed weights with a variance of $Var[W^{i}] = 2/(n_i + n_{i+1})$. Commonly, the constants are initialized to a small number.

2.1.11. Transfer Learning

Training CNNs from scratch is computationally expensive and time-consuming as it requires large labeled datasets and high-performance computing hardware. An alternative is to train CNNs on very large existing datasets, such as ImageNet [2] or the PASCAL VOC dataset [18], and then use the network either as a parameter initialization for fine-tuning or as a static convolutional feature extractor for custom tasks [19]. This is supported by the observations presented in [20] that show that earlier convolutional layers come to represent generic features such as edge or color blob detectors that are useful for custom tasks. On the contrary, the last convolutional layers learn more specific features of the original training data. Therefore, replacing the classifier layer and continuing the training via back-propagation will help fine-tune the weights in the entire network for the new custom task.

2.2. Object Detection

2.2.1. Object Detection Models

Traditional object detection algorithms extract hand-crafted image features at every location and scale, such as the Histogram of Oriented Gradients (HOG) [21], and the Scale-Invariant Feature Transform (SIFT) [22]. Then, objects are modeled allowing for geometrically constrained deformation using, for instance, the Deformable Part-Based Models [23]. Lastly, a classifier, such as a linear Support Vector Machine (SVM) is used for classification.

Deep learning and CNN-based object detection models outperform the traditional algorithms since they rely on the convolutional features learned by the network. For instance, OverFeat [24] used a multiscale sliding window procedure with CNNs for the localization and classification of a single object. The algorithm performs better than the traditional algorithms, but its disjoint architecture makes it slow for real-time detection. In 2014, Girshick et al. introduced the R-CNN, or regions with CNN features in [25]. The algorithm generates class-agnostic region proposals via selective search, extracts convolutional features with a CNN, and applies a binary linear SVM for classification. R-CNNs achieved high accuracy in object detection benchmarks but were slow for real-time object detection since the multi-stage architecture incurred long processing times. Its variants include Fast R-CNNs and Faster R-CNNs, which aim to unify and speed up the disjoint system. In 2015, the You Only Look Once (YOLO) algorithm was introduced in [26]. YOLO's network architecture was inspired by GoogLeNet, and it approaches object detection as a regression

problem. The algorithm imposes a grid onto the images and uses a CNN to predict the bounding box coordinates and the class probabilities for each grid cell. YOLO is significantly faster than R-CNNs since it has a unified structure and it predicts the location and the class probabilities in one evaluation without region proposals. However, it has higher localization errors than R-CNNs.

2.2.2. Performance Evaluation Metrics

In the project, the pre-2010 PASCAL VOC [18] object detection benchmark challenge evaluation scheme and metrics are used for evaluating models. In this scheme, the ultimate performance metric is the mean Average Precision (mAP), which is derived using the ranked retrieval results of the models. The detections are ranked based on the model's detection confidence score and further classified into positive or negative types based on the Intersection over Union (IoU). The IoU is the ratio of the area intersection and the area union of a predicted bounding box and its corresponding ground truth bounding box. The IoU is defined in Equation 18.

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{18}$$

where B_p and B_{gt} are the ground-truth and the predicted rectangular bounding boxes.

The PASCAL VOC scheme uses only a single IoU threshold of 0.5. While true positive detections have greater, false-positive detections have lower IoU with their ground truth bounding box than the IoU threshold. A detection also counts as false positive if it has IoU with a ground-truth bounding box other than its own. False-negative detections mean that the model failed to produce a bounding box for a ground-truth object. In object detection, true negatives are not used as there is an infinite number of bounding boxes that can correctly not be detected. Before conducting the IoU analysis, the detections in an image are ranked, or ordered, based on the model's confidence score. If the confidence score is below the confidence threshold, the detection is either true negative or false negative depending on the subsequent IoU analysis. If the confidence score is above the confidence threshold, the detection is either true positive or false positive depending on the subsequent IoU analysis. Using the detection types, the model's precision and recall can be computed as shown in Equation 19 and Equation 20, respectively.

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{19}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{20}$$

While the precision of the model defines its ability to detect only the relevant objects, the

recall of the model defines its ability to detect all of the relevant objects. Having computed the precision and the recall, the precision at each recall level r is interpolated to the maximum precision found at any recall value $\tilde{r}$ that exceeds r. This rule is given mathematically in Equation 21.

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r}\geq r} p(\tilde{r}) \tag{21}$$

The Average Precision (AP) of an object class is the average of the interpolated precision values, p_{interp}, sampled at eleven distinct recall values as shown in Equation 22.

$$AP = \frac{1}{11} \sum_{r\in\{0,0.1,\ldots,1\}} p_{interp}(r) \tag{22}$$

Finally, the mAP of the object detection model is the mean of the object class APs.

2.3. Graphical Processing Units and Edge Computing

GPUs have highly parallel architecture that enables them to break down the matrix multiplications of the forward-propagation and the back-propagation into separate tasks that can be solved simultaneously. As a result, the appearance of General-Purpose GPU (GPGPU) programming models, and specialized GPU-accelerated deep learning libraries helped reduce CNN training times, which brought about the breakthroughs of deep learning and CNNs. GPUs are increasingly used in embedded AI computing platforms. Edge computing refers to the phenomenon that computation is performed on the edge and IoT devices instead of powerful servers or the cloud. One of its advantages is that sensitive data can be stored and processed locally instead of sending it to the cloud or external servers. The reduced data communication and round-trip times reduce the latency and speed up response time. As a result, devices at the edge can operate in premises with limited or unreliable connectivity. However, edge computing devices are resource-constrained in terms of memory, throughput, and power. Therefore, there is a growing need for novel techniques to efficiently deploy deep learning models on them. In [27] such techniques include the optimization of models for resource-constrained environments, the compression and reduction of the scale of models, and the distribution of data and models across multiple devices. Alternatively, the hardware may be optimized, which increasingly includes the integration of GPUs into embedded AI computing platforms.

3. Methodology

3.1. The Object Detection Model

The DetectNet object detection model was selected for the project. The model description file for training is included in Appendix G: DetectNet Description File for Training. The visualization of the model architecture is included in Appendix H: DetecNet Architecture Visualization. In this section, the features of the model are summarized based on section 2 with the aid of [28]. At the end of the summary, supporting evidence is shown for choosing DetectNet over other object detection models.

Overall network architecture. DetectNet includes a fully-convolutional network (FCN) variant of GoogLeNet, discussed in section 2.1.3, without the original data input layers, final global average pooling layer, and output layers. As a result, DetectNet leverages the features of the Inception v1 module discussed in section 2.1.3. While these layers are removed to enable transfer learning, the absence of fully-connected layers allows the network to accept images of varying sizes. DetectNet uses the FCN as a sliding window with a receptive field (i.e.: the image patch size in pixels that impact the output of the last convolution) of 555 x 555 pixels and a stride of 16 pixels. Experimentally, this results in DetectNet being sensitive to objects in the range of 40x40 and 400x400 pixels. The algorithm uses a data representation in which a grid is imposed onto images similarly to YOLO discussed in section 2.2.1. The training objective of the algorithm is to predict whether an object is present in each grid cell and to predict the bounding box corner coordinates of that object relative to the center of each grid cell. Overall, the FCN variant of GoogLeNet within DetectNet has 5 980 727 learnable parameters.

Convolutional and pooling layers. Since DetectNet includes an FCN variant of GoogLeNet, the layers are predominantly convolutional. For instance, the "conv2/3x3" convolutional layer is parameterized as $c_1 = 64,\ c_2 = 192,\ f_1 = f_2 = f = 3,\ s_1 = s_2 = s = 1$, and $p_1 = p_2 = p = 1$ using the notation in section 2.1.3. The model uses maximum pooling layers across the entire network. An example is the “pool2/3x3_s2” pooling layer that is parameterized as $f_1 = f_2 = f = 3$, $s_1 = s_2 = s = 2,\ c_1 = 192,\ c_2 = 192$ using the notation in section 2.1.3.

Activation functions. The nonlinearities are exclusively implemented with the ReLU activation function discussed in section 0. An example is the “conv2/relu_3x3” activation layer.

Classifier and bounding boxes. The final softmax classifier and the auxiliary softmax classifier mid-way through GoogLeNet are removed. Instead, in the “coverage/sig” layer, a logistic sigmoid

classifier predicts the object coverage per grid cell, that is, whether an object is present in each grid cell. The bounding box corner coordinates relative to the center of each grid square are predicted by a bounding box regressor in the convolutional layer called “bbox/regressor”. Then, clustering layers cluster the bounding boxes proposed by the regressor and produce the final list of predicted bounding boxes per object class. The clustering is done by the “cluster_gt” and the “cluster” layers during training and inference, respectively.

***Loss and cost function*.** The final cost function is the linear combination of the object coverage loss and the bounding box coordinates loss. The “coverage_loss”, shown in Equation 23 [28], is the normalized sum of the Euclidean distances (L2 loss) between the true and the predicted object coverage across all the grid squares in a mini-batch.

$$L_{coverage} = \frac{1}{2M'} \sum_{i=1}^{M'} \left| coverage_i^t - coverage_i^p \right|^2 \tag{23}$$

where M' is the mini-batch size, $coverage_i^t$ is the true object coverage, and $coverage_i^p$ is the predicted object coverage.

The “bbox_loss”, shown in Equation 24 [28], is the normalized sum of the mean absolute differences (L1 loss) between the true and the predicted bounding box corners in a mini-batch.

$$L_{bounding\ box} = \frac{1}{2M'} \sum_{i=1}^{M'} \left[\left| x_1^t - x_1^p \right| + \left| y_1^t - y_1^p \right| + \left| x_2^t - x_2^p \right| + \left| y_2^t - y_2^p \right| \right] \tag{24}$$

where M' is the mini-batch size, (x_1^t, y_1^t) and (x_2^t, y_2^t) are the coordinates of the upper-left and lower-right corners of the ground-truth bounding box, and (x_1^p, y_1^p) and (x_2^p, y_2^p) are the coordinates of the upper-left and lower-right corners of the predicted bounding box.

Regularization and data augmentation. Regularization is achieved by dropout layers and online data augmentation, both of which are discussed in section 2.1.7. An example of dropout is the "pool5/drop_s1" dropout layer that has a dropout ratio of 40 %. During, training, the “train_transform” layer applies online data augmentation, the parameters of which are defined in “detectnet_augmentation_param”. The online augmentation techniques of DetectNet include, for instance, image scaling between 80 – 120 % with a probability of 40 %, and image desaturation.

Parameter optimization algorithms. In the project, the Adam optimizer is used to optimize the learnable parameter due to its superior convergence properties discussed in section 2.1.8. The parameters of Adam are set to $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-6}$. The learning rate of the optimizer is discussed in section 3.4.2.

Normalization strategies. DetectNet uses LRN, which is discussed in section 0. An example is the “conv2/norm2” LRN layer with parameters $k = 0$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$.

Weight initialization techniques. The weights are exclusively initialized using the Xavier initialization discussed in section 2.1.10. An example is the "conv2/3x3_reduce" convolutional layer, the weights of which are drawn from a uniform distribution with a variance of 0.01. The constants of the layer are initialized to 0.2.

Transfer learning. In the project, pre-trained parameters of GoogLeNet were used for fine-tuning the model on a custom object detection task. The parameters of the ImageNet trained GoogLeNet were kindly made available by the Berkeley Vision and Learning Center (BVLC) [29].

Choosing DetectNet over other models. Since DetectNet is a one-stage object detector algorithm similar to YOLO, it is faster than R-CNNs. Analogously to YOLO, DetectNet approaches object detection as a regression problem while relying on the CNN features learned by the network. However, since DetectNet incorporates an FCN variant of GoogLeNet, it does not have as high localization errors as YOLO. As a result, DetectNet is suitable for accurate, real-time object detection, which is the aim of the project.

3.2. Experimental Setup and Hardware

The trained object detection models were deployed for real-time object detection on the NVIDIA Jetson Nano embedded AI and edge computing platform, which is shown in Figure 6.

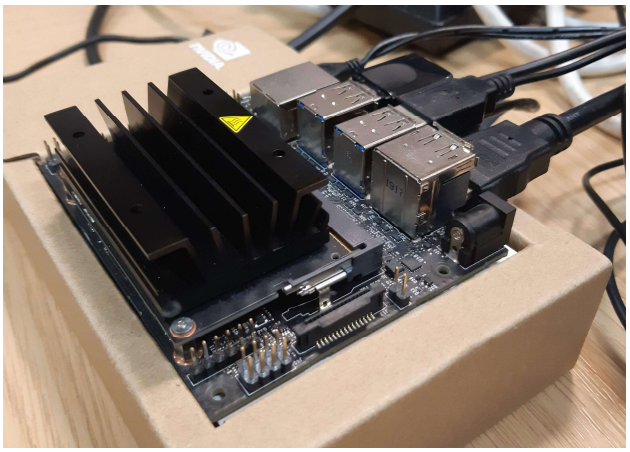


Figure 6: The NVIDIA Jetson Nano embedded AI and edge computing platform.

Outlined in [30], its integrated 128-core Maxwell GPU, which has a compute capability of 5.3 and a memory of 4 GB, enables the real-time performance of AI applications, the principles of which are discussed in section 2.3. The Jetson Nano captures frames and videos via a USB connected camera of 1280 x 720 pixels resolution. The platform runs the Ubuntu 18.04 aarch64 Linux-

distribution operating system. To program the device, a keyboard, a mouse, and a monitor were also connected to it. The models were trained and validated in the NVIDIA Deep Learning GPU Training System (DIGITS) [31], which is a compact web app for managing end-to-end deep learning workflows. DIGITS provides tools for data management, the design and training of deep learning models, and real-time performance evaluation and visualization. In the DIGITS workflow, the host machine runs a DIGITS server that handles data, model training, and validation, and the trained models are transferred onto the Jetson Nano for real-time inferencing. The host machine in the project was a Lenovo Ideapad Y700 laptop, which ran the Ubuntu 16.04 LTS operating system and had a built-in NVIDIA GeForce GTX 960M CUDA-enabled GPU with a GPU Compute Capability of 5.0 and 4.0 GB memory. The DetectNet model architecture was defined in the NVCaffe [32] deep learning framework format within DIGITS, the description file of which is discussed in section 3.1. NVCaffe integrates with the NVIDIA CUDA [33] GPGPU programming model and the NVIDIA CUDA Deep Neural Network (cuDNN) [34] GPU-accelerated deep learning library. Using the CUDA GPGPU programming model and the cuDNN deep learning library, the object detection model could be efficiently mapped to the GPUs of the Jetson Nano and the host machine as described in section 2.3. Furthermore, the TensorRT SDK [35] helped run optimized versions of the trained models with low latency and high-throughput on the Jetson Nano. The majority of programming, data analysis, and data visualization was done in Python 2.7 and 3.7 [36].

3.3. The Object Detection Dataset

The objects selected for the object detection task included six-sided boardgame dices, AAA, AA, and 9 V batteries, and toy cars. Negative examples in the dataset, or “other objects”, included tea candles, highlighters, and spoons. An image dataset of these objects was built through different means that included scraping images off the Internet with the Bing Image Search API [37], remixing existing datasets [38] from the public domain, extracting video frames from videos downloaded from YouTube, and manually taking photographs. The images were resized into 640 x 640 pixels in line with the 16-pixel stride of DetectNet mentioned in section 3.1, and their aspect ratio was kept using zero-padding. Care was taken for images to show objects between 40 x 40 and 400 x 400 pixels due to the sensitivity of DetectNet mentioned in section 3.1. The resized images were annotated using an annotation tool [39] published in the public domain. The annotation of the full dataset took around 5 weeks. This specific labeling tool was selected since it produces annotation data in the KITTI format [31] that is compatible with the DIGITS deep learning workflow. The KITTI format

defines a set of parameters for each object in each image that includes *type*, *truncated*, *occluded*, *alpha*, *bbox*, *dimensions*, *location*, *rotation_y*, and *score*. In the project, the relevant parameters are *type*, *bbox, rotation_y* and *score.* The *type* parameter describes the object type which can be one of "dice", "toycar", "battery", and "dontcare". The "dontcare" object type is used for the other objects, such as tea candles, highlighters, and spoons, which serve as negative examples during training. These objects are ignored by DetectNet during training. The *bbox* parameter is an ordered set of four coordinates that define the top-left, and the bottom-right vertices of the ground-truth bounding box. The *rotation*_y parameter defines the rotation of the object around the camera's Y-axis, and it is further investigated in section 5.2. The *score* parameter defines the model's detection confidence score it is used for test-time performance evaluation in section 4.1. Figure 7 shows the annotation file that corresponds to the subfigure depicting two dices in Figure 8. Examples of annotated images with the ground-truth bounding boxes superimposed are shown in Figure 8, and more are included in Appendix I: Examples of Annotated Images.

```
1 dice 0.0 0 0.0 438 128 538 241 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 dice 0.0 0 0.0 156 158 247 267 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Figure 7: Example of an annotation file.

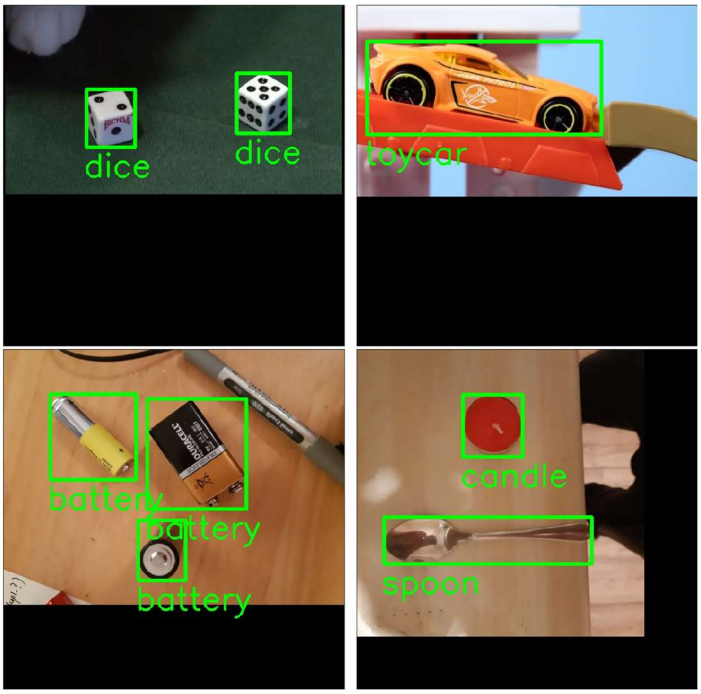


Figure 8: Examples of annotated images.

The summary of the dataset is shown in Table 1. The dataset was summarized, analyzed, and visualized with the script included in Appendix J: Data Analysis and Visualization Python Script.

Table 1: Dataset summary.

	training+validation					test					training+validation+test (full)				
[1]obj	tc[2]	b[3]	d[4]	o[5]	all[6]	tc	b	d	o	all	tc	b	d	o	all
img[7]	455	529	423	187	1524	40	40	40	0	120	495	569	463	187	1644
bbox[8]	706	872	833	237	2648	49	56	62	0	167	755	928	895	237	2815
bbox/img[9]	1.55	1.65	1.97	1.27	1.74	1.23	1.40	1.55	0	1.39	1.53	1.63	1.93	1.27	1.71
h[10]	128	148	73	170	121	96	150	76	0	107	126	148	73	170	120
w[11]	238	134	69	154	143	200	156	80	0	141	236	136	70	154	143

[1]: obj stands for object type; [2]: tc stands for toycar; [3]: b stands for battery; [4]: d stands for dice; [5]: o stands for other objects (highlighter, candle, spoon); [6]: all stands for all of the objects; [7]: img stands for the number of images in the dataset that show a certain object; [8]: bbox stands for the number of bounding boxes in the dataset of a certain object type; [9]: bbox/img stands for the number of bounding boxes per image of a certain object type; [10]: h stands for the average height of the bounding boxes of a certain object type; [11]: w stands for the average width of the bounding boxes of a certain object type

The full dataset includes a total of 1644 images and 2815 bounding boxes resulting in an average of 1.71 bounding boxes per image. The total number of bounding boxes comprises 495 toy cars, 569 batteries, 463 dices, and 187 other objects. The training dataset is used to train the object detection model as described in section 3.4.1. The validation dataset is used for model validation and hyperparameters search as described in sections 3.4.1 and 3.4.2, respectively. The test images remain unseen to the model and are used for testing as described in section 4.1. The training and the validation datasets include 1524 images and 2648 bounding boxes, of which 455 are toy cars, 529 are batteries, 423 are dices, and 187 other objects. The test dataset includes 120 images and 167 bounding boxes, of which 40 are toy cars, 40 are batteries, 40 are dices, and 0 is other objects. The frequency distribution of objects per image is shown in Figure 9.

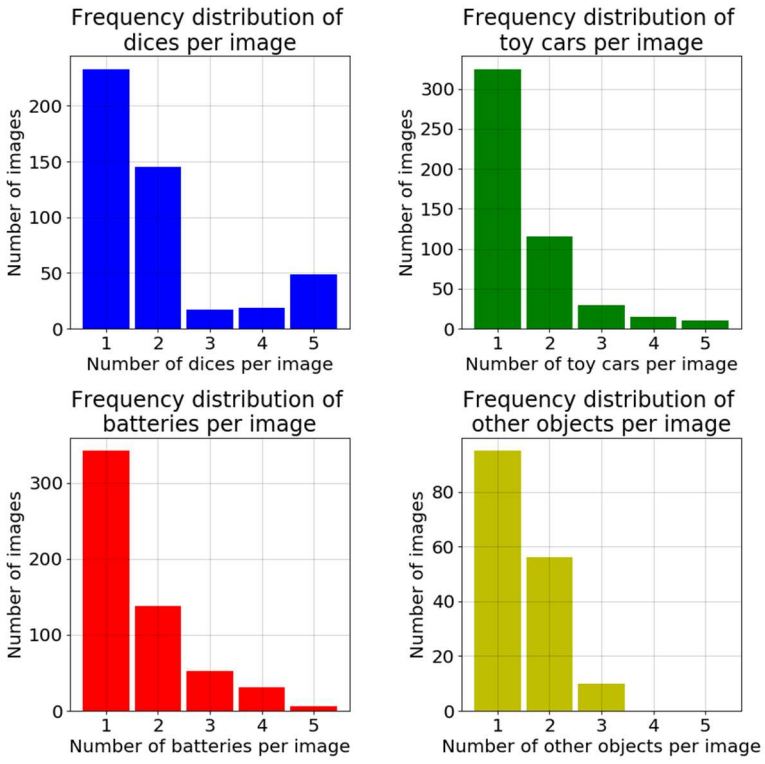


Figure 9: The frequency distribution of objects per image.

As shown in Figure 9, more than half of the images include one bounding box, and less than half of them include two, three, four, and five bounding boxes. Similarly to the PASCAL VOC challenge, every image in the project contains at least one ground-truth bounding box. The maximum number of bounding boxes in an image is five. The distribution of the width and height of the bounding boxes is visualized in Figure 10. In Figure 10, the best DetectNet sensitivity range between 40 x 40 and 400 x 400 pixels is marked by the black dashed lines. The bounding widths and heights both have a positively skewed frequency distribution with a mean width of 143 pixels and a mean height of 120 pixels. The dice bounding boxes have a mean width of 70 pixels and a mean height of 73 pixels, which support the tendency that the dices tend to have square-shaped bounding boxes. Many of the bounding boxes of toy cars are wider than taller, which is supported by the mean width of 236 pixels and a mean height of 126 pixels. The standard deviation of toy car bounding box widths is the greatest among the objects with a value of 131 pixels, which indicates that the dataset includes a diverse set of toy car images such as top view, side view, bottom view. The battery bounding boxes have a greater height than width, which is indicated by the mean width of 136 pixels and the mean height of 148 pixels. The bounding boxes of the other objects do not show any evident tendency.

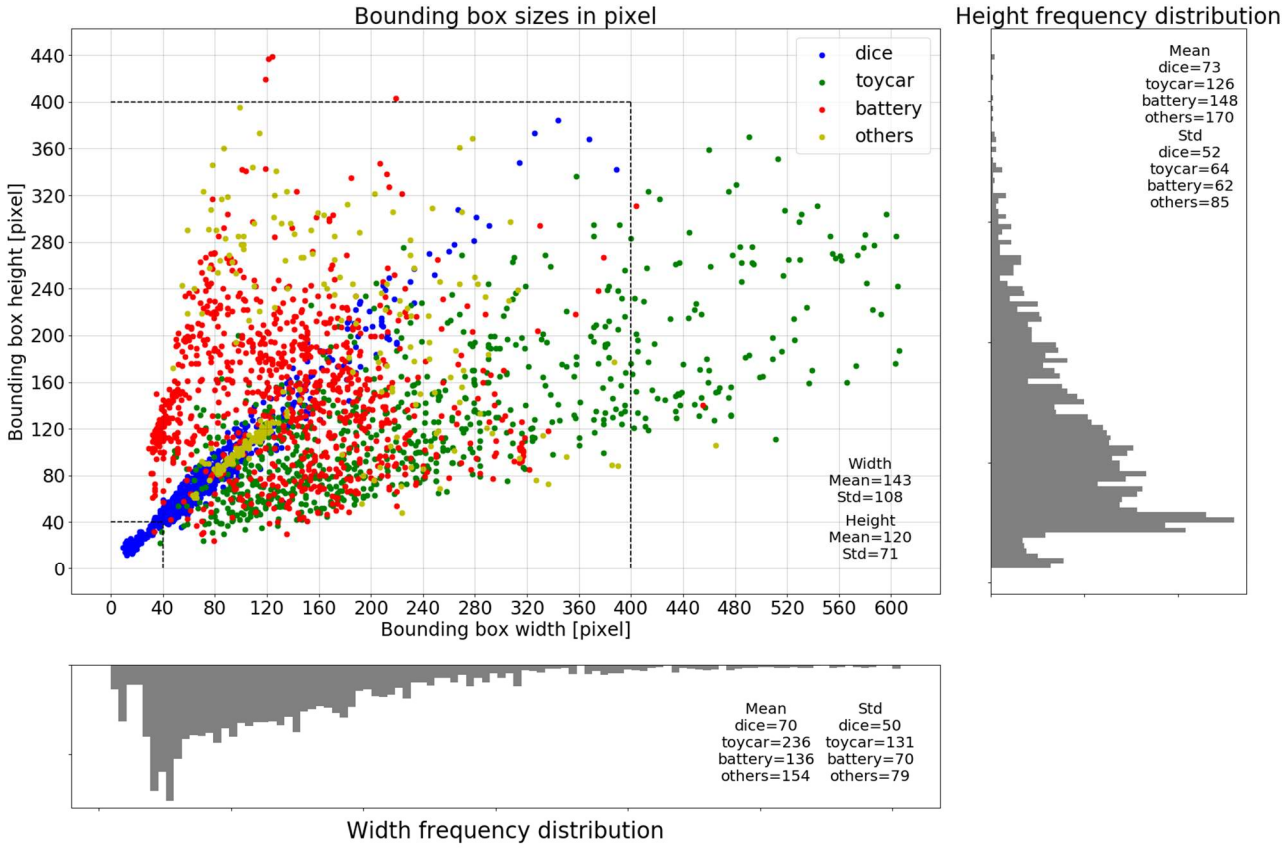


Figure 10: Visualization of bounding box sizes.

3.4. Practical Implementation

3.4.1. The Training and Validation of the Object Detection Model

The model training and validation setup in DIGITS includes importing the dataset, creating the object detection model, setting hyperparameters such as the learning rate, and optionally specifying pre-trained parameters for transfer learning. Firstly, the dataset described in section 3.3 was imported into DIGITS, then the parameter optimization algorithm and the object detection model were specified. The model and the optimizer parameters are as shown in Table 2.

Table 2: Final Model Training Parameters.

Optimization Algorithm	
Algorithm	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-6}$)
Number of training epochs	200
Mini-batch size	5
Base learning rate	10^{-4}
Learning rate decay policy	Exponential decay with 0.995 decay rate
Object Detection Algorithm	
Algorithm and model	DetectNet
Number of object classes to detect	3
Pre-trained parameters	BVLC ImageNet GoogLeNet

Discussed in section 2.1.8, the Adam parameter optimization algorithm was used in the project for its superb statistical properties and fast convergence rate. The epoch number was set to 200, which means that each image in the training dataset was forward and backpropagated 200 times during training. The mini-batch size was set to 5, which is limited by the compute capability of the host machine GPU described in section 3.2. Following the results of the hyperparameter search discussed in section 3.4.2, the base learning rate was set to 10^{-4}, and it was decreased over time using an exponential learning decay policy with a decay rate of 0.995. The DetectNet object detection model, which was discussed in section 3.1, was trained to detect the three object classes described in section 3.3. For the starting point of training, the BVLC ImageNet pre-trained GoogLeNet parameters were used as described in section 3.1. Screenshots and accompanying description of the final model and optimizer setup in DIGITS are included in Appendix K: Training and Validation Setup of the Final Model in DIGITS. Figure 11 shows the DIGITS interface during training and validation that provides visualizations of training and validation performance metrics. The metrics include the coverage and bounding box losses, which are discussed in section 3.1. Further to the losses, the interface also shows the precision, the recall, and the mAP of the three object classes that include toy cars (class0), dices (class1), and batteries (class2). The metric curves

are shown as a function of the mini-batch iterations and epochs. The training performance metric curves in Figure 11 show that the model is learning to correctly detect the objects in the training dataset as the training losses decrease and the training precisions, recalls, and mAPs increase. The validation performance metrics show that the model is getting better at generalizing to unseen examples as the corresponding metrics follow their training counterparts. While the recall and the precision in Figure 11 are identical to that described in section 2.2.2, it is important to note that DIGITS computes the mAP for each object class and it differs from the mAP metric used in sections 3.4.2 and 4.1. While in the project the mAP describes the general model performance across all object classes in the report, DIGITS computes the mAP for each object class as the product of the recall and the precision [28]. The training time of the final model with 5 980 727 learnable parameters took approximately 24 hours with the hardware setup discussed in section 3.2. DIGITS saves a snapshot of the trained model parameters at each defined epoch and the end of the training. The snapshot includes an inference version of the model description file similar to that included in Appendix G: DetectNet Description File for Training, and the parameters of the trained DetectNet model. The trained model was first tested within DIGITS, which provides visualizations of the different layers across the network during network inference. The network inference visualizations of the layers mentioned in section 3.1 are included in Appendix L: Visualization of Layers During Inference within DIGITS. The trained model was downloaded to the Jetson and real-time object detection was launched with the script included in Appendix M: DetectNet Inference Python Script.

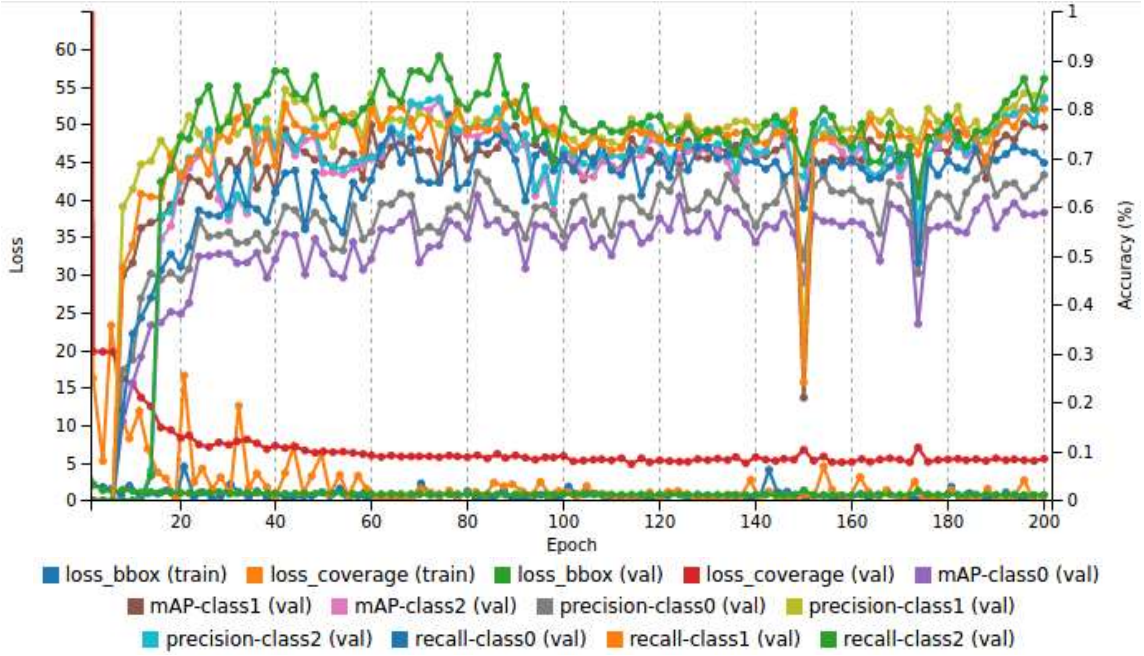


Figure 11: Training and validation performance metrics in DIGITS.

3.4.2. Object Detection Model Hyperparameter Search

The non-learnable parameters, or hyperparameters, of DetectNet, include the dropout rate, the learning rate and its decay policy, the normalization layer parameters, and the parameter initialization scheme, each of which was discussed in sections 2.1.7, 2.1.8, 0, and 2.1.10, respectively. The set of hyperparameters that gave the best model performance metrics was found through a hyperparameter search, during which the model was trained with different values of hyperparameters and its performance was tested on the validation set. The hyperparameter search was a grid search, in which five learning rates in the set $\{10^x \mid x \in \mathbb{Z}\ and\ x \in [-2, -6]\}$ were tested while the rest of the hyperparameters were kept constant. Therefore, five models with different learning rates were trained as described in section 3.4.1, each of which took 24 hours with the hardware setup described in section 3.2. Table 3 shows the results in terms of the object class APs and the model mAP for five different IoU thresholds, the description of which is provided in section 2.2.2.

Table 3: Learning rate hyperparameter search.

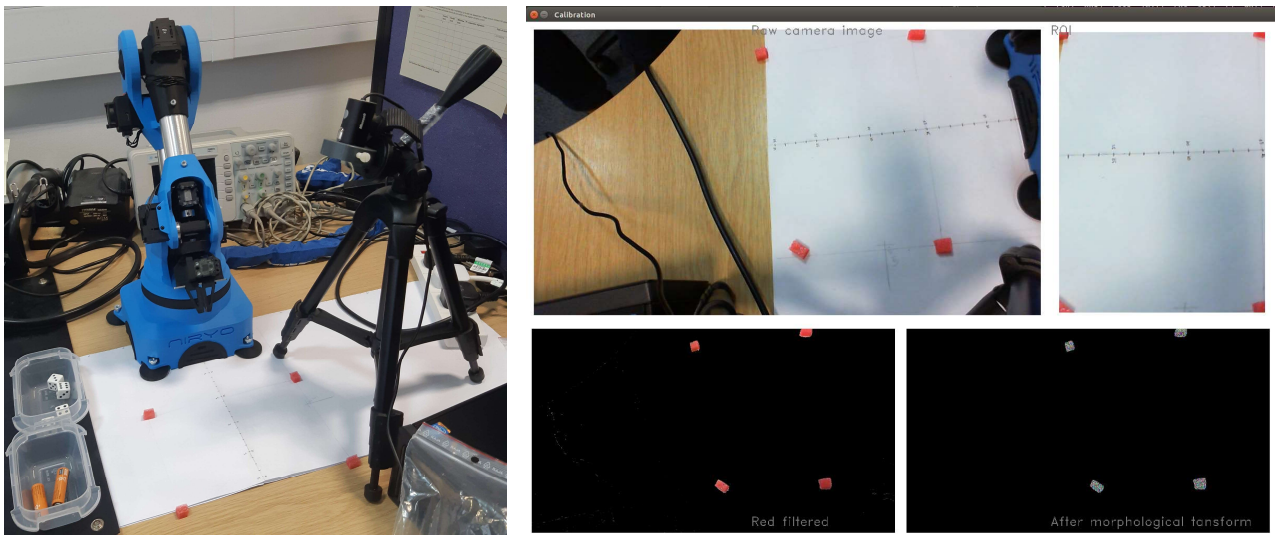
Learning rate	Object type	Metric	0.5 IoU[3]	0.6 IoU	0.7 IoU	0.8 IoU	0.9 IoU
10^{-2}	toycar	AP[1]	0	0	0	0	0
	battery	AP	0	0	0	0	0
	dice	AP	0	0	0	0	0
	all	mAP[2]	0	0	0	0	0
10^{-3}	toycar	AP	0.83	0.81	0.69	0.38	0.02
	battery	AP	0	0	0	0	0
	dice	AP	0.94	0.89	0.86	0.77	0.41
	all	mAP	0.59	0.57	0.52	0.38	0.14
10^{-4}	toycar	AP	0.97	0.96	0.95	0.82	0.31
	battery	AP	1.0	0.97	0.96	0.85	0.60
	dice	AP	0.98	0.98	0.98	0.98	0.64
	all	mAP	0.98	0.97	0.96	0.88	0.52
10^{-5}	toycar	AP	0.97	0.89	0.86	0.44	0.08
	battery	AP	0.99	0.96	0.92	0.76	0.33
	dice	AP	0.94	0.92	0.87	0.78	0.37
	all	mAP	0.97	0.92	0.88	0.66	0.26
10^{-6}	toycar	AP	0	0	0	0	0
	battery	AP	0	0	0	0	0
	dice	AP	0	0	0	0	0
	all	mAP	0	0	0	0	0

[1]AP: Average Precision; [2]mAP: mean Average Precision; [3]IoU: Intersection over Union threshold

As shown in Table 3, the best learning rate turned out to be 10^{-4}. The model trained with this learning rate yielded generally higher per-object class APs than other models, which resulted in a validation mAP above or close 90 % except for the 0.9 IoU threshold when it achieved 52 %. As a result of the hyperparameter search, the learning rate of the final model was set to 10^{-4}.

3.4.3. The Vision System of the Pick-and-Place Machine

The trained models were transferred onto the Jetson Nano that served as the vision system of a pick-and-place machine with a Niryo One robotic arm. The pick-and-place machine, which is shown in Figure 12 (a), operates in a mutual coordinate system that is set up during calibration. The calibration window of the Jetson Nano is shown in Figure 12 (b).



(a) (b)

Figure 12: (a) The pick-and-place machine, and (b) its calibration.

The camera of the Jetson Nano is calibrated to the mutual coordinate system through perspective transformation, which translates the 3D view of the coordinate system into a 2D image. The four points of the transformation are the four vertices of the rectangular area, which are marked by four pieces of a red sponge. To find these points, firstly, a red filter is used to extract the four red blobs from the raw camera frame. The raw frame and the red filtered frame are shown in the top-left and the bottom-left parts of the calibration window. Then, morphological noise removal techniques, such as morphological opening and closing, are used to eliminate the potential reddish spots from the filtered image as shown in the bottom-right of the calibration window. The middle coordinates of the remaining pixel clusters, which are found through k-means clustering the remaining blobs with four clusters, are then used in the perspective transformation as shown in the top-right part of the calibration window. Once calibrated, the vision system predicts the location and the type of objects within the mutual coordinate system and sends this information to the robotic arm via an SSH server. The arm uses the information to sort the objects in the rectangular area. Examples of detected objects from the camera's perspective are included in Appendix N: Examples of Real-Time Object Detection. Since the operation of the robotic arm was part of another student's project, it is not discussed in detail in the report.

4. Results

4.1. The Object Detection Model Results

Having trained the final model as described in section 3.4.1, it was tested on the unseen test images within DIGITS. The object detection performance of the model was evaluated using the PASCAL VOC evaluation scheme and metrics discussed in 2.2.2. Whereas the PASCAL VOC evaluation scheme uses a single IoU threshold of 0.5, a total of five IoU thresholds in the set $\{x \times 10^{-1} | x \in \mathbb{Z}\ and\ x \in [5,9]\}$ were used in the project to better characterize the model performance. Furthermore, fifty equally-spaced confidence score thresholds were sampled from the interval $[0, 3.5]$. The detections produced by the model were classified as true positive, false positive, and false negative using the model's detection confidence score and the IoU of the predicted and the ground-truth bounding boxes. For each IoU threshold, the precision and the recall were calculated according to Equations 19 and 20, and the precisions were interpolated using Equation 21. Then, the per-object class APs were computed as the average of the interpolated precisions sampled at recalls in the set $\{x \times 10^{-1} |\ x \in \mathbb{Z}\ and\ x \in [0,10]\}$, as shown in Equation 22. Finally, the IoU threshold specific mAP of the model is the average of the per object-class APs. Table 4 shows the final model's performance metrics. Table 4 and Figure 13 were produced using the script included in Appendix P: Performance Analysis and Visualization Script.

Table 4: Object detection performance summary.

Dataset	Object type	Metric	0.5 IoU[3]	0.6 IoU	0.7 IoU	0.8 IoU	0.9 IoU
training	toycar	AP[1]	1.00	1.00	1.00	0.99	0.83
	battery	AP	1.00	1.00	1.00	1.00	0.92
	dice	AP	1.00	1.00	1.00	1.00	0.90
	overall	mAP[2]	1.00	1.00	1.00	1.00	0.88
validation	toycar	AP	0.97	0.96	0.95	0.82	0.31
	battery	AP	1.00	0.97	0.96	0.85	0.60
	dice	AP	0.98	0.98	0.98	0.98	0.64
	overall	mAP	0.98	0.97	0.96	0.88	0.52
testing	toycar	AP	1.00	1.00	1.00	0.94	0.47
	battery	AP	0.99	0.99	0.97	0.91	0.66
	dice	AP	1.00	1.00	1.00	1.00	0.73
	overall	mAP	1.00	1.00	0.99	0.95	0.62

[1]AP: Average Precision; [2]mAP: mean Average Precision; [3]IoU: Intersection over Union threshold

The final model's confidence and IoU thresholds were both set to 0.5 as described in section 5.1. For this threshold setting, some of the test images with the superimposed predicted bounding boxes, confidence scores, and IoU scores are included in Appendix O: Examples of Detected Objects in Images. In the images, the absence of a predicted bounding box means a missed detection. The precision and the recall are functions of the confidence and the IoU thresholds, which is visualized

in the form of Precision-Recall (PR) curves. The PR curves for the model's test performance of detecting toy cars, batteries, and dices are shown in Figure 13 (a), (b), and (c). While the dashed lines connect the model's recall and precision pairs for different IoU and the confidence threshold combinations, the solid lines connect the interpolated precisions. Some of the PR curves overlap, which is the most pronounced in the case of dices. The overlap does not mean the absence of curves. Figure 13 (d) shows some of the missed detections and misdetections at test-time, in which *gt* stands for ground-truth bounding box (green), *det* stands for predicted bounding box (yellow), *d* stands for dice, *t* stands for toy car, and *b* stands for battery.

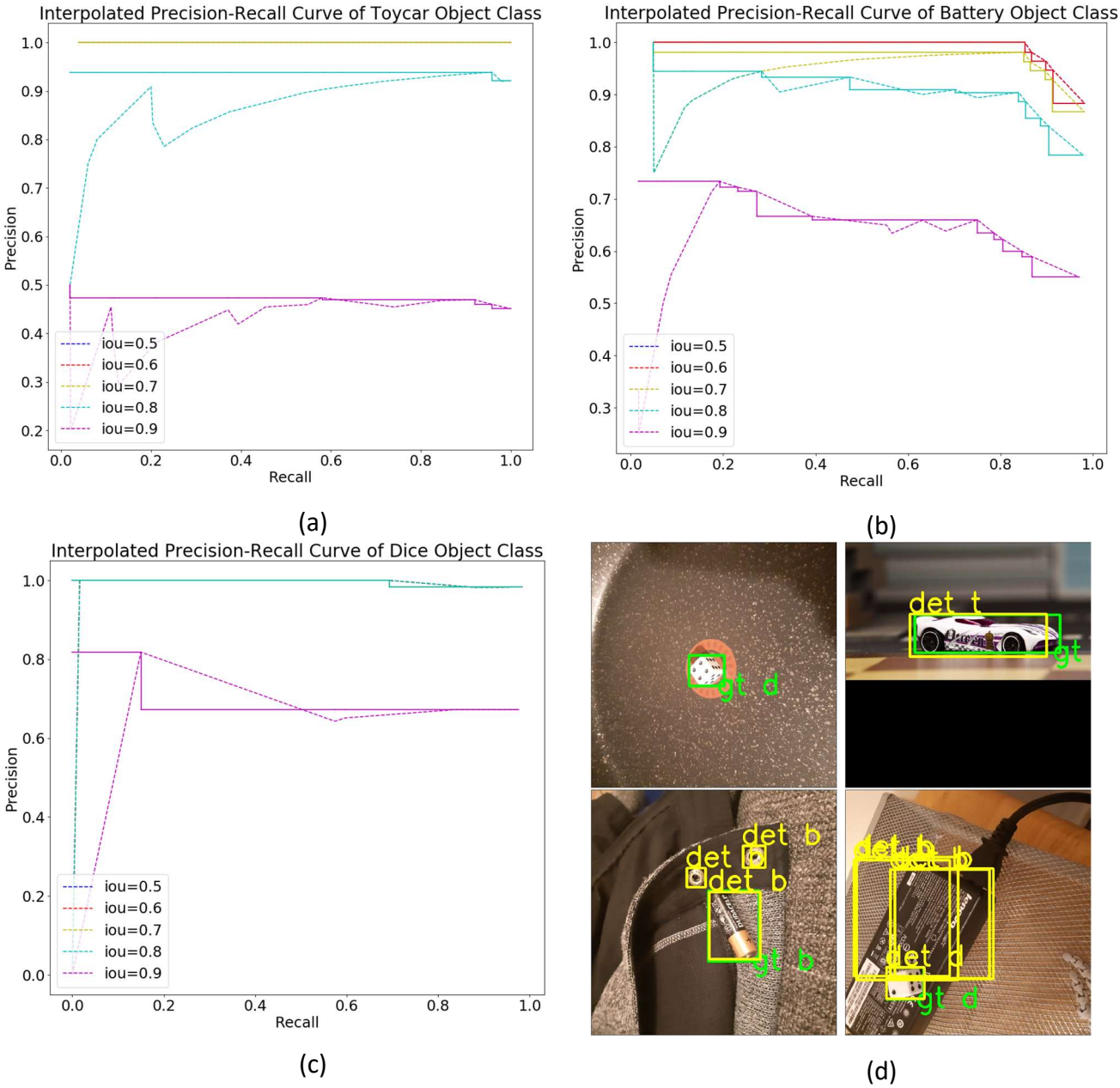


Figure 13: The test dataset PR curves for toy cars (a), batteries (b), dices (c), and misdetections and missed detection at test-time (d).

4.2. The Vision System Results

The trained model was deployed on the Jetson Nano for real-time object detection, which served as the vision system of the pick-and-place machine. The IoU and the confidence threshold of the vision system were both set to 0.5 as described in section 5.1. Figure 14 shows the inference speed of the trained DetectNet model running on the Jetson Nano in the case of nine real-life examples included in Appendix N: Examples of Real-Time Object Detection. The y-axis shows the names of the real-life examples and two bar plots for the time taken by the different CPU and GPU inference processes on the Jetson Nano.

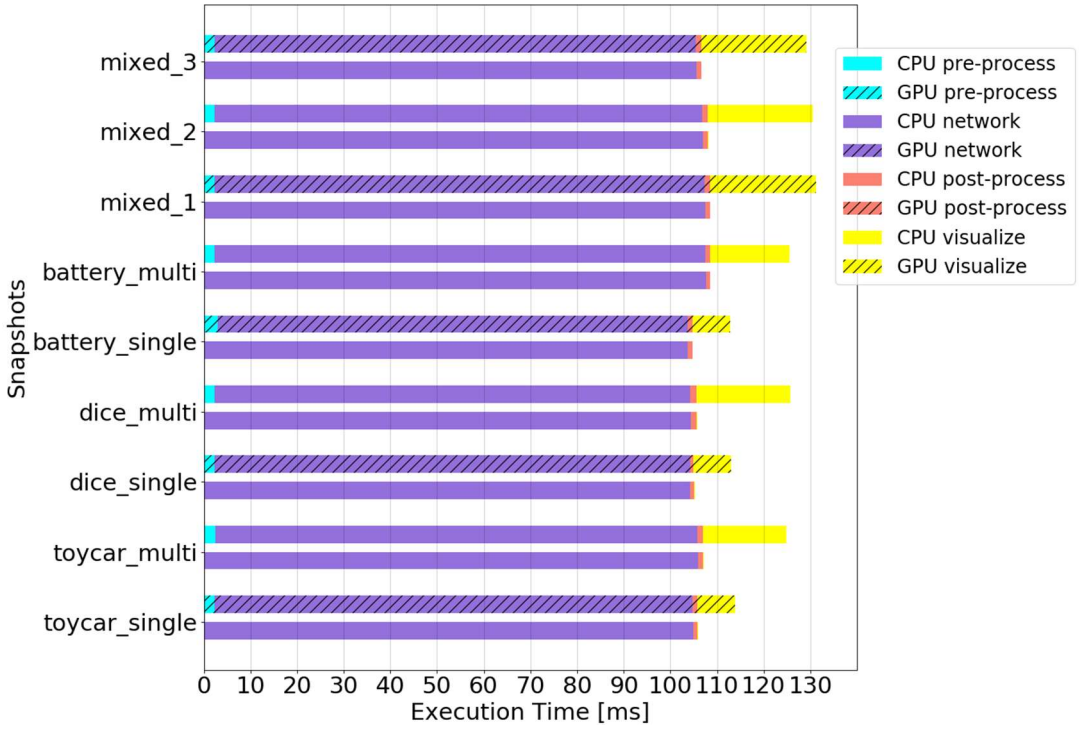


Figure 14: Inference time of the Jetson Nano.

The performance of the pick-and-place machine was evaluated through the live testing of object sorting within the mutual coordinate system. Unfortunately, the gripper of the robot arm was insufficient in size to consistently grab toy cars. Therefore, the live testing evaluation was conducted for sorting dices and batteries, the results of which are shown in Table 5.

Table 5: Live testing of the sorting performance of the pick-and-place machine.

Object type	All picks	Successful picks	Time Δ_1 [s][1]	Time Δ_2 [s][2]
dice	20	20	1.13	9.57
battery	30	29	1.19	9.8

[1]Time Δ_1 [s]: The time in seconds passed from object appearance to start of arm movement (based on the average of five measurements); [2]Time Δ_2 [s]: The time in seconds passed from object appearance to end of pick-and-place routine (based on the average of five measurements)

5. Discussion

5.1. Discussion of the Object Detection Model Results

The training dataset was used to optimize the learnable parameters of DetectNet for 200 training epochs, and the validation dataset was used for model validation and hyperparameter search. Therefore, only the test dataset includes images unseen to the model, which is why it was used to conduct a representative evaluation of the real-life model performance.

Figure 13 shows the PR curves of the test performance of the model for five IoU thresholds. Figure 13 (b) shows the model's performance for detecting batteries and it is used as an example to discuss the relationship between the PR curves and Table 4. In this subfigure, each arbitrary point along the PR curves maps an IoU and a confidence threshold combination from the sets mentioned in section 4.1 to a pair of precision and recall. In other words, the PR curves describe the model's performance in the case of different threshold settings. The confidence threshold increases from right to left along the trajectory of the curves. For instance, on the trajectory of the 0.8 IoU threshold curve at lower confidence thresholds, there exists a section that maps a precision of 0.9 to a recall of 0.8. In this threshold setting, the 0.8 IoU threshold means that the predicted and the ground-truth bounding boxes are required to overlap at least in 80% of their combined area, as defined in Equation 18. According to Equations 19 and 20, the precision of 0.9 means that 90 % of all the detected batteries were in fact batteries, and the recall of 0.8 means that 80 % of all the batteries were successfully detected. In general, lower confidence thresholds cause the model to produce fewer false negatives and more false positives that result in lower precision and higher recall. Inversely, higher confidence thresholds cause the model to produce more false negatives and fewer false positives that result in higher precision and lower recall. Using Equation 21 to average the interpolated precision values across the 0.8 IoU threshold PR curve yields a battery AP of 0.91. The battery AP of 0.91 means that on average 91 % of all of the detected batteries are in fact batteries. The same statistic is found at the intersection of the battery row and the 0.8 IoU threshold column of the test section in Table 4. The toy car and dice APs for the 0.8 IoU threshold are derived analogously. Averaging the three per-object class APs for the 0.8 IoU threshold yields the ultimate model performance measure, the mAP of the model. As shown in the test section of Table 4, the mAP of the model for the 0.8 IoU threshold is equal to 0.95. The 0.95 mAP means that 95% of all the detections produced by the model were correct. The same analysis was conducted for the other four IoU thresholds, and the results are shown in Table 4. In general, as the required overlap

between the predicted and the ground-truth bounding boxes increases, the mAP decreases. For IoU thresholds smaller than 0.9, the mAP stays well above 90 %, which indicates a remarkable model performance on unseen images. In the case of the 0.9 IoU threshold, the mAP drops to 62 %. This means that on average more than 3 out of 5 of all the model detections are correct even for the strictest bounding box overlap requirement. Following this analysis, the confidence score threshold and the IoU threshold of the final model were both set to 0.5 to achieve a good trade-off between precision and recall that yields a high mAP.

The final model generally did not achieve 100 % mAP for higher IoU thresholds at test-time, because of missed detections and misdetections, some of which are shown in Figure 13 (d). The upper-left image in the subfigure shows a dice that stays undetected by the model for every IoU and confidence threshold combination. The model potentially fails to detect the dice due to the high light intensity or the circular shape around it. The upper-right image in the subfigure shows that the model correctly detects the toy car, but the predicted and the ground-truth bounding boxes have an IoU of 0.79. This means that the detection counts as true positive for every IoU thresholds below 0.8, and it counts as a false negative for IoU thresholds above 0.8. Undetected objects and inaccurate bounding box predictions increase the false-negative rate that results in a drop in the recall, AP, and mAP. The lower-left and the lower-right images in the subfigure show interesting cases of misdetections. While the lower-left image shows a true positive battery detection that has an IoU greater than 90 %, it also shows that the model incorrectly detects some jacket buttons as batteries. It is interesting because one might argue that jacket buttons do look like batteries from top-view or bottom view. The lower-right image shows a true positive dice detection with IoU close to 100 %. However, the model also produces four false-positive battery detections for the text written on the side of a laptop charger. This might be due to the learned convolutional features of the trained model that mistake certain texts for batteries. Misdetections increase the false-positive rate that results in decreased precision, AP, and mAP.

Areas of improvement include the elimination of the missed detections and misdetections, which limit the performance of the object detection model. For comparison, while the project dataset includes 1644 images with 2815 bounding box annotations, the ImageNet [2] database includes a total of 14,197,122 images and 1,034,908 images with bounding box annotations at the time of writing. Since it is a collectively agreed in the deep learning that the model performance is limited by the availability of labeled data, substantially increasing the project dataset size would likely improve the performance of the trained model. In particular, increasing the number of images

of other objects from the current 187 to around three times as many as the number of dices, batteries, or toy cars, would help bring down the number of false-positives and misdetections as the model would see more negative examples during training. Another solution could be to train the model for more than 200 epochs. However, too many training epochs could potentially cause the model to overfit the training data and result in worse test performance. If this approach was taken, the learning rate would need to be downscaled and regularization and data augmentation techniques would need to receive extra attention to prevent over-fitting. Furthermore, performance improvements could be achieved by a more educated hyperparameter search scheme that extends the non-learnable parameter optimization to the learning rate decay policy and rate, dropout rate, and LRN parameters in addition to the learning rate. For instance, it is proposed in [40] that random search outperforms grid search given the same computational budget since randomness searches "a larger, less promising configuration space". As mentioned in sections 3.4.1 and 3.4.2, one training experiment took 24 hours on the host machine described in section 3.2. The most apparent limitation of the host machine hardware is that the Adam optimizer could use a mini-batch of only five training images. However, mini-batch sizes that speed up convergence are usually higher than that. Therefore, expanding the training dataset size, increasing the training epoch number, and extending the hyperparameter search would all substantially increase the training time unless higher-performance GPUs were used in the host machine. For instance, while the host machine GPU has a compute capability of 5.0 and memory of 4 GB, the NVIDIA TITAN RTX GPU has a compute capability of 7.5 and a memory of 24 GB [41].

5.2. Discussion of the Vision System Results

Figure 14 shows the execution time of the different CPU and GPU processes of the Jetson Nano in the case of the nine different real-time object detection examples included in Appendix N: Examples of Real-Time Object Detection. The processes include pre-processing, network inference, post-processing, and visualization. As shown in Figure 14, the average time taken by the CPU and the GPU of the Jetson to process a single frame is under approximately 130 ms. Since the processes overlap, the platform could run the model at a consistent speed of 10 frames per second (FPS), which allowed the platform to serve as the real-time object detection vision system of the pick-and-place machine. The performance of the pick-and-place machine was evaluated in the live testing of various sorting exercises. As mentioned in section 4.1, since toy cars were generally too big for the gripper of the arm, the live testing was conducted for sorting dices and batteries only. In the sorting

exercises, the machine aimed to detect, pick, and place the different objects into different designated baskets located next to the mutual coordinate system. Table 5 shows that while the machine could successfully pick and place all of the 20 dices, it made an error during sorting 30 batteries. On average, it took 1.13 s for the machine to start the arm movement from the appearance of dices in the rectangular area and it took 9.57 s for it to execute a single pick-and-place operation. The same time estimates in the case of batteries are 1.19 s and 9.80 s, respectively. This execution time is suitable for prototyping in academia but would need to be decreased for industrial applications. It was found that the gripper orientation relative to the battery orientation affected the success of sorting. When the gripper attempted to grab batteries lengthways, it failed on multiple occasions. Therefore, the pick-and-place machine was programmed to avoid the aforementioned relative orientation as much as possible. The performance of the pick-and-place machine could have been better characterized had more data been recorded during live testing. The cause of the limited amount of sorting exercise test results is described in Appendix E: The Effects of Coronavirus (COVID-19) on the Project.

Areas of improvement include helping the pick-and-place machine be able to pick up objects irrelevant to the relative gripper orientation and accelerating the pick-and-place routine. A solution to the relative orientation issue could be the use of the *rotation_y* KITTI object detection annotation parameter during model training. As mentioned in section 3.3, the *rotation_y* parameter defines the rotation of the object around the Y-axis in the camera's coordinate system. Including the *rotation_y* parameter in the object annotation files and incorporating a *rotation_y* loss term in the loss function of DetectNet could help the model learn to identify the orientation of objects. Then, the orientation information could be passed to the robotic arm alongside the type and location of the objects. In turn, the robotic arm could be programmed to include the object orientation information in the pick-and-place routine. The acceleration of the pick-and-place routine would require a speed up in the robotic arm movement since the object detection model only contributes the aforementioned 130 ms to the execution time. Even though the speed of the vision system is not an issue in the project, it could be improved if a more powerful embedded AI computing platform was used. For instance, the Jetson AGX Xavier platform includes a 512-core NVIDIA Volta GPU with a compute capability of 7.2 and a memory of 32 GB [41]. On the contrary, the Jetson Nano includes a 128-core NVIDIA Maxwell GPU with a compute capability of 5.3 and a memory 4 GB, as described in section 3.2.

6. Conclusions

The completed project work fulfilled the overall aim to detect select objects in real-time with a CNN-based object detection model on an embedded AI computing platform and adapt the platform to serve as the vision system of a pick-and-place machine. The aim was achieved through the completion of the objectives listed in section 1.2.

Firstly, the mathematics of deep learning and CNNs were discussed with an emphasis on the role of convolutional, fully-connected, and pooling layers in CNNs, the back-propagation algorithm, and common parameter optimization and initialization techniques. Then, several proven object detection algorithms were reviewed, including the GoogLeNet-based DetectNet, which was selected for the project. Then, an object detection dataset was created that included images of the dices, toy cars, batteries, and a few other objects that served as negative examples. Having reviewed the necessary theory and created the dataset, DetectNet was trained in the NVIDIA DIGITS deep learning training system, and the best set of model parameters was found through a hyperparameter search. The performance of the trained model was evaluated using the PASCAL VOC evaluation scheme and metrics, and results showed that the model learned to detect the select objects in the unseen test images. Finally, the model was deployed on the NVIDIA Jetson Nano embedded AI computing platform for real-time object detection, and the platform was adapted to serve as the vision system of a pick-and-place machine. The machine was tasked to sort dices and batteries, and its performance was evaluated in live sorting tests.

The discoveries made throughout the project are of great interest to many industries. This is indicated by press releases about some industrial applications with the Jetson family [42] and the number of followers of the GitHub repositories of the DIGITS system [31] and the Jetson Nano project [43]. However, detailed instructions and guidelines on how to train DetectNet to detect more than two objects and how to integrate it with a robotic arm are only scarcely available to the public. Therefore, this report is a useful addition to the available practical guidelines and instructions, that could potentially help people and companies explore real-time object detection with deep learning on an embedded GPU system.

7. References

[1] I. Goodfellow, Y. Bengio and A. Courville, "Deep Learning", MIT Press, 2016.

[2] J. Deng, W. Dong, R. Socher, L. Li, K. Li and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[3] A. Krizhevsky, I. Sutskever and G. E. Hinton, “Imagenet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, 2012.

[4] D. Küpper, M. Lorenz, K. Kuhlmann, O. Bouffault, Y. Lim, J. V. Wyck, S. Köcher and J. Schlageter, "AI in the Factory of the Future: The Ghost in the Machine" [Online], Boston Consulting Group, (2018). Available at: https://www.bcg.com/publications/2018/artificial-intelligence-factory-future.aspx [Accessed: 18 March 2020].

[5] G. Cybenko, “Approximation by Superpositions of a Sigmoidal Function,” *Mathematics of Control, Signals and Systems,* vol. 2, pp. 303-314, 1989.

[6] A. Ng, K. Katanforoosh and Y. Bensouda Mourri, "Deep Learning Specialization - Online Course on Coursera" [Online], deeplearning.ai, (2019). Available at: http://deeplearning.ai/ [Accessed: 22 March 2020].

[7] M. Lin, Q. Chen and S. Yan, “Network in Network,” *arXiv Preprint arXiv:1312.4400,* 2013.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going Deeper with Convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[9] S. Arora, A. Bhaskara, R. Ge and T. Ma, “Provable Bounds for Learning Some Deep Representations,” in *International Conference on Machine Learning*, 2014.

[10] D. O. Hebb, "The Organization of Behavior: A Neuropsychological Theory", Wiley, New York, 1949.

[11] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.

[12] D. E. Rumelhart, G. E. Hinton and R. J. Williams, “Learning Representations by Back-Propagating Errors,” in *Neurocomputing: Foundations of Research*, Cambridge, MA: MIT Press, 1988, p. 696–699.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computation,* vol. 1, pp. 541-551, 1989.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[15] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE,* vol. 86, pp. 2278-2324, 11 1998.

[16] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv preprint arXiv:1412.6980,* 2014.

[17] X. Glorot and Y. Bengio, “Understanding the Difficulty of Training Deep Feedforward Neural Networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

[18] M. Everingham, A. Zisserman, C. K. I. Williams, L. Van Gool, M. Allan, C. M. Bishop, O. Chapelle, N. Dalal, T. Deselaers, G. Dorkó and others, “The 2005 Pascal Visual Object Classes Challenge,” in *Machine Learning Challenges Workshop*, 2005.

[19] A. Karpathy, J. Johnson, F.-F. Li and S. Yeung, "CS231n: Convolutional Neural Networks for Visual Recognition" [Online], Stanford University, (2019). Available at: http://cs231n.github.io/ [Accessed: 22 March 2020].

[20] J. Yosinski, J. Clune, Y. Bengio and H. Lipson, “How transferable are features in deep neural networks?,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 3320-3328.

[21] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005.

[22] D. G. Lowe, “Object Recognition from Local Scale-Invariant Features,” in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999.

[23] P. F. Felzenszwalb, R. B. Girshick, D. McAllester and D. Ramanan, “Object Detection with Discriminatively Trained Part-Based Models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 32, pp. 1627-1645, 9 2010.

[24] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” *arXiv preprint arXiv:1312.6229,* 2013.

[25] R. Girshick, J. Donahue, T. Darrell and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” *2014 IEEE Conference on Computer Vision and Pattern Recognition,* 2014.

[26] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2016.

[27] S. Voghoei, N. H. Tonekaboni, J. G. Wallace and H. R. Arabnia, “Deep Learning at the Edge,” in *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2018.

[28] J. B. Andrew Tao and S. Sarathy, "DetectNet: Deep Neural Network for Object Detection in DIGITS", NVIDIA Developer Blog, (2016). Available at: https://devblogs.nvidia.com/detectnet-deep-neural-network-object-detection-digits/ [Accessed: 24 April 2020].

[29] The Berkeley Vision and Learning Center, *BVLC Caffe [GitHub Repository],* GitHub, 2015.

[30] D. Franklin, "Jetson Nano Brings AI Computing to Everyone", NVIDIA Developer Blog, (2020). Available at: https://devblogs.nvidia.com/jetson-nano-ai-computing/ [Accessed: 27 April 2020].

[31] NVIDIA, *DIGITS: Deep Learning GPU Training System [GitHub Repository],* GitHub, 2020.

[32] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell, “Caffe: Convolutional Architecture for Fast Feature Embedding,” *arXiv preprint arXiv:1408.5093,* 2014.

[33] NVIDIA, "CUDA Toolkit Documentation" [Computer program], NVIDIA, (2020). Available at: https://docs.nvidia.com/cuda/ [Accessed: 27 March 2020].

[34] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro and E. Shelhamer, “cuDNN: Efficient Primitives for Deep Learning,” *arXiv preprint arXiv:1410.0759,* 2014.

[35] NVIDIA, "NVIDIA TensorRT: Programmable Inference Accelerator" [Computer program], NVIDIA, (2020). Available at: https://developer.nvidia.com/tensorrt [Accessed: 27 March 2020].

[36] Python Software Foundation, "Python Language (version 2.7 and 3.7)" [Computer program], Python Software Foundation, (2020). Available at: https://www.python.org [Accessed: 27 March 2020].

[37] Microsoft Cognitive Services, "Bing Image Search REST API" [Computer program], Microsoft Cognitive Services, (2020). Available at: https://westus.dev.cognitive.microsoft.com/docs/services/8336afba49a84475ba401758c0dbf749/operations/faa10a421d7c417f95e2b31a [Accessed: 27 April 2020].

[38] Roboflow, "6 Sided Dice Dataset" [Dataset], Roboflow, (2020). Available at: https://public.roboflow.ai/object-detection/dice [Accessed: 27 April 2020].

[39] Alp, "Alp's Labeling Tool (ALT)" [Computer program], Alp, (2017). Available at: https://alpslabel.wordpress.com/2017/01/26/alt/ [Accessed: 27 April 2020].

[40] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research,* vol. 13, pp. 281-305, 2012.

[41] NVIDIA Developer, "CUDA GPUs: Recommended GPU for Developers", NVIDIA Developer, (2020). Available at: https://developer.nvidia.com/cuda-gpus [Accessed: 27 April 2020].

[42] K. Uchiyama, "Japan's Komatsu Selects NVIDIA as Partner for Deploying AI to Create Safer, More Efficient Construction Sites" [Online], NVIDIA, (2017). Available at: https://nvidianews.nvidia.com/news/japans-komatsu-selects-nvidia-as-partner-for-deploying-ai-to-create-safer-more-efficient-construction-sites [Accessed: 18 March 2020].

[43] D. Franklin, *Hello AI World: NVIDIA Jetson [GitHub Repository],* GitHub, 2020.

8. Appendices

8.1. Appendix A: Online Course Certificates

Neural Networks and Deep Learning, Deep Learning Specialization on Coursera by deeplearning.ai (URL: https://www.coursera.org/account/accomplishments/verify/44BAF4JS6NA7)

Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization, Deep Learning Specialization on Coursera by deeplearning.ai (URL: https://www.coursera.org/account/accomplishments/verify/9KDRNG5SP2KY)

Structuring Machine Learning Projects, Deep Learning Specialization on Coursera by deeplearning.ai (URL: https://www.coursera.org/account/accomplishments/verify/4RGJQD6WBALU)

Convolutional Neural Networks, Deep Learning Specialization on Coursera by deeplearning.ai (URL: https://www.coursera.org/account/accomplishments/verify/EB6Z4BMGHG5X)

8.2. Appendix B: Preparatory Exercises

Exercise 1 - Face recognition with Eigenfaces

- Aim: Recognise faces using Principal Component Analysis (PCA) and a K-Nearest Neighbors (KNN) Classifier
- Data: Labelled Faces in the Wild (http://vis-www.cs.umass.edu/lfw/)
- Main Technologies used: Python 3.7, sklearn, numpy, pandas, matplotlib.pyplot, PIL
- Exercise URL: https://github.com/mark-antal-csizmadia/eigenfaces

Exercise 2: Multilayer Perceptron (MLP) for the recognition of the MNIST hand-written digits

- Aim: Train a MLP to tecognise hand-written digits in the MNIST dataset
- Data: MNIST The MNIST Database of Handwritten Digits (http://yann.lecun.com/exdb/mnist/)
- Main Technologies used: Python 3.7, sklearn, numpy, pandas, matplotlib.pyplot
- Exercise URL: https://github.com/mark-antal-csizmadia/mlp

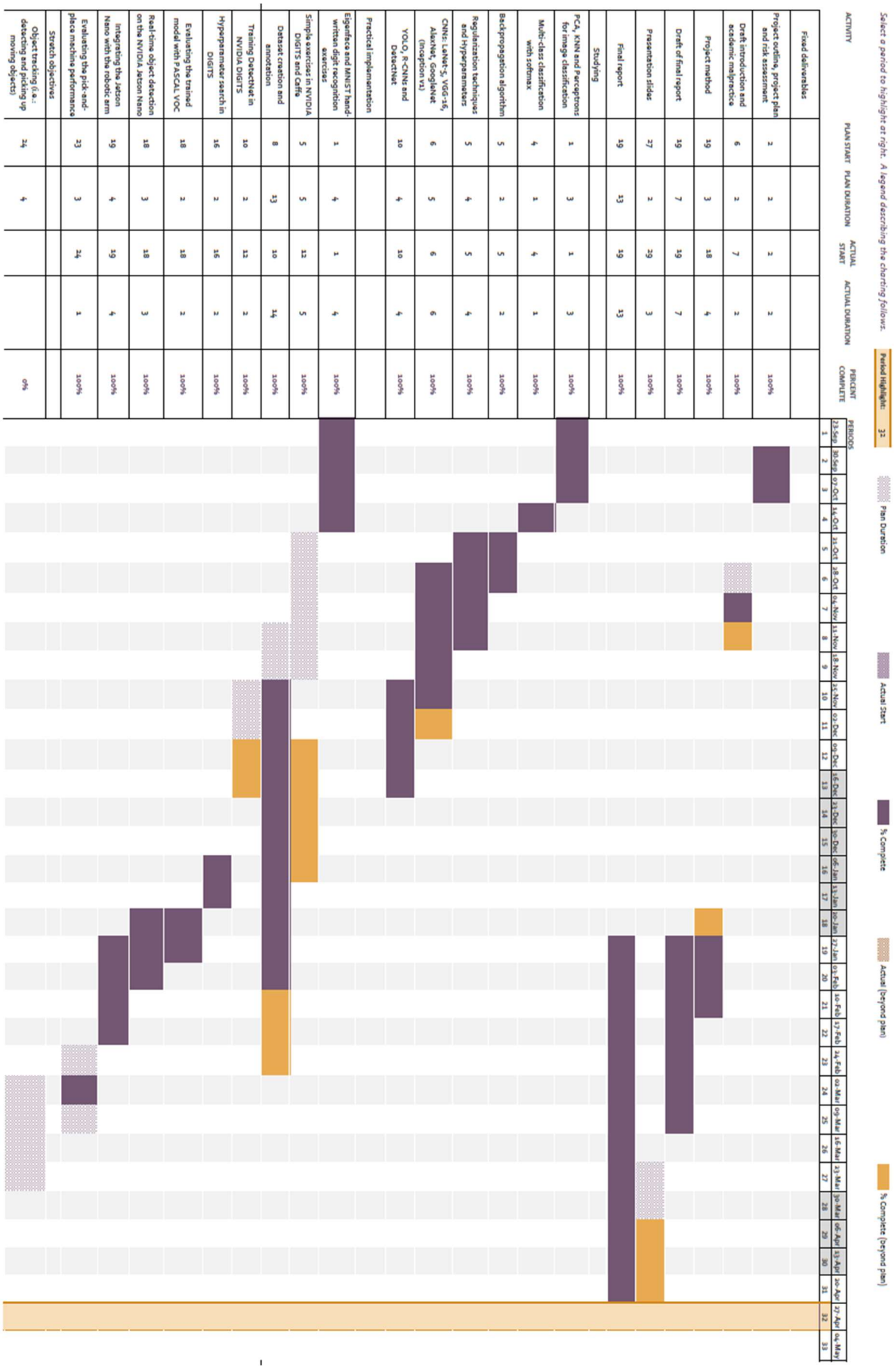
Exercise 3: Naïve implementation of Convolutional Neural Networks (CNNs) for image classification on the MNSIT and CIFAR10 databases

- Aim: Build CNNs from scratch without any deep learning library or package, and classify images in popular datasets
- Data: MNIST The MNIST Database of Handwritten Digits (http://yann.lecun.com/exdb/mnist/) and CIFAR10 (https://www.cs.toronto.edu/~kriz/cifar.html)
- Main Technologies used: Python 3.7, sklearn, numpy, pandas, matplotlib.pyplot
- Exercise URL: https://github.com/mark-antal-csizmadia/naive_cnn

Exercise 4: The Tensorflow implementation of popular CNN models such as LeNet-5 for image classification on the MNSIT and CIFAR10 databases

- Aim: Build CNNs with the Tensorflow machine learning library, and classify images in popular datasets
- Data: MNIST The MNIST Database of Handwritten Digits (http://yann.lecun.com/exdb/mnist/) and CIFAR10 (https://www.cs.toronto.edu/~kriz/cifar.html)
- Main Technologies used: Python 3.7, Tensorflow 1
- Exercise URL: https://github.com/mark-antal-csizmadia/cnn_tensorflow_1

8.3. Appendix C: Progress Report



8.4. Appendix D: Risk Assessment

Date:	Assessed by:	Checked / Validated by:	Location(s):	Assessment ref no.:	Review date:
08/10/2019	Mark Csizmadia	Dr. Hujun Yin	Sackville Street Building D45, Barnes Wallis Building, 32 Chorlton Road M15 4AU		27/01/2019

Task / premises:
Construction site waste classification (in particular, wood) for improved recycling using deep learning and Convolutional Neural Networks. Research project, mostly involves extensive self-directed learning in libraries and the use of Display Screen Equipment (DSE) and GPUs.

Activity	Hazard	Who might be harmed and how	Existing measures to control risk	Risk rating	Result
Use of Display Screen Equipment (DSE) – posture and back	Wrong setting of work chair/seat height, seat back height, seat back angle/tilt can lead to slouched posture, hunched shoulder	Student working on project	Adjusting work chair/seat so that the upper body is supported and healthy posture is possible. Taking regular breaks.	LOW	T (trivial task)
Use of Display Screen Equipment (DSE) – viewing the screen for long time spans	Flickering screen, high brightness	Student working on project	Adequate adjustment of screen and viewing distance. Taking regular breaks.	LOW	T
Use of electronic devices (NVIDIA Jetson Nano) and walking around laboratory/office	Cables and other accessories of electronic devices in the work environment (slipping, tripping, falling over cables)	Student working on project and other people in the environment	Keeping work environment tidy at all times of clothes and cables.	LOW	T
Food and drinks in laboratory/office	Ruining electronic devices by spilling liquid on them	School, owner of electronic devices	No food or drink consumption near electronic devices	LOW	T
Data generation as project progresses	Data loss or corruption	Student working on project	Data backup on external hardware and/or cloud	LOW	T

Action plan

Ref No	Further action required	Action by whom	Action by when	Done
	Create cloud storage account (data backup on external hardware and/or cloud)	Mark Csizmadia	07/10/2019	YES

8.5. Appendix E: The Effects of Coronavirus (COVID-19) on the Project

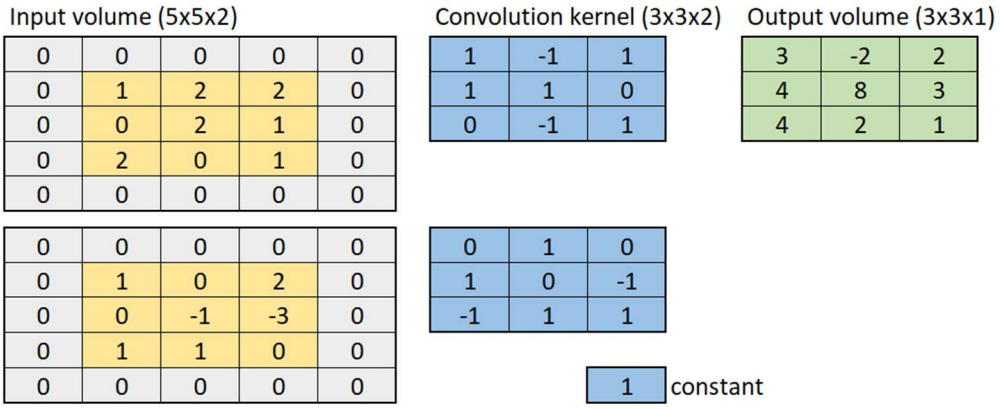
This statement describes how the project has been affected by the closure of the University's campus, experimental facilities, and computer clusters due to coronavirus (COVID-19).

Due to the inability to access the NVIDIA Jetson Nano platform, which is the property of the University of Manchester and was left at the lab, some minor parts of the vision system could not be finished. These include the final tests of the vision system calibration in different lighting conditions, the polish of the integration of the vision system with the robotic arm, and the implementation of some cosmetic improvements to the vision system interface. Furthermore, the inability to access the robotic arm also reduced the number of live test results of the pick-and-place machine sorting exercises. However, prompt actions were taken to mitigate the effects of the closure. For instance, the pick-and-place machine performance measurements presented in the report were taken on the day of the closure. Furthermore, the early anticipation of the closure allowed for recording videos and taking images of the vision system and the pick-and-place machine in action, though in early, unpolished form. The effects that could not be mitigated mostly concern the cosmetic improvement of the vision system interface, and the polish of the vision system integration with the robotic arm.

Overall, two to three weeks' worth of project work was lost due to the closure. Since most of the practical project work had already been done by that time, the remaining project work could be seamlessly completed outside the University's facilities.

8.6. Appendix F: Example of Forward-Propagation in Convolutional Layers

The example convolutional layer calculation mentioned Section 2.1.3 is shown below. The calculation uses Equation 3 and Figure 2.



$$O(1,1) + constant = O_0(1,1) + O_1(1,1) + constant =$$

$$(I_0 * K_0)(1,1) + (I_1 * K_1)(1,1) + constant =$$

$$\sum_{m=0}^{2}\sum_{n=0}^{2}[I_0(1+m,1+n)K_0(m,n)] + \sum_{m=0}^{2}\sum_{n=0}^{2}[I_1(1+m,1+n)K_1(m,n)] + constant =$$

$$\big(I_0(1,1) \times K_0(0,0) + I_0(1,2) \times K_0(0,1) + I_0(1,3) \times K_0(0,2) + I_0(2,1) \times K_0(1,0) +$$
$$I_0(2,2) \times K_0(1,1) + I_0(2,3) \times K_0(1,2) + I_0(3,1) \times K_0(2,0) + I_0(3,2) \times K_0(2,1) +$$
$$I_0(3,3) \times K_0(2,2)\big) +$$

$$\big(I_1(1,1) \times K_1(0,0) + I_1(1,2) \times K_1(0,1) + I_1(1,3) \times K_1(0,2) + I_1(2,1) \times K_1(1,0) +$$
$$I_1(2,2) \times K_1(1,1) + I_1(2,3) \times K_1(1,2) + I_1(3,1) \times K_1(2,0) + I_1(3,2) \times K_1(2,1) +$$
$$I_1(3,3) \times K_1(2,2)\big) + constant =$$

$$(1 \times 1 + 2 \times (-1) + 2 \times 1 + 0 \times 1 + 2 \times 1 + 1 \times 0 + 2 \times 0 + 0 \times (-1) + 1 \times 1) +$$

$$(1 \times 0 + 0 \times 1 + 2 \times 0 + 0 \times 1 + (-1) \times 0 + (-3) \times (-1) + 1 \times (-1) + 1 \times 1 + 0 \times 1) + 1 =$$

$$3 + 4 + 1 = 8$$

8.7. Appendix G: DetectNet Description File for Training

The original description file of DetectNet comes with the DIGITS installation and may be found on GitHub [31]. It is for the detection of one or two objects, and the input image size is different from that in the project. Therefore, it was adapted to the detection of three objects and the input mage size was changed to 640 x 640 pixels. Since these are rather non-trivial modifications, the modified description file is included in this appendix. The full description file is split into two halves – the first half is included in the column on the left-hand side, and the second half is included in the column on the right-hand side.

```
# DetectNet network

# Data/Input layers
name: "DetectNet"
layer {
  name: "train_data"
  type: "Data"
  top: "data"
  data_param {
    backend: LMDB
    source:
"examples/kitti/kitti_train_images.lmdb"
    batch_size: 10
  }
  include: { phase: TRAIN }
}
layer {
  name: "train_label"
  type: "Data"
  top: "label"
  data_param {
    backend: LMDB
    source:
"examples/kitti/kitti_train_labels.lmdb"
    batch_size: 10
  }
  include: { phase: TRAIN }
}
layer {
  name: "val_data"
  type: "Data"
  top: "data"
  data_param {
    backend: LMDB
    source:
"examples/kitti/kitti_test_images.lmdb"
    batch_size: 6
  }
  include: { phase: TEST stage: "val" }
}
layer {
  name: "val_label"
  type: "Data"
  top: "label"
  data_param {
    backend: LMDB
    source:
"examples/kitti/kitti_test_labels.lmdb"
    batch_size: 6
  }
  include: { phase: TEST stage: "val" }
}
layer {
  name: "deploy_data"
  type: "Input"
```

```
layer {
  name: "inception_4c/relu_1x1"
  type: "ReLU"
  bottom: "inception_4c/1x1"
  top: "inception_4c/1x1"
}

layer {
  name: "inception_4c/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4b/output"
  top: "inception_4c/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "inception_4c/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4c/3x3_reduce"
  top: "inception_4c/3x3_reduce"
}
layer {
  name: "inception_4c/3x3"
  type: "Convolution"
  bottom: "inception_4c/3x3_reduce"
  top: "inception_4c/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
```

```
  top: "data"
  input_param {
    shape {
      dim: 1
      dim: 3
      dim: 640
      dim: 640
    }
  }
  include: { phase: TEST not_stage: "val" }
}

# Data transformation layers
layer {
  name: "train_transform"
  type: "DetectNetTransformation"
  bottom: "data"
  bottom: "label"
  top: "transformed_data"
  top: "transformed_label"
  detectnet_groundtruth_param: {
    stride: 16
    scale_cvg: 0.4
    gridbox_type: GRIDBOX_MIN
    coverage_type: RECTANGULAR
    min_cvg_len: 20
    obj_norm: true
    image_size_x: 640
    image_size_y: 640
    crop_bboxes: true
    object_class: { src: 1 dst: 0} # 1 -> 0
    object_class: { src: 2 dst: 1} # 2 -> 1
    object_class: { src: 3 dst: 2} # 3 -> 2
  }
   detectnet_augmentation_param: {
    crop_prob: 1
    shift_x: 32
    shift_y: 32
    flip_prob: 0.5
    rotation_prob: 0
    max_rotate_degree: 5
    scale_prob: 0.4
    scale_min: 0.8
    scale_max: 1.2
    hue_rotation_prob: 0.8
    hue_rotation: 30
    desaturation_prob: 0.8
    desaturation_max: 0.8
  }
  transform_param: {
    mean_value: 127
  }
  include: { phase: TRAIN }
}
layer {
  name: "val_transform"
  type: "DetectNetTransformation"
  bottom: "data"
  bottom: "label"
  top: "transformed_data"
  top: "transformed_label"
  detectnet_groundtruth_param: {
    stride: 16
    scale_cvg: 0.4
    gridbox_type: GRIDBOX_MIN
    coverage_type: RECTANGULAR
    min_cvg_len: 20
    obj_norm: true
    image_size_x: 640
    image_size_y: 640
    crop_bboxes: false
    object_class: { src: 1 dst: 0} # 1 -> 0
    object_class: { src: 2 dst: 1} # 2 -> 1
    object_class: { src: 3 dst: 2} # 3 -> 2
  }
  transform_param: {
    mean_value: 127
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4c/relu_3x3"
  type: "ReLU"
  bottom: "inception_4c/3x3"
  top: "inception_4c/3x3"
}
layer {
  name: "inception_4c/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4b/output"
  top: "inception_4c/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 24
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4c/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4c/5x5_reduce"
  top: "inception_4c/5x5_reduce"
}
layer {
  name: "inception_4c/5x5"
  type: "Convolution"
  bottom: "inception_4c/5x5_reduce"
  top: "inception_4c/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
```

```
  }
  include: { phase: TEST stage: "val" }
}
layer {
  name: "deploy_transform"
  type: "Power"
  bottom: "data"
  top: "transformed_data"
  power_param {
    shift: -127
  }
  include: { phase: TEST not_stage: "val" }
}

# Label conversion layers
layer {
  name: "slice-label"
  type: "Slice"
  bottom: "transformed_label"
  top: "foreground-label"
  top: "bbox-label"
  top: "size-label"
  top: "obj-label"
  top: "coverage-label"
  slice_param {
    slice_dim: 1
    slice_point: 1
    slice_point: 5
    slice_point: 7
    slice_point: 8
  }
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}
layer {
  name: "coverage-block"
  type: "Concat"
  bottom: "foreground-label"
  bottom: "foreground-label"
  bottom: "foreground-label"
  bottom: "foreground-label"
  top: "coverage-block"
  concat_param {
    concat_dim: 1
  }
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}
layer {
  name: "size-block"
  type: "Concat"
  bottom: "size-label"
  bottom: "size-label"
  top: "size-block"
  concat_param {
    concat_dim: 1
  }
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}
layer {
  name: "obj-block"
  type: "Concat"
  bottom: "obj-label"
  bottom: "obj-label"
  bottom: "obj-label"
  bottom: "obj-label"
  top: "obj-block"
  concat_param {
    concat_dim: 1
  }
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}
layer {
  name: "bb-label-norm"
  type: "Eltwise"
```

```
layer {
  name: "inception_4c/relu_5x5"
  type: "ReLU"
  bottom: "inception_4c/5x5"
  top: "inception_4c/5x5"
}
layer {
  name: "inception_4c/pool"
  type: "Pooling"
  bottom: "inception_4b/output"
  top: "inception_4c/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4c/pool_proj"
  type: "Convolution"
  bottom: "inception_4c/pool"
  top: "inception_4c/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4c/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4c/pool_proj"
  top: "inception_4c/pool_proj"
}
layer {
  name: "inception_4c/output"
  type: "Concat"
  bottom: "inception_4c/1x1"
  bottom: "inception_4c/3x3"
  bottom: "inception_4c/5x5"
  bottom: "inception_4c/pool_proj"
  top: "inception_4c/output"
}

layer {
  name: "inception_4d/1x1"
  type: "Convolution"
  bottom: "inception_4c/output"
  top: "inception_4d/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 112
    kernel_size: 1
    weight_filler {
```

```
  bottom: "bbox-label"
  bottom: "size-block"
  top: "bbox-label-norm"
  eltwise_param {
    operation: PROD
  }
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}
layer {
  name: "bb-obj-norm"
  type: "Eltwise"
  bottom: "bbox-label-norm"
  bottom: "obj-block"
  top: "bbox-obj-label-norm"
  eltwise_param {
    operation: PROD
  }
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}

##############################################
######################
# Start of convolutional network
##############################################
######################

layer {
  name: "conv1/7x7_s2"
  type: "Convolution"
  bottom: "transformed_data"
  top: "conv1/7x7_s2"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    pad: 3
    kernel_size: 7
    stride: 2
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "conv1/relu_7x7"
  type: "ReLU"
  bottom: "conv1/7x7_s2"
  top: "conv1/7x7_s2"
}

layer {
  name: "pool1/3x3_s2"
  type: "Pooling"
  bottom: "conv1/7x7_s2"
  top: "pool1/3x3_s2"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}

layer {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_1x1"
  type: "ReLU"
  bottom: "inception_4d/1x1"
  top: "inception_4d/1x1"
}
layer {
  name: "inception_4d/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4c/output"
  top: "inception_4d/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 144
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4d/3x3_reduce"
  top: "inception_4d/3x3_reduce"
}
layer {
  name: "inception_4d/3x3"
  type: "Convolution"
  bottom: "inception_4d/3x3_reduce"
  top: "inception_4d/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 288
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_3x3"
  type: "ReLU"
```

```
  name: "pool1/norm1"
  type: "LRN"
  bottom: "pool1/3x3_s2"
  top: "pool1/norm1"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}

layer {
  name: "conv2/3x3_reduce"
  type: "Convolution"
  bottom: "pool1/norm1"
  top: "conv2/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "conv2/relu_3x3_reduce"
  type: "ReLU"
  bottom: "conv2/3x3_reduce"
  top: "conv2/3x3_reduce"
}

layer {
  name: "conv2/3x3"
  type: "Convolution"
  bottom: "conv2/3x3_reduce"
  top: "conv2/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 192
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "conv2/relu_3x3"
  type: "ReLU"
  bottom: "conv2/3x3"
  top: "conv2/3x3"
}
layer {
  name: "inception_4d/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4c/output"
  top: "inception_4d/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4d/5x5_reduce"
  top: "inception_4d/5x5_reduce"
}
layer {
  name: "inception_4d/5x5"
  type: "Convolution"
  bottom: "inception_4d/5x5_reduce"
  top: "inception_4d/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_5x5"
  type: "ReLU"
  bottom: "inception_4d/5x5"
  top: "inception_4d/5x5"
}
layer {
  name: "inception_4d/pool"
  type: "Pooling"
  bottom: "inception_4c/output"
  top: "inception_4d/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
```

```
  top: "conv2/3x3"
}

layer {
  name: "conv2/norm2"
  type: "LRN"
  bottom: "conv2/3x3"
  top: "conv2/norm2"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}

layer {
  name: "pool2/3x3_s2"
  type: "Pooling"
  bottom: "conv2/norm2"
  top: "pool2/3x3_s2"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}

layer {
  name: "inception_3a/1x1"
  type: "Convolution"
  bottom: "pool2/3x3_s2"
  top: "inception_3a/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "inception_3a/relu_1x1"
  type: "ReLU"
  bottom: "inception_3a/1x1"
  top: "inception_3a/1x1"
}

layer {
  name: "inception_3a/3x3_reduce"
  type: "Convolution"
  bottom: "pool2/3x3_s2"
  top: "inception_3a/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 1
    pad: 1
  }
}
layer {
  name: "inception_4d/pool_proj"
  type: "Convolution"
  bottom: "inception_4d/pool"
  top: "inception_4d/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4d/pool_proj"
  top: "inception_4d/pool_proj"
}
layer {
  name: "inception_4d/output"
  type: "Concat"
  bottom: "inception_4d/1x1"
  bottom: "inception_4d/3x3"
  bottom: "inception_4d/5x5"
  bottom: "inception_4d/pool_proj"
  top: "inception_4d/output"
}

layer {
  name: "inception_4e/1x1"
  type: "Convolution"
  bottom: "inception_4d/output"
  top: "inception_4e/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_1x1"
  type: "ReLU"
  bottom: "inception_4e/1x1"
  top: "inception_4e/1x1"
}
```

```
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "inception_3a/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_3a/3x3_reduce"
  top: "inception_3a/3x3_reduce"
}

layer {
  name: "inception_3a/3x3"
  type: "Convolution"
  bottom: "inception_3a/3x3_reduce"
  top: "inception_3a/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "inception_3a/relu_3x3"
  type: "ReLU"
  bottom: "inception_3a/3x3"
  top: "inception_3a/3x3"
}

layer {
  name: "inception_3a/5x5_reduce"
  type: "Convolution"
  bottom: "pool2/3x3_s2"
  top: "inception_3a/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 16
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
```

```
layer {
  name: "inception_4e/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4d/output"
  top: "inception_4e/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 160
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4e/3x3_reduce"
  top: "inception_4e/3x3_reduce"
}
layer {
  name: "inception_4e/3x3"
  type: "Convolution"
  bottom: "inception_4e/3x3_reduce"
  top: "inception_4e/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 320
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_3x3"
  type: "ReLU"
  bottom: "inception_4e/3x3"
  top: "inception_4e/3x3"
}
layer {
  name: "inception_4e/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4d/output"
  top: "inception_4e/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
```

```
  }
}
layer {
  name: "inception_3a/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_3a/5x5_reduce"
  top: "inception_3a/5x5_reduce"
}
layer {
  name: "inception_3a/5x5"
  type: "Convolution"
  bottom: "inception_3a/5x5_reduce"
  top: "inception_3a/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_3a/relu_5x5"
  type: "ReLU"
  bottom: "inception_3a/5x5"
  top: "inception_3a/5x5"
}

layer {
  name: "inception_3a/pool"
  type: "Pooling"
  bottom: "pool2/3x3_s2"
  top: "inception_3a/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}

layer {
  name: "inception_3a/pool_proj"
  type: "Convolution"
  bottom: "inception_3a/pool"
  top: "inception_3a/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
```

```
    }
  }
  convolution_param {
    num_output: 32
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4e/5x5_reduce"
  top: "inception_4e/5x5_reduce"
}
layer {
  name: "inception_4e/5x5"
  type: "Convolution"
  bottom: "inception_4e/5x5_reduce"
  top: "inception_4e/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_5x5"
  type: "ReLU"
  bottom: "inception_4e/5x5"
  top: "inception_4e/5x5"
}
layer {
  name: "inception_4e/pool"
  type: "Pooling"
  bottom: "inception_4d/output"
  top: "inception_4e/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4e/pool_proj"
  type: "Convolution"
  bottom: "inception_4e/pool"
  top: "inception_4e/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
```

```
      value: 0.2
    }
  }
}
layer {
  name: "inception_3a/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_3a/pool_proj"
  top: "inception_3a/pool_proj"
}

layer {
  name: "inception_3a/output"
  type: "Concat"
  bottom: "inception_3a/1x1"
  bottom: "inception_3a/3x3"
  bottom: "inception_3a/5x5"
  bottom: "inception_3a/pool_proj"
  top: "inception_3a/output"
}

layer {
  name: "inception_3b/1x1"
  type: "Convolution"
  bottom: "inception_3a/output"
  top: "inception_3b/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "inception_3b/relu_1x1"
  type: "ReLU"
  bottom: "inception_3b/1x1"
  top: "inception_3b/1x1"
}

layer {
  name: "inception_3b/3x3_reduce"
  type: "Convolution"
  bottom: "inception_3a/output"
  top: "inception_3b/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
```

```
    }
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4e/pool_proj"
  top: "inception_4e/pool_proj"
}
layer {
  name: "inception_4e/output"
  type: "Concat"
  bottom: "inception_4e/1x1"
  bottom: "inception_4e/3x3"
  bottom: "inception_4e/5x5"
  bottom: "inception_4e/pool_proj"
  top: "inception_4e/output"
}



layer {
  name: "inception_5a/1x1"
  type: "Convolution"
  bottom: "inception_4e/output"
  top: "inception_5a/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_1x1"
  type: "ReLU"
  bottom: "inception_5a/1x1"
  top: "inception_5a/1x1"
}

layer {
  name: "inception_5a/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4e/output"
  top: "inception_5a/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
```

```
      value: 0.2
    }
  }
}
layer {
  name: "inception_3b/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_3b/3x3_reduce"
  top: "inception_3b/3x3_reduce"
}
layer {
  name: "inception_3b/3x3"
  type: "Convolution"
  bottom: "inception_3b/3x3_reduce"
  top: "inception_3b/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 192
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_3b/relu_3x3"
  type: "ReLU"
  bottom: "inception_3b/3x3"
  top: "inception_3b/3x3"
}

layer {
  name: "inception_3b/5x5_reduce"
  type: "Convolution"
  bottom: "inception_3a/output"
  top: "inception_3b/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_3b/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_3b/5x5_reduce"
  top: "inception_3b/5x5_reduce"
}
layer {
  }
  convolution_param {
    num_output: 160
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_5a/3x3_reduce"
  top: "inception_5a/3x3_reduce"
}

layer {
  name: "inception_5a/3x3"
  type: "Convolution"
  bottom: "inception_5a/3x3_reduce"
  top: "inception_5a/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 320
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_3x3"
  type: "ReLU"
  bottom: "inception_5a/3x3"
  top: "inception_5a/3x3"
}
layer {
  name: "inception_5a/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4e/output"
  top: "inception_5a/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
```

```
  name: "inception_3b/5x5"
  type: "Convolution"
  bottom: "inception_3b/5x5_reduce"
  top: "inception_3b/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_3b/relu_5x5"
  type: "ReLU"
  bottom: "inception_3b/5x5"
  top: "inception_3b/5x5"
}

layer {
  name: "inception_3b/pool"
  type: "Pooling"
  bottom: "inception_3a/output"
  top: "inception_3b/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_3b/pool_proj"
  type: "Convolution"
  bottom: "inception_3b/pool"
  top: "inception_3b/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_3b/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_3b/pool_proj"
  top: "inception_3b/pool_proj"
}
    }
  }
}
layer {
  name: "inception_5a/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_5a/5x5_reduce"
  top: "inception_5a/5x5_reduce"
}
layer {
  name: "inception_5a/5x5"
  type: "Convolution"
  bottom: "inception_5a/5x5_reduce"
  top: "inception_5a/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_5x5"
  type: "ReLU"
  bottom: "inception_5a/5x5"
  top: "inception_5a/5x5"
}
layer {
  name: "inception_5a/pool"
  type: "Pooling"
  bottom: "inception_4e/output"
  top: "inception_5a/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_5a/pool_proj"
  type: "Convolution"
  bottom: "inception_5a/pool"
  top: "inception_5a/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
```

```
layer {
  name: "inception_3b/output"
  type: "Concat"
  bottom: "inception_3b/1x1"
  bottom: "inception_3b/3x3"
  bottom: "inception_3b/5x5"
  bottom: "inception_3b/pool_proj"
  top: "inception_3b/output"
}

layer {
  name: "pool3/3x3_s2"
  type: "Pooling"
  bottom: "inception_3b/output"
  top: "pool3/3x3_s2"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}

layer {
  name: "inception_4a/1x1"
  type: "Convolution"
  bottom: "pool3/3x3_s2"
  top: "inception_4a/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 192
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "inception_4a/relu_1x1"
  type: "ReLU"
  bottom: "inception_4a/1x1"
  top: "inception_4a/1x1"
}

layer {
  name: "inception_4a/3x3_reduce"
  type: "Convolution"
  bottom: "pool3/3x3_s2"
  top: "inception_4a/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
    }
  }
}
layer {
  name: "inception_5a/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_5a/pool_proj"
  top: "inception_5a/pool_proj"
}
layer {
  name: "inception_5a/output"
  type: "Concat"
  bottom: "inception_5a/1x1"
  bottom: "inception_5a/3x3"
  bottom: "inception_5a/5x5"
  bottom: "inception_5a/pool_proj"
  top: "inception_5a/output"
}

layer {
  name: "inception_5b/1x1"
  type: "Convolution"
  bottom: "inception_5a/output"
  top: "inception_5b/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 384
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5b/relu_1x1"
  type: "ReLU"
  bottom: "inception_5b/1x1"
  top: "inception_5b/1x1"
}
layer {
  name: "inception_5b/3x3_reduce"
  type: "Convolution"
  bottom: "inception_5a/output"
  top: "inception_5b/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 1
    decay_mult: 0
  }
  convolution_param {
    num_output: 192
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
```

```
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "inception_4a/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4a/3x3_reduce"
  top: "inception_4a/3x3_reduce"
}

layer {
  name: "inception_4a/3x3"
  type: "Convolution"
  bottom: "inception_4a/3x3_reduce"
  top: "inception_4a/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 208
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "inception_4a/relu_3x3"
  type: "ReLU"
  bottom: "inception_4a/3x3"
  top: "inception_4a/3x3"
}

layer {
  name: "inception_4a/5x5_reduce"
  type: "Convolution"
  bottom: "pool3/3x3_s2"
  top: "inception_4a/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 16
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4a/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4a/5x5_reduce"
  top: "inception_4a/5x5_reduce"
}
layer {
  name: "inception_4a/5x5"
  type: "Convolution"
  bottom: "inception_4a/5x5_reduce"
  top: "inception_4a/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 48
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4a/relu_5x5"
  type: "ReLU"
  bottom: "inception_4a/5x5"
  top: "inception_4a/5x5"
}
layer {
  name: "inception_4a/pool"
  type: "Pooling"
  bottom: "pool3/3x3_s2"
  top: "inception_4a/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4a/pool_proj"
  type: "Convolution"
  bottom: "inception_4a/pool"
  top: "inception_4a/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4a/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4a/pool_proj"
  top: "inception_4a/pool_proj"
}
layer {
  name: "inception_4a/output"
  type: "Concat"
  bottom: "inception_4a/1x1"
  bottom: "inception_4a/3x3"
  bottom: "inception_4a/5x5"
  bottom: "inception_4a/pool_proj"
  top: "inception_4a/output"
}
layer {
  name: "inception_4b/1x1"
  type: "Convolution"
  bottom: "inception_4a/output"
  top: "inception_4b/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 160
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_1x1"
  type: "ReLU"
  bottom: "inception_4b/1x1"
  top: "inception_4b/1x1"
}
layer {
  name: "inception_4b/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4a/output"
  top: "inception_4b/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 112
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4b/3x3_reduce"
  top: "inception_4b/3x3_reduce"
}
layer {
  name: "inception_4b/3x3"
  type: "Convolution"
  bottom: "inception_4b/3x3_reduce"
  top: "inception_4b/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 224
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_3x3"
  type: "ReLU"
  bottom: "inception_4b/3x3"
  top: "inception_4b/3x3"
}
layer {
  name: "inception_4b/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4a/output"
  top: "inception_4b/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 24
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4b/5x5_reduce"
  top: "inception_4b/5x5_reduce"
}
layer {
  name: "inception_4b/5x5"
  type: "Convolution"
  bottom: "inception_4b/5x5_reduce"
  top: "inception_4b/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_5x5"
  type: "ReLU"
  bottom: "inception_4b/5x5"
  top: "inception_4b/5x5"
}
layer {
  name: "inception_4b/pool"
  type: "Pooling"
  bottom: "inception_4a/output"
  top: "inception_4b/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4b/pool_proj"
  type: "Convolution"
  bottom: "inception_4b/pool"
  top: "inception_4b/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4b/pool_proj"
  top: "inception_4b/pool_proj"
}
layer {
  name: "inception_4b/output"
  type: "Concat"
  bottom: "inception_4b/1x1"
  bottom: "inception_4b/3x3"
  bottom: "inception_4b/5x5"
  bottom: "inception_4b/pool_proj"
  top: "inception_4b/output"
}
layer {
  name: "inception_4c/1x1"
  type: "Convolution"
  bottom: "inception_4b/output"
  top: "inception_4c/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4c/relu_1x1"
  type: "ReLU"
  bottom: "inception_4c/1x1"
  top: "inception_4c/1x1"
}
layer {
  name: "inception_4c/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4b/output"
  top: "inception_4c/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4c/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4c/3x3_reduce"
  top: "inception_4c/3x3_reduce"
}
layer {
  name: "inception_4c/3x3"
  type: "Convolution"
  bottom: "inception_4c/3x3_reduce"
  top: "inception_4c/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4c/relu_3x3"
  type: "ReLU"
  bottom: "inception_4c/3x3"
  top: "inception_4c/3x3"
}
layer {
  name: "inception_4c/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4b/output"
  top: "inception_4c/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 24
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4c/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4c/5x5_reduce"
  top: "inception_4c/5x5_reduce"
}
layer {
  name: "inception_4c/5x5"
  type: "Convolution"
  bottom: "inception_4c/5x5_reduce"
  top: "inception_4c/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4c/relu_5x5"
  type: "ReLU"
  bottom: "inception_4c/5x5"
  top: "inception_4c/5x5"
}
layer {
  name: "inception_4c/pool"
  type: "Pooling"
  bottom: "inception_4b/output"
  top: "inception_4c/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4c/pool_proj"
  type: "Convolution"
  bottom: "inception_4c/pool"
  top: "inception_4c/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4c/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4c/pool_proj"
  top: "inception_4c/pool_proj"
}
layer {
  name: "inception_4c/output"
  type: "Concat"
  bottom: "inception_4c/1x1"
  bottom: "inception_4c/3x3"
  bottom: "inception_4c/5x5"
  bottom: "inception_4c/pool_proj"
  top: "inception_4c/output"
}
layer {
  name: "inception_4d/1x1"
  type: "Convolution"
  bottom: "inception_4c/output"
  top: "inception_4d/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 112
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_1x1"
  type: "ReLU"
  bottom: "inception_4d/1x1"
  top: "inception_4d/1x1"
}
layer {
  name: "inception_4d/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4c/output"
  top: "inception_4d/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 144
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4d/3x3_reduce"
  top: "inception_4d/3x3_reduce"
}
layer {
  name: "inception_4d/3x3"
  type: "Convolution"
  bottom: "inception_4d/3x3_reduce"
  top: "inception_4d/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 288
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_3x3"
  type: "ReLU"
  bottom: "inception_4d/3x3"
  top: "inception_4d/3x3"
}
layer {
  name: "inception_4d/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4c/output"
  top: "inception_4d/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4d/5x5_reduce"
  top: "inception_4d/5x5_reduce"
}
layer {
  name: "inception_4d/5x5"
  type: "Convolution"
  bottom: "inception_4d/5x5_reduce"
  top: "inception_4d/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_5x5"
  type: "ReLU"
  bottom: "inception_4d/5x5"
  top: "inception_4d/5x5"
}
layer {
  name: "inception_4d/pool"
  type: "Pooling"
  bottom: "inception_4c/output"
  top: "inception_4d/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4d/pool_proj"
  type: "Convolution"
  bottom: "inception_4d/pool"
  top: "inception_4d/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4d/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4d/pool_proj"
  top: "inception_4d/pool_proj"
}
layer {
  name: "inception_4d/output"
  type: "Concat"
  bottom: "inception_4d/1x1"
  bottom: "inception_4d/3x3"
  bottom: "inception_4d/5x5"
  bottom: "inception_4d/pool_proj"
  top: "inception_4d/output"
}
layer {
  name: "inception_4e/1x1"
  type: "Convolution"
  bottom: "inception_4d/output"
  top: "inception_4e/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_1x1"
  type: "ReLU"
  bottom: "inception_4e/1x1"
  top: "inception_4e/1x1"
}
layer {
  name: "inception_4e/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4d/output"
  top: "inception_4e/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 160
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4e/3x3_reduce"
  top: "inception_4e/3x3_reduce"
}
layer {
  name: "inception_4e/3x3"
  type: "Convolution"
  bottom: "inception_4e/3x3_reduce"
  top: "inception_4e/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 320
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_3x3"
  type: "ReLU"
  bottom: "inception_4e/3x3"
  top: "inception_4e/3x3"
}
layer {
  name: "inception_4e/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4d/output"
  top: "inception_4e/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4e/5x5_reduce"
  top: "inception_4e/5x5_reduce"
}
layer {
  name: "inception_4e/5x5"
  type: "Convolution"
  bottom: "inception_4e/5x5_reduce"
  top: "inception_4e/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_5x5"
  type: "ReLU"
  bottom: "inception_4e/5x5"
  top: "inception_4e/5x5"
}
layer {
  name: "inception_4e/pool"
  type: "Pooling"
  bottom: "inception_4d/output"
  top: "inception_4e/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4e/pool_proj"
  type: "Convolution"
  bottom: "inception_4e/pool"
  top: "inception_4e/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4e/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4e/pool_proj"
  top: "inception_4e/pool_proj"
}
layer {
  name: "inception_4e/output"
  type: "Concat"
  bottom: "inception_4e/1x1"
  bottom: "inception_4e/3x3"
  bottom: "inception_4e/5x5"
  bottom: "inception_4e/pool_proj"
  top: "inception_4e/output"
}
layer {
  name: "pool4/3x3_s2"
  type: "Pooling"
  bottom: "inception_4e/output"
  top: "pool4/3x3_s2"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "inception_5a/1x1"
  type: "Convolution"
  bottom: "pool4/3x3_s2"
  top: "inception_5a/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_1x1"
  type: "ReLU"
  bottom: "inception_5a/1x1"
  top: "inception_5a/1x1"
}
layer {
  name: "inception_5a/3x3_reduce"
  type: "Convolution"
  bottom: "pool4/3x3_s2"
  top: "inception_5a/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 160
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_5a/3x3_reduce"
  top: "inception_5a/3x3_reduce"
}
layer {
  name: "inception_5a/3x3"
  type: "Convolution"
  bottom: "inception_5a/3x3_reduce"
  top: "inception_5a/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 320
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_3x3"
  type: "ReLU"
  bottom: "inception_5a/3x3"
  top: "inception_5a/3x3"
}
layer {
  name: "inception_5a/5x5_reduce"
  type: "Convolution"
  bottom: "pool4/3x3_s2"
  top: "inception_5a/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_5a/5x5_reduce"
  top: "inception_5a/5x5_reduce"
}
layer {
  name: "inception_5a/5x5"
  type: "Convolution"
  bottom: "inception_5a/5x5_reduce"
  top: "inception_5a/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_5x5"
  type: "ReLU"
  bottom: "inception_5a/5x5"
  top: "inception_5a/5x5"
}
layer {
  name: "inception_5a/pool"
  type: "Pooling"
  bottom: "pool4/3x3_s2"
  top: "inception_5a/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_5a/pool_proj"
  type: "Convolution"
  bottom: "inception_5a/pool"
  top: "inception_5a/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5a/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_5a/pool_proj"
  top: "inception_5a/pool_proj"
}
layer {
  name: "inception_5a/output"
  type: "Concat"
  bottom: "inception_5a/1x1"
  bottom: "inception_5a/3x3"
  bottom: "inception_5a/5x5"
  bottom: "inception_5a/pool_proj"
  top: "inception_5a/output"
}
layer {
  name: "inception_5b/1x1"
  type: "Convolution"
  bottom: "inception_5a/output"
  top: "inception_5b/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 384
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5b/relu_1x1"
  type: "ReLU"
  bottom: "inception_5b/1x1"
  top: "inception_5b/1x1"
}
layer {
  name: "inception_5b/3x3_reduce"
  type: "Convolution"
  bottom: "inception_5a/output"
  top: "inception_5b/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 192
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5b/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_5b/3x3_reduce"
  top: "inception_5b/3x3_reduce"
}
layer {
  name: "inception_5b/3x3"
  type: "Convolution"
  bottom: "inception_5b/3x3_reduce"
  top: "inception_5b/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5b/relu_3x3"
  type: "ReLU"
  bottom: "inception_5b/3x3"
  top: "inception_5b/3x3"
}
layer {
  name: "inception_5b/5x5_reduce"
  type: "Convolution"
  bottom: "inception_5a/output"
  top: "inception_5b/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 48
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_5b/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_5b/5x5_reduce"
  top: "inception_5b/5x5_reduce"
}
layer {
  name: "inception_5b/5x5"
  type: "Convolution"
  bottom: "inception_5b/5x5_reduce"
  top: "inception_5b/5x5"
  param {
```

```
  bottom: "inception_4a/5x5_reduce"
  top: "inception_4a/5x5_reduce"
}
layer {
  name: "inception_4a/5x5"
  type: "Convolution"
  bottom: "inception_4a/5x5_reduce"
  top: "inception_4a/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 48
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4a/relu_5x5"
  type: "ReLU"
  bottom: "inception_4a/5x5"
  top: "inception_4a/5x5"
}
layer {
  name: "inception_4a/pool"
  type: "Pooling"
  bottom: "pool3/3x3_s2"
  top: "inception_4a/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4a/pool_proj"
  type: "Convolution"
  bottom: "inception_4a/pool"
  top: "inception_4a/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4a/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4a/pool_proj"
  top: "inception_4a/pool_proj"
}
layer {
  name: "inception_4a/output"
  type: "Concat"
  bottom: "inception_4a/1x1"
  bottom: "inception_4a/3x3"
  bottom: "inception_4a/5x5"
  top: "inception_4a/output"
}
layer {
  name: "inception_4b/1x1"
  type: "Convolution"
  bottom: "inception_4a/output"
  top: "inception_4b/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_1x1"
  type: "ReLU"
  bottom: "inception_4b/1x1"
  top: "inception_4b/1x1"
}
layer {
  name: "inception_4b/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4a/output"
  top: "inception_4b/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4b/3x3_reduce"
  top: "inception_4b/3x3_reduce"
}
layer {
  name: "inception_4b/3x3"
  type: "Convolution"
  bottom: "inception_4b/3x3_reduce"
  top: "inception_4b/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 192
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_3x3"
  type: "ReLU"
  bottom: "inception_4b/3x3"
  top: "inception_4b/3x3"
}
layer {
  name: "inception_4b/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4a/output"
  top: "inception_4b/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4b/5x5_reduce"
  top: "inception_4b/5x5_reduce"
}
layer {
  name: "inception_4b/5x5"
  type: "Convolution"
  bottom: "inception_4b/5x5_reduce"
  top: "inception_4b/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_5x5"
  type: "ReLU"
  bottom: "inception_4b/5x5"
  top: "inception_4b/5x5"
}
layer {
  name: "inception_4b/pool"
  type: "Pooling"
  bottom: "inception_4a/output"
  top: "inception_4b/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4b/pool_proj"
  type: "Convolution"
  bottom: "inception_4b/pool"
  top: "inception_4b/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4b/pool_proj"
  top: "inception_4b/pool_proj"
}
layer {
  name: "inception_4b/output"
  type: "Concat"
  bottom: "inception_4b/1x1"
  bottom: "inception_4b/3x3"
  bottom: "inception_4b/5x5"
```

```
  bottom: "inception_4a/pool_proj"
  top: "inception_4a/pool_proj"
}
layer {
  name: "inception_4a/output"
  type: "Concat"
  bottom: "inception_4a/1x1"
  bottom: "inception_4a/3x3"
  bottom: "inception_4a/5x5"
  bottom: "inception_4a/pool_proj"
  top: "inception_4a/output"
}

layer {
  name: "inception_4b/1x1"
  type: "Convolution"
  bottom: "inception_4a/output"
  top: "inception_4b/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 160
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}

layer {
  name: "inception_4b/relu_1x1"
  type: "ReLU"
  bottom: "inception_4b/1x1"
  top: "inception_4b/1x1"
}
layer {
  name: "inception_4b/3x3_reduce"
  type: "Convolution"
  bottom: "inception_4a/output"
  top: "inception_4b/3x3_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 112
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.09
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception_4b/3x3_reduce"
  top: "inception_4b/3x3_reduce"
  bottom: "inception_5b/pool_proj"
  top: "inception_5b/output"
}
layer {
  name: "pool5/drop_s1"
  type: "Dropout"
  bottom: "inception_5b/output"
  top: "pool5/drop_s1"
  dropout_param {
    dropout_ratio: 0.4
  }
}
layer {
  name: "cvg/classifier"
  type: "Convolution"
  bottom: "pool5/drop_s1"
  top: "cvg/classifier"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 3
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.
    }
  }
}
layer {
  name: "coverage/sig"
  type: "Sigmoid"
  bottom: "cvg/classifier"
  top: "coverage"
}
layer {
  name: "bbox/regressor"
  type: "Convolution"
  bottom: "pool5/drop_s1"
  top: "bboxes"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 4
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.
    }
  }
}

################################################
######################
# End of convolutional network
################################################
######################
```

```
}
layer {
  name: "inception_4b/3x3"
  type: "Convolution"
  bottom: "inception_4b/3x3_reduce"
  top: "inception_4b/3x3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 224
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_3x3"
  type: "ReLU"
  bottom: "inception_4b/3x3"
  top: "inception_4b/3x3"
}
layer {
  name: "inception_4b/5x5_reduce"
  type: "Convolution"
  bottom: "inception_4a/output"
  top: "inception_4b/5x5_reduce"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 24
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.2
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_5x5_reduce"
  type: "ReLU"
  bottom: "inception_4b/5x5_reduce"
  top: "inception_4b/5x5_reduce"
}
layer {
  name: "inception_4b/5x5"
  type: "Convolution"
  bottom: "inception_4b/5x5_reduce"
  top: "inception_4b/5x5"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
```

```
# Convert bboxes
layer {
  name: "bbox_mask"
  type: "Eltwise"
  bottom: "bboxes"
  bottom: "coverage-block"
  top: "bboxes-masked"
  eltwise_param {
    operation: PROD
  }
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}
layer {
  name: "bbox-norm"
  type: "Eltwise"
  bottom: "bboxes-masked"
  bottom: "size-block"
  top: "bboxes-masked-norm"
  eltwise_param {
    operation: PROD
  }
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}
layer {
  name: "bbox-obj-norm"
  type: "Eltwise"
  bottom: "bboxes-masked-norm"
  bottom: "obj-block"
  top: "bboxes-obj-masked-norm"
  eltwise_param {
    operation: PROD
  }
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}

# Loss layers
layer {
  name: "bbox_loss"
  type: "L1Loss"
  bottom: "bboxes-obj-masked-norm"
  bottom: "bbox-obj-label-norm"
  top: "loss_bbox"
  loss_weight: 2
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}
layer {
  name: "coverage_loss"
  type: "EuclideanLoss"
  bottom: "coverage"
  bottom: "coverage-label"
  top: "loss_coverage"
  include { phase: TRAIN }
  include { phase: TEST stage: "val" }
}

# Cluster bboxes
layer {
    type: 'Python'
    name: 'cluster'
    bottom: 'coverage'
    bottom: 'bboxes'
    top: 'bbox-list-class0'
    top: 'bbox-list-class1'
    top: 'bbox-list-class2'
    python_param {
        module:
'caffe.layers.detectnet.clustering'
        layer: 'ClusterDetections'
        param_str : '640, 640, 16, 0.6, 3,
0.02, 22, 3'
    }
    include: { phase: TEST }
}
```

```
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_5x5"
  type: "ReLU"
  bottom: "inception_4b/5x5"
  top: "inception_4b/5x5"
}
layer {
  name: "inception_4b/pool"
  type: "Pooling"
  bottom: "inception_4a/output"
  top: "inception_4b/pool"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 1
    pad: 1
  }
}
layer {
  name: "inception_4b/pool_proj"
  type: "Convolution"
  bottom: "inception_4b/pool"
  top: "inception_4b/pool_proj"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.1
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "inception_4b/relu_pool_proj"
  type: "ReLU"
  bottom: "inception_4b/pool_proj"
  top: "inception_4b/pool_proj"
}
layer {
  name: "inception_4b/output"
  type: "Concat"
  bottom: "inception_4b/1x1"
  bottom: "inception_4b/3x3"
  bottom: "inception_4b/5x5"
  bottom: "inception_4b/pool_proj"
  top: "inception_4b/output"
}
layer {
  name: "inception_4c/1x1"
```

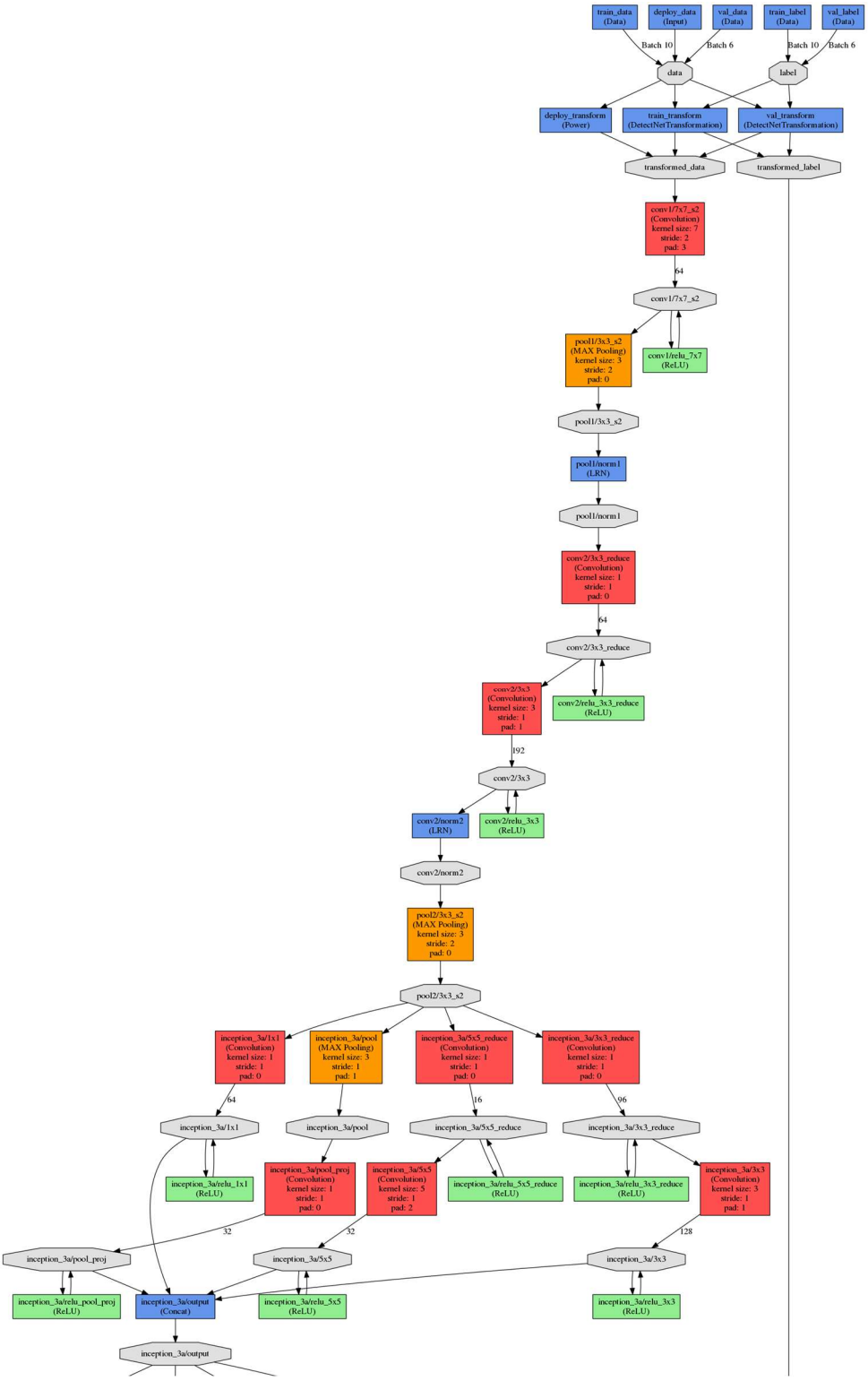
```
# Calculate mean average precision
layer {
  type: 'Python'
  name: 'cluster_gt'
  bottom: 'coverage-label'
  bottom: 'bbox-label'
  top: 'bbox-list-label-class0'
  top: 'bbox-list-label-class1'
  top: 'bbox-list-label-class2'
  python_param {
      module:
'caffe.layers.detectnet.clustering'
      layer: 'ClusterGroundtruth'
      param_str : '640, 640, 16, 3'
  }
  include: { phase: TEST stage: "val" }
}
layer {
    type: 'Python'
    name: 'score-class0'
    bottom: 'bbox-list-label-class0'
    bottom: 'bbox-list-class0'
    top: 'bbox-list-scored-class0'
    python_param {
        module:
'caffe.layers.detectnet.mean_ap'
        layer: 'ScoreDetections'
    }
    include: { phase: TEST stage: "val" }
}
layer {
    type: 'Python'
    name: 'mAP-class0'
    bottom: 'bbox-list-scored-class0'
    top: 'mAP-class0'
    top: 'precision-class0'
    top: 'recall-class0'
    python_param {
        module:
'caffe.layers.detectnet.mean_ap'
        layer: 'mAP'
        param_str : '640, 640, 16'
    }
    include: { phase: TEST stage: "val" }
}

layer {
    type: 'Python'
    name: 'score-class1'
    bottom: 'bbox-list-label-class1'
    bottom: 'bbox-list-class1'
    top: 'bbox-list-scored-class1'
    python_param {
        module:
'caffe.layers.detectnet.mean_ap'
        layer: 'ScoreDetections'
    }
    include: { phase: TEST stage: "val" }
}
layer {
    type: 'Python'
    name: 'mAP-class1'
    bottom: 'bbox-list-scored-class1'
    top: 'mAP-class1'
    top: 'precision-class1'
    top: 'recall-class1'
    python_param {
        module:
'caffe.layers.detectnet.mean_ap'
        layer: 'mAP'
        param_str : '640, 640, 16'
    }
    include: { phase: TEST stage: "val" }
}

layer {
```

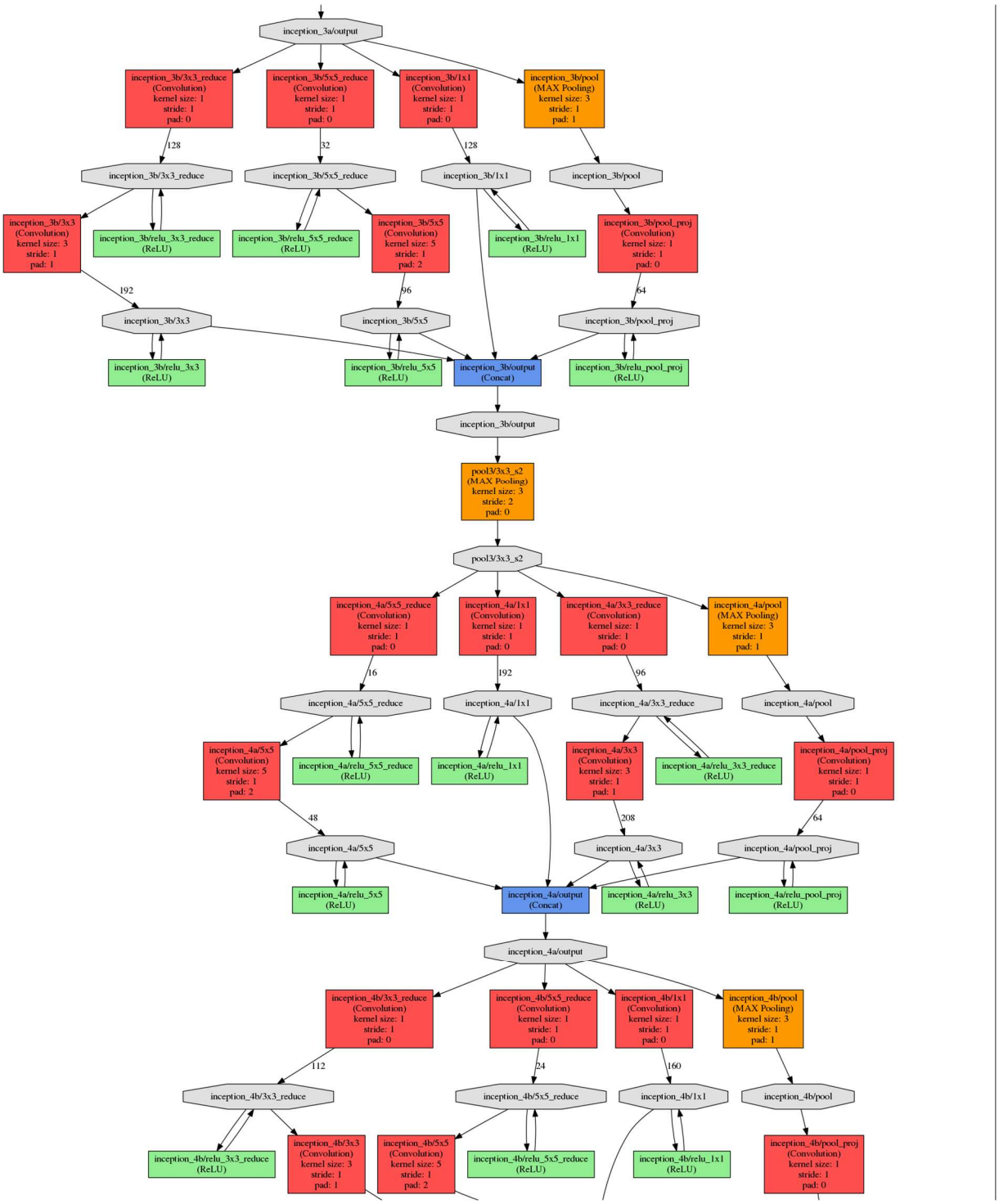
```
  type: "Convolution"
  bottom: "inception_4b/output"
  top: "inception_4c/1x1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 128
    kernel_size: 1
    weight_filler {
      type: "xavier"
      std: 0.03
    }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
```

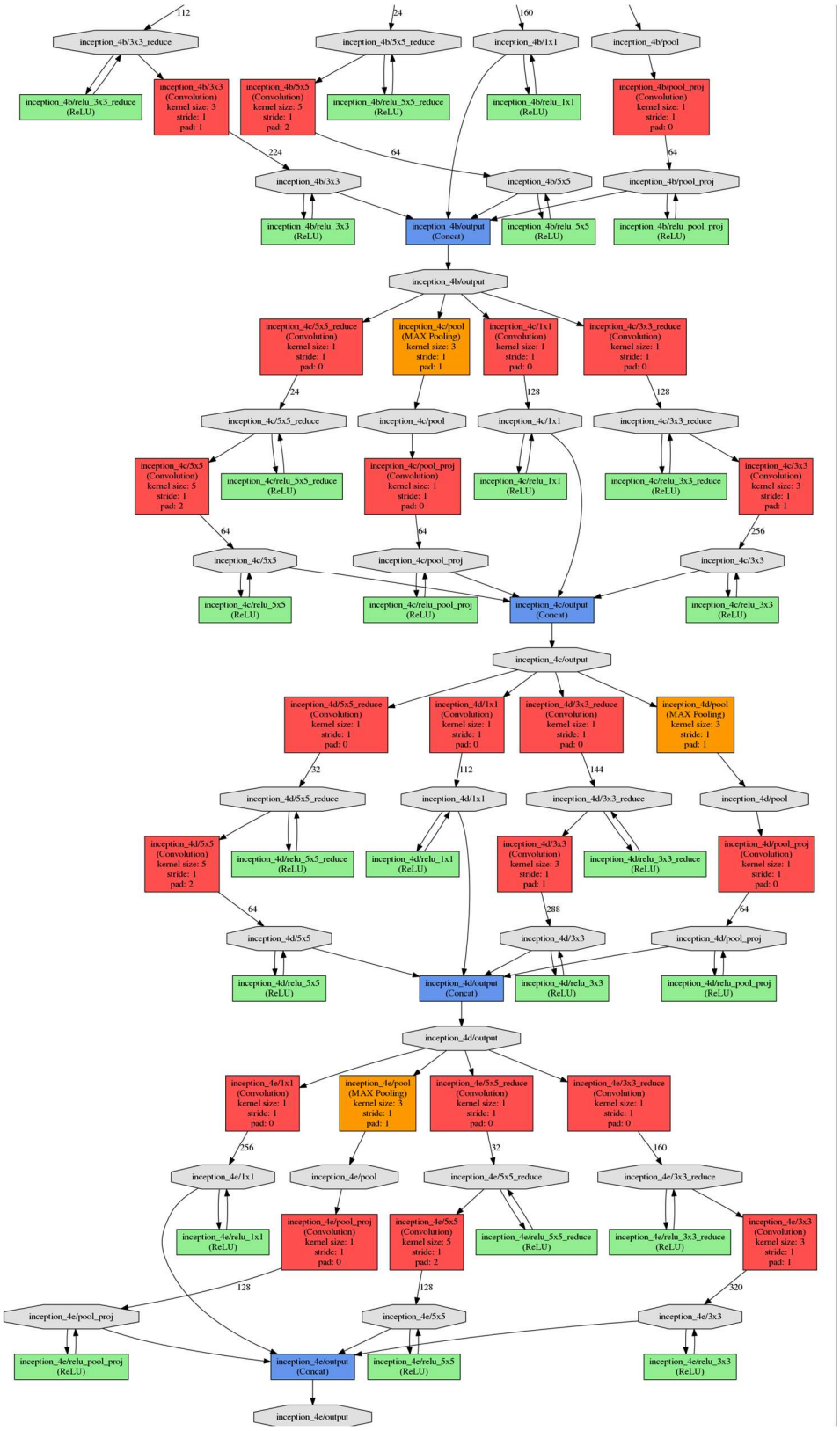
```
    type: 'Python'
    name: 'score-class2'
    bottom: 'bbox-list-label-class2'
    bottom: 'bbox-list-class2'
    top: 'bbox-list-scored-class2'
    python_param {
        module:
'caffe.layers.detectnet.mean_ap'
        layer: 'ScoreDetections'
    }
    include: { phase: TEST stage: "val" }
}
layer {
    type: 'Python'
    name: 'mAP-class2'
    bottom: 'bbox-list-scored-class2'
    top: 'mAP-class2'
    top: 'precision-class2'
    top: 'recall-class2'
    python_param {
        module:
'caffe.layers.detectnet.mean_ap'
        layer: 'mAP'
        param_str : '640, 640, 16'
    }
    include: { phase: TEST stage: "val" }
}
```

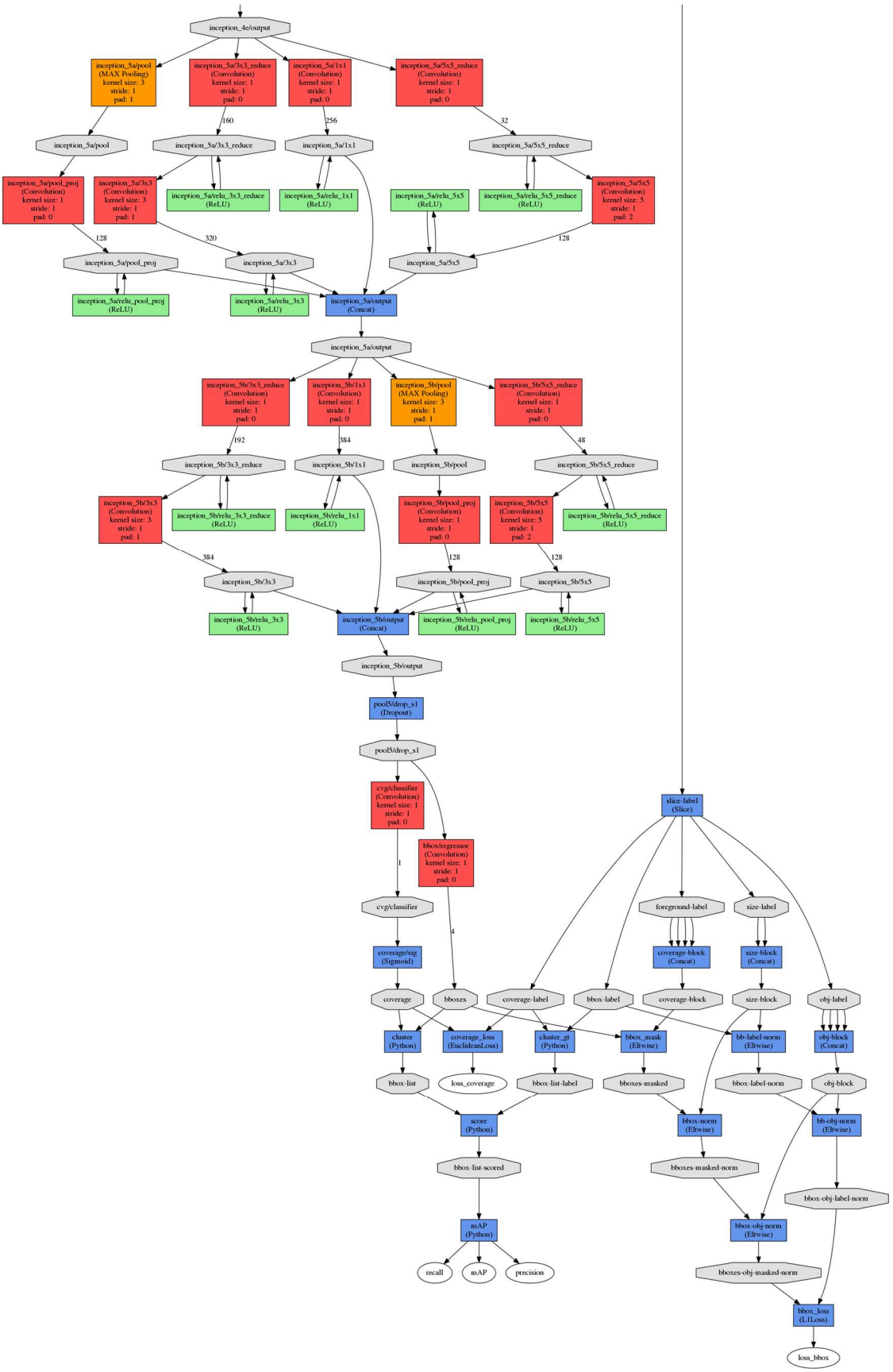
8.8. Appendix H: DetecNet Architecture Visualization

The DetectNet architecture visualization is split into four images for better visibility.



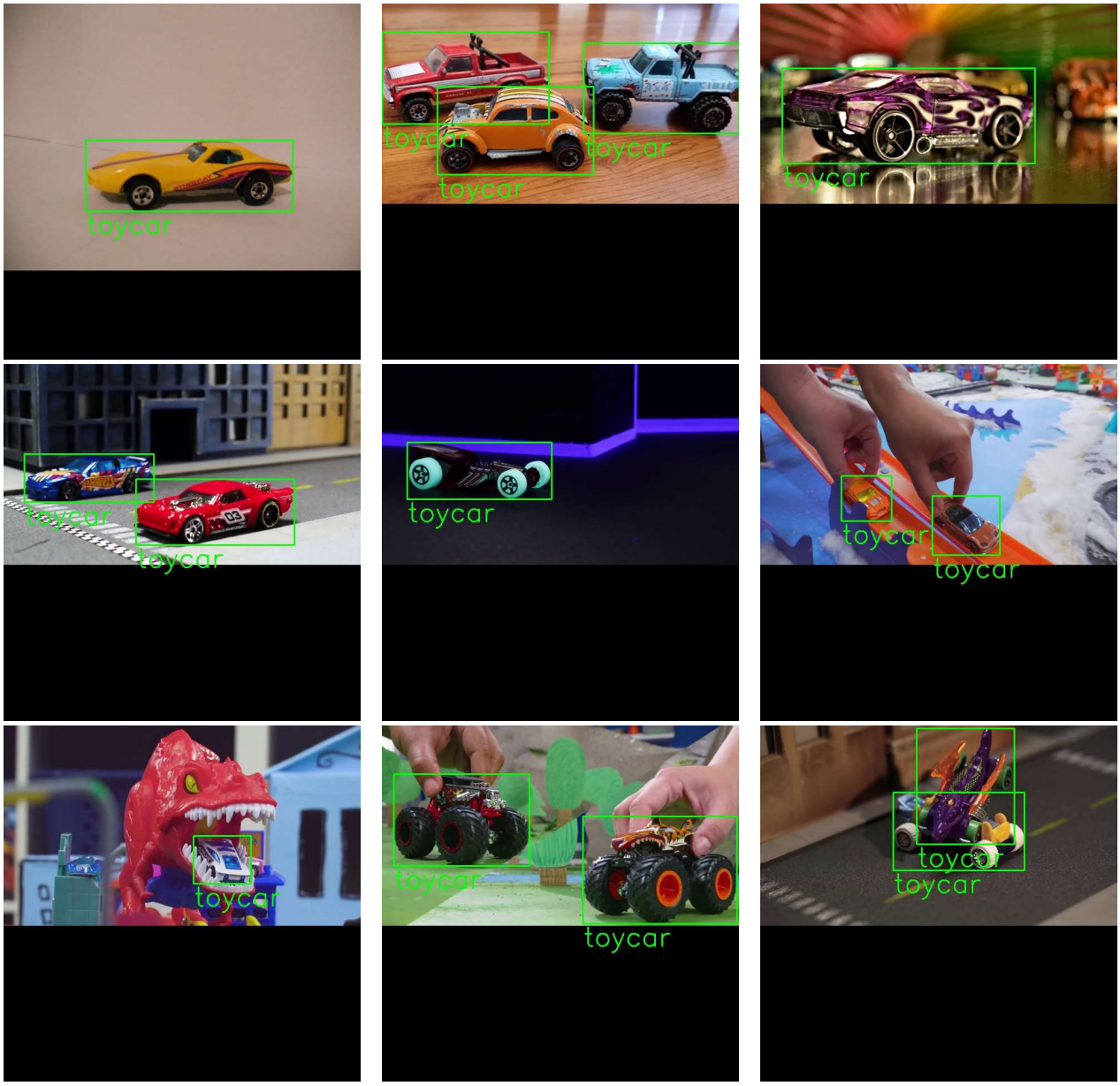

inception_3a/output
inception_3b/3x3_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_3b/5x5_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_3b/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
inception_3b/pool (MAX Pooling) kernel size: 3 stride: 1 pad: 1
128
32
128
inception_3b/3x3_reduce
inception_3b/5x5_reduce
inception_3b/1x1
inception_3b/pool
inception_3b/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
inception_3b/relu_3x3_reduce (ReLU)
inception_3b/relu_5x5_reduce (ReLU)
inception_3b/5x5 (Convolution) kernel size: 5 stride: 1 pad: 2
inception_3b/relu_1x1 (ReLU)
inception_3b/pool_proj (Convolution) kernel size: 1 stride: 1 pad: 0
192
96
64
inception_3b/3x3
inception_3b/5x5
inception_3b/pool_proj
inception_3b/relu_3x3 (ReLU)
inception_3b/relu_5x5 (ReLU)
inception_3b/output (Concat)
inception_3b/relu_pool_proj (ReLU)
inception_3b/output
pool3/3x3_s2 (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool3/3x3_s2
inception_4a/5x5_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4a/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4a/3x3_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4a/pool (MAX Pooling) kernel size: 3 stride: 1 pad: 1
16
192
96
inception_4a/5x5_reduce
inception_4a/1x1
inception_4a/3x3_reduce
inception_4a/pool
inception_4a/5x5 (Convolution) kernel size: 5 stride: 1 pad: 2
inception_4a/relu_5x5_reduce (ReLU)
inception_4a/relu_1x1 (ReLU)
inception_4a/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
inception_4a/relu_3x3_reduce (ReLU)
inception_4a/pool_proj (Convolution) kernel size: 1 stride: 1 pad: 0
48
208
64
inception_4a/5x5
inception_4a/3x3
inception_4a/pool_proj
inception_4a/relu_5x5 (ReLU)
inception_4a/output (Concat)
inception_4a/relu_3x3 (ReLU)
inception_4a/relu_pool_proj (ReLU)
inception_4a/output
inception_4b/3x3_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4b/5x5_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4b/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4b/pool (MAX Pooling) kernel size: 3 stride: 1 pad: 1
112
24
160
inception_4b/3x3_reduce
inception_4b/5x5_reduce
inception_4b/1x1
inception_4b/pool
inception_4b/relu_3x3_reduce (ReLU)
inception_4b/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
inception_4b/5x5 (Convolution) kernel size: 5 stride: 1 pad: 2
inception_4b/relu_5x5_reduce (ReLU)
inception_4b/relu_1x1 (ReLU)
inception_4b/pool_proj (Convolution) kernel size: 1 stride: 1 pad: 0

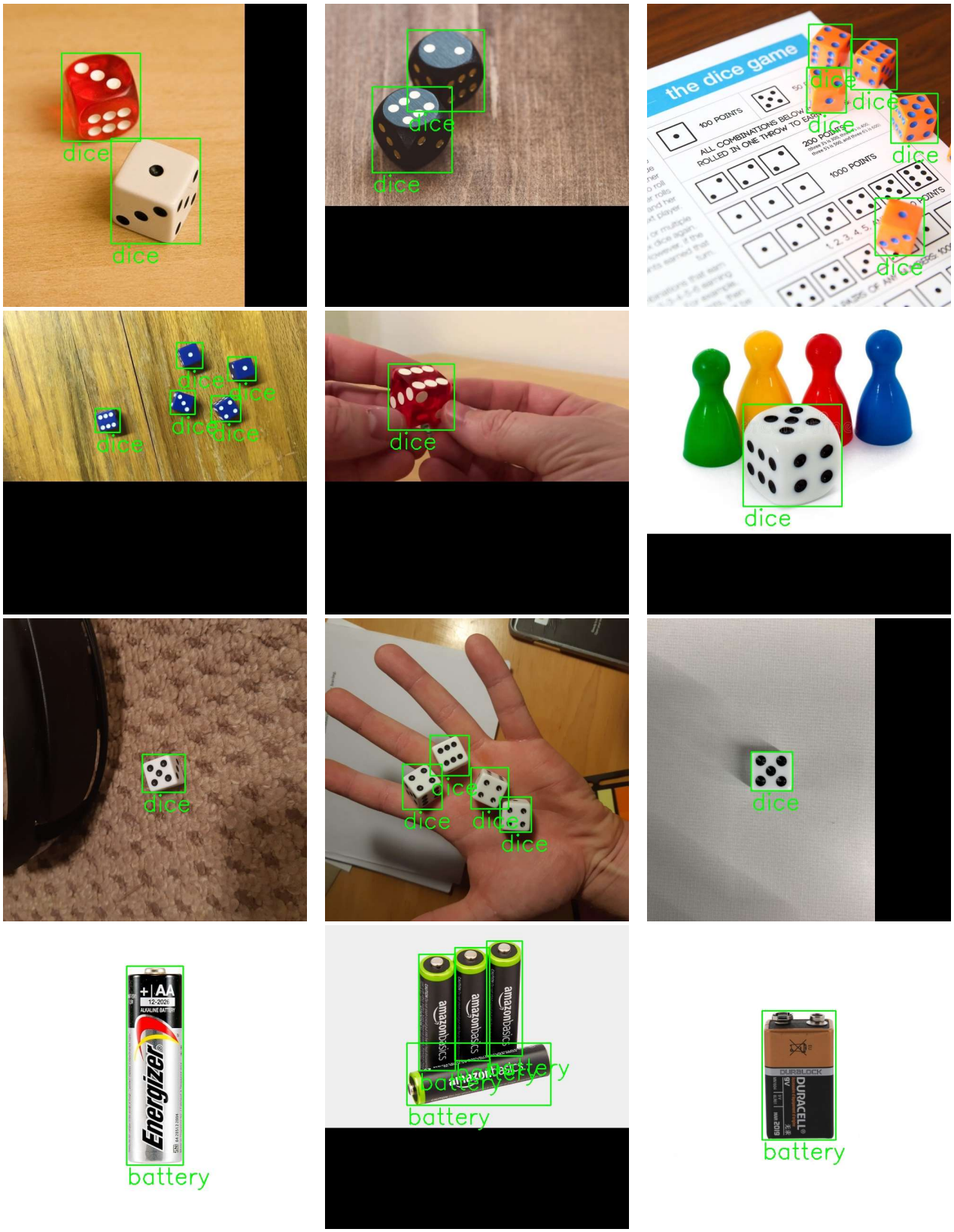

112
24
160
inception_4b/3x3_reduce
inception_4b/5x5_reduce
inception_4b/1x1
inception_4b/pool
inception_4b/relu_3x3_reduce (ReLU)
inception_4b/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
inception_4b/5x5 (Convolution) kernel size: 5 stride: 1 pad: 2
inception_4b/relu_5x5_reduce (ReLU)
inception_4b/relu_1x1 (ReLU)
inception_4b/pool_proj (Convolution) kernel size: 1 stride: 1 pad: 0
224
64
64
inception_4b/3x3
inception_4b/5x5
inception_4b/pool_proj
inception_4b/relu_3x3 (ReLU)
inception_4b/output (Concat)
inception_4b/relu_5x5 (ReLU)
inception_4b/relu_pool_proj (ReLU)
inception_4b/output
inception_4c/5x5_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4c/pool (MAX Pooling) kernel size: 3 stride: 1 pad: 1
inception_4c/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4c/3x3_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
24
128
128
inception_4c/5x5_reduce
inception_4c/pool
inception_4c/1x1
inception_4c/3x3_reduce
inception_4c/5x5 (Convolution) kernel size: 5 stride: 1 pad: 2
inception_4c/relu_5x5_reduce (ReLU)
inception_4c/pool_proj (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4c/relu_1x1 (ReLU)
inception_4c/relu_3x3_reduce (ReLU)
inception_4c/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
64
64
256
inception_4c/5x5
inception_4c/pool_proj
inception_4c/3x3
inception_4c/relu_5x5 (ReLU)
inception_4c/relu_pool_proj (ReLU)
inception_4c/output (Concat)
inception_4c/relu_3x3 (ReLU)
inception_4c/output
inception_4d/5x5_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4d/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4d/3x3_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4d/pool (MAX Pooling) kernel size: 3 stride: 1 pad: 1
32
112
144
inception_4d/5x5_reduce
inception_4d/1x1
inception_4d/3x3_reduce
inception_4d/pool
inception_4d/5x5 (Convolution) kernel size: 5 stride: 1 pad: 2
inception_4d/relu_5x5_reduce (ReLU)
inception_4d/relu_1x1 (ReLU)
inception_4d/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
inception_4d/relu_3x3_reduce (ReLU)
inception_4d/pool_proj (Convolution) kernel size: 1 stride: 1 pad: 0
64
288
64
inception_4d/5x5
inception_4d/3x3
inception_4d/pool_proj
inception_4d/relu_5x5 (ReLU)
inception_4d/output (Concat)
inception_4d/relu_3x3 (ReLU)
inception_4d/relu_pool_proj (ReLU)
inception_4d/output
inception_4e/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4e/pool (MAX Pooling) kernel size: 3 stride: 1 pad: 1
inception_4e/5x5_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4e/3x3_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
256
32
160
inception_4e/1x1
inception_4e/pool
inception_4e/5x5_reduce
inception_4e/3x3_reduce
inception_4e/relu_1x1 (ReLU)
inception_4e/pool_proj (Convolution) kernel size: 1 stride: 1 pad: 0
inception_4e/5x5 (Convolution) kernel size: 5 stride: 1 pad: 2
inception_4e/relu_5x5_reduce (ReLU)
inception_4e/relu_3x3_reduce (ReLU)
inception_4e/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
128
128
320
inception_4e/pool_proj
inception_4e/5x5
inception_4e/3x3
inception_4e/relu_pool_proj (ReLU)
inception_4e/output (Concat)
inception_4e/relu_5x5 (ReLU)
inception_4e/relu_3x3 (ReLU)
inception_4e/output

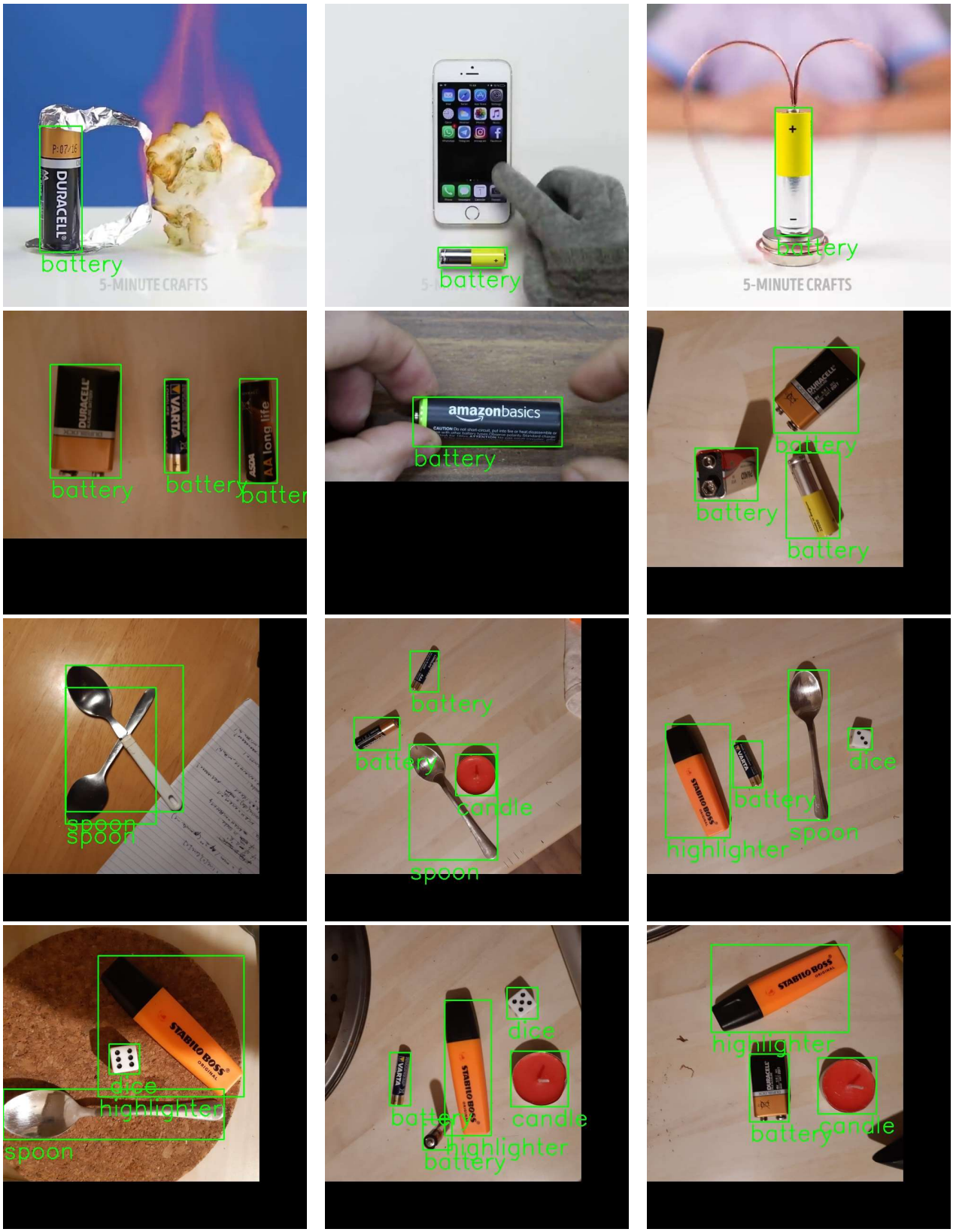

inception_4e/output
inception_5a/pool (MAX Pooling) kernel size: 3 stride: 1 pad: 1
inception_5a/3x3_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_5a/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
inception_5a/5x5_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
160
256
32
inception_5a/pool
inception_5a/3x3_reduce
inception_5a/1x1
inception_5a/5x5_reduce
inception_5a/pool_proj (Convolution) kernel size: 1 stride: 1 pad: 0
inception_5a/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
inception_5a/relu_3x3_reduce (ReLU)
inception_5a/relu_1x1 (ReLU)
inception_5a/relu_5x5 (ReLU)
inception_5a/relu_5x5_reduce (ReLU)
inception_5a/5x5 (Convolution) kernel size: 5 stride: 1 pad: 2
128
320
128
inception_5a/pool_proj
inception_5a/3x3
inception_5a/5x5
inception_5a/relu_pool_proj (ReLU)
inception_5a/relu_3x3 (ReLU)
inception_5a/output (Concat)
inception_5a/output
inception_5b/3x3_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_5b/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
inception_5b/pool (MAX Pooling) kernel size: 3 stride: 1 pad: 1
inception_5b/5x5_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
192
384
48
inception_5b/3x3_reduce
inception_5b/1x1
inception_5b/pool
inception_5b/5x5_reduce
inception_5b/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
inception_5b/relu_3x3_reduce (ReLU)
inception_5b/relu_1x1 (ReLU)
inception_5b/pool_proj (Convolution) kernel size: 1 stride: 1 pad: 0
inception_5b/5x5 (Convolution) kernel size: 5 stride: 1 pad: 2
inception_5b/relu_5x5_reduce (ReLU)
384
128
128
inception_5b/3x3
inception_5b/pool_proj
inception_5b/5x5
inception_5b/relu_3x3 (ReLU)
inception_5b/output (Concat)
inception_5b/relu_pool_proj (ReLU)
inception_5b/relu_5x5 (ReLU)
inception_5b/output
pool5/drop_s1 (Dropout)
pool5/drop_s1
cvg/classifier (Convolution) kernel size: 1 stride: 1 pad: 0
bbox/regressor (Convolution) kernel size: 1 stride: 1 pad: 0
1
4
cvg/classifier
coverage/sig (Sigmoid)
coverage
bboxes
cluster (Python)
coverage_loss (EuclideanLoss)
bbox-list
loss_coverage
score (Python)
bbox-list-scored
mAP (Python)
recall
mAP
precision
slice-label (Slice)
foreground-label
size-label
coverage-block (Concat)
size-block (Concat)
coverage-label
bbox-label
coverage-block
size-block
obj-label
cluster_gt (Python)
bbox_mask (Eltwise)
bb-label-norm (Eltwise)
obj-block (Concat)
bboxes-masked
bbox-label-norm
obj-block
bbox-norm (Eltwise)
bb-obj-norm (Eltwise)
bboxes-masked-norm
bbox-obj-label-norm
bbox-obj-norm (Eltwise)
bboxes-obj-masked-norm
bbox_loss (L1Loss)
loss_bbox

8.9. Appendix I: Examples of Annotated Images

This appendix includes examples of the labeled object detection dataset used in the project. The images were selected at random. The image dataset serves the non-profit educational purpose of this project and does not harm the copyright owner's ability to profit from their original work. Therefore, the YouTube video frames used in the project are assumed to be following YouTube's and fair use policy.




dice
dice
dice
dice
the dice game
100 POINTS
ALL COMBINATIONS BELOW ROLLED IN ONE THROW TO EARN
200 POINTS
1000 POINTS
dice
dice
dice
dice
dice
dice
dice
dice
dice
dice
dice
dice
dice
dice
dice
dice
dice
dice
battery
battery
battery


battery
5-MINUTE CRAFTS
battery
battery
5-MINUTE CRAFTS
battery
battery
battery
amazonbasics
battery
battery
battery
battery
spoon
spoon
battery
battery
candle
spoon
dice
battery
spoon
highlighter
dice
highlighter
spoon
dice
candle
highlighter
battery
highlighter
battery
candle

8.10. Appendix J: Data Analysis and Visualization Python Script

This Python script was used to analyze and visualize the object detection dataset.

```
# -*- coding: utf-8 -*-
# NOTE:
# This script has been written for Mark Csizmadia's final year BEng Electronic
# Engineering project, and is discuseed below.
#
# This script analyzes and visualizes the bounding box sizes in the dataset.

# Import packages.
import numpy as np
import os
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 22})
import argparse
import pandas as pd
import cv2

# Construct the argument parser and parse the arguments.
parser = argparse.ArgumentParser(
        description="Visualise the bounding box sizes.",
        formatter_class=argparse.RawTextHelpFormatter)

parser.add_argument("--path_to_images",
                    type=str,
                    required=True,
                    help="Path to directoy of all images.")

parser.add_argument("--path_to_annotation",
                    type=str,
                    required=True,
                    help="Path to directoy of all annotation.")

parser.add_argument("--path_to_bboxed_images",
                    type=str,
                    required=True,
                    help="Path to directoy to save all bboxed images.")

parser.add_argument("--path_to_save_viz",
                    type=str,
                    required=True,
                    help="Path to directoy to save all bboxed images.")

if __name__ == "__main__":

    # Retrieve arguments.
    opt, argv = parser.parse_known_args()

    # Dict for creating a df later.
    bbox_dict = {"object_type":[], "width":[], "height":[], "area":[], "img":[], "bbox":[]}

    # Counters.
    bbox_counter = 0
    img_counter = 0

    dice_counter = 0
    toycar_counter = 0
    battery_counter = 0
    spoon_counter = 0
    candle_counter = 0
    highlighter_counter = 0

    # Iterate over the files in the directory opt.dir.
    for dir_name, subdir_list, file_list in os.walk(opt.path_to_annotation):
        print("[INFO] Reading annotation data from: {0}\n".format(dir_name))
        for idx, filename in enumerate(file_list):
            # Get file, if file does not have txt extension, continue for loop.
            path_to_file = os.path.join(dir_name, filename)

            _, file_extension = os.path.splitext(path_to_file)
            fileID = filename.split(".")[0]
            path_to_image = os.path.join(opt.path_to_images, fileID + ".jpg")
```

```
            path_to_bboxed_image = os.path.join(opt.path_to_bboxed_images, fileID + ".jpg")

            if file_extension == ".txt":
                print("[INFO] Processing: {0}\n".format(filename))
            else:
                continue

            # If found an annotation file (txt extension), open.
            file_context_manager = open(path_to_file, "r", encoding="Latin-1")

            # Read the lines in the annotation file.
            lines = file_context_manager.readlines()

            # Clean the lines from whitespaces.
            lines = [line.rstrip().split(" ") for line in lines]

            # If the annotation file is faulty (i.e.: no objects),
            # raise exception.
            object_type = None
            if len(lines) < 1:
                raise Exception


            # Iterate over the annotation for each object in the current
            # annotation file.
            bbox_list = []
            for line in lines:
                object_type = line[0]

                # If incorrect object type, rasie Exception.
                if object_type not in ["dice", "toycar", "battery", "spoon", "candle",
"highlighter"]:
                    raise Exception("Invalid object in {0}.".format(filename))

                # Counters.
                if object_type == "dice":
                    dice_counter = dice_counter + 1
                if object_type == "toycar":
                    toycar_counter = toycar_counter + 1
                if object_type == "battery":
                    battery_counter = battery_counter + 1
                if object_type == "spoon":
                    spoon_counter = spoon_counter + 1
                if object_type == "candle":
                    candle_counter = candle_counter + 1
                if object_type == "highlighter":
                    highlighter_counter = highlighter_counter + 1

                # Get the bbox vertex coordinates.
                top_left = (float(line[4]), float(line[5]))
                top_right = (float(line[6]), float(line[5]))
                bottom_right = (float(line[6]), float(line[7]))
                bottom_left = (float(line[4]), float(line[7]))

                # Get the properties of the bbox.
                width = top_right[0] - top_left[0]
                height = bottom_left[1] - top_left[1]
                area = width * height


                # Save the properties of the bbox.
                bbox_dict["object_type"].append(object_type)
                bbox_dict["width"].append(width)
                bbox_dict["height"].append(height)
                bbox_dict["area"].append(area)
                bbox_dict["img"].append(filename)
                bbox_dict["bbox"].append([top_left, bottom_right])

                bbox_list.append([object_type, top_left, bottom_right])

                bbox_counter = bbox_counter + 1

            img_np = cv2.imread(path_to_image)
            for bbox in bbox_list:
                ot = bbox[0]
                tl = bbox[1]
                br = bbox[2]
```

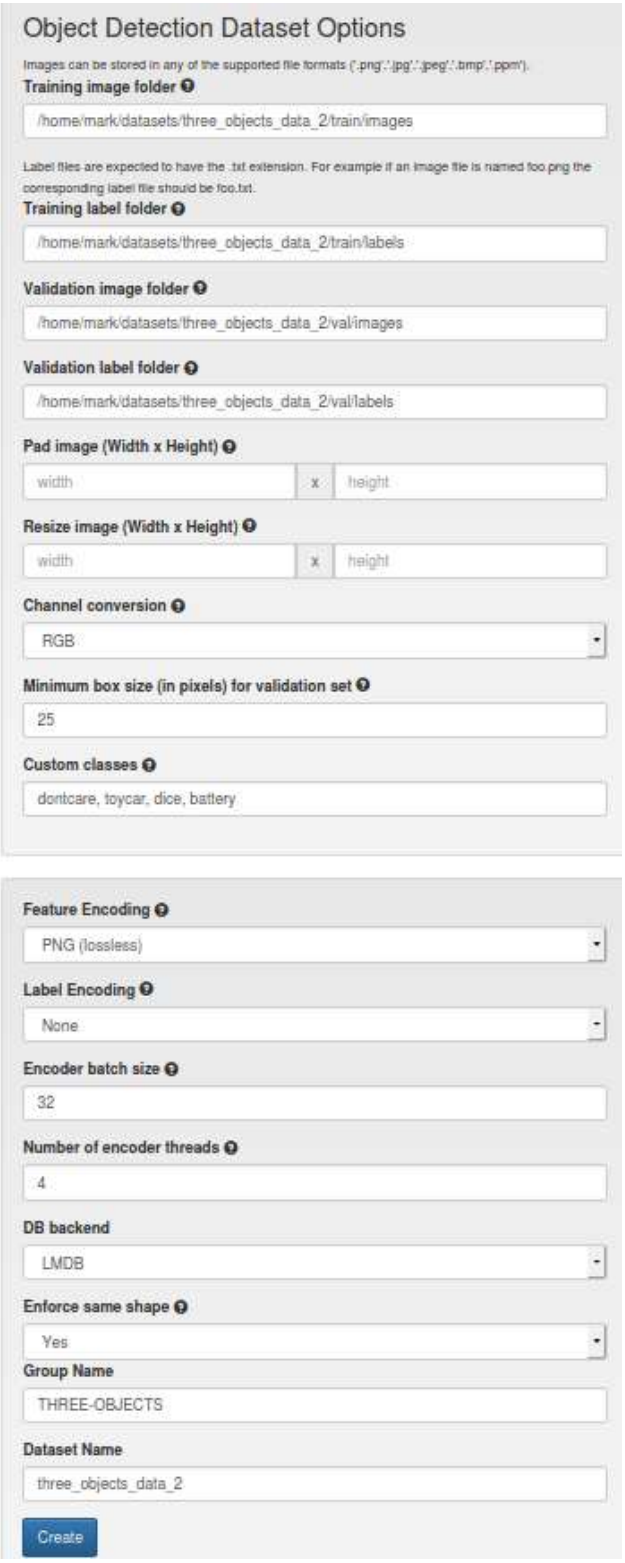
```
        cv2.rectangle(img_np,(int(tl[0]),int(tl[1])),
                      (int(br[0]),int(br[1])),
                      (0,255,0),2)
        cv2.putText(img_np,ot,(int(tl[0])+1,int(br[1])+32),0,1.2,(0,255,0),2)
        cv2.imwrite(path_to_bboxed_image, img_np)


    img_counter = img_counter + 1

    # Close the current annotation file.
    file_context_manager.close()

# Create dataframe.
df = pd.DataFrame(data=bbox_dict)

# Create Boolean indexers.
dice_cond = df["object_type"] == "dice"
toycar_cond = df["object_type"] == "toycar"
battery_cond = df["object_type"] == "battery"
spoon_cond = df["object_type"] == "spoon"
candle_cond = df["object_type"] == "candle"
highlighter_cond = df["object_type"] == "highlighter"
others_cond = np.logical_or(np.logical_or(spoon_cond, candle_cond),
                            highlighter_cond)

# Create figure.
fig = plt.figure(figsize=(25, 17), dpi= 80)
grid = plt.GridSpec(4, 4, hspace=0.5, wspace=0.2)

# Create axes.
ax_main = fig.add_subplot(grid[:-1, :-1])
ax_right = fig.add_subplot(grid[:-1, -1], xticklabels=[], yticklabels=[])
ax_bottom = fig.add_subplot(grid[-1, 0:-1], xticklabels=[], yticklabels=[])

# Plot stuff.
# Plot on the main axis.
ax_main.scatter(df[dice_cond]["width"],
                df[dice_cond]["height"],
                marker='o',color="b", label="dice")

ax_main.scatter(df[toycar_cond]["width"],
                df[toycar_cond]["height"],
                marker='o',color="g", label="toycar")

ax_main.scatter(df[battery_cond]["width"],
                df[battery_cond]["height"],
                marker='o',color="r", label="battery")

ax_main.scatter(df[others_cond]["width"],
                df[others_cond]["height"],
                marker='o',color="y", label="others")

# Plot undesired zones.
# I.e.: w and  < 40 pixel or w and h > 400 pixel
x1, y1 = [40, 40], [0, 40]
x2, y2 = [0, 40], [40, 40]
ax_main.plot(x1, y1, "--", color="k")
ax_main.plot(x2, y2, "--", color="k")

x3, y3 = [400, 400], [0, 400]
x4, y4 = [0, 400], [400, 400]
ax_main.plot(x3, y3, "--", color="k")
ax_main.plot(x4, y4, "--", color="k")

# Plot on the histograms.
ax_bottom.hist(df["width"], bins=100, histtype='stepfilled',
               orientation='vertical', color='grey')
ax_bottom.invert_yaxis()
ax_bottom.set_title("Width frequency distribution", y=-0.25)

ax_right.hist(df["height"], bins=100, histtype='stepfilled',
              orientation='horizontal', color='grey')
ax_right.set_title("Height frequency distribution")

# Visualisation cosmetics.
# Set ticks on x and y axis.
ax_main.set_xticks(
```

```
        ticks=[i for i in range(0, int(df["width"].max()+20), 40)])
ax_main.set_yticks(
        ticks=[i for i in range(0, int(df["height"].max()+20), 40)])

# Set the grid.
ax_main.grid(alpha=0.3, color='k', linestyle='-',
             linewidth=0.5, which="both")

ax_main.legend(loc="upper right")

ax_main.set_title("Bounding box sizes in pixel")
ax_main.set_xlabel('Bounding box width [pixel]')
ax_main.set_ylabel('Bounding box height [pixel]')

# Caclulate stuff from df.
w_below_40 = (df["width"] < 40).sum()
h_below_40 = (df["height"] < 40).sum()
a_below_40x40 = (df["area"] < 40**2).sum()

h_above_400 = (400 < df["height"]).sum()
w_above_400 = (400 < df["width"]).sum()
a_above_400x400 = (400**2 < df["area"]).sum()

dice_img = df[dice_cond]["img"].nunique()
toycar_img = df[toycar_cond]["img"].nunique()
battery_img = df[battery_cond]["img"].nunique()
spoon_img = df[spoon_cond]["img"].nunique()
candle_img = df[candle_cond]["img"].nunique()
highlighter_img = df[highlighter_cond]["img"].nunique()

# Assertion checks.
assert toycar_counter + dice_counter + \
    battery_counter + spoon_counter + \
    candle_counter + highlighter_counter == bbox_counter

# Print stuff.
print("[INFO] Width and height below 40: {0}/{1}, {2}/{3}\n".format(
        w_below_40, bbox_counter, h_below_40, bbox_counter))
print("[INFO] Width and height above 400: {0}/{1}, {2}/{3}\n".format(
        w_above_400, bbox_counter, h_above_400, bbox_counter))
print("[INFO] Area below 40x40 and above "
      "400x400: {0}/{1}, {2}/{3}\n".format(a_below_40x40, bbox_counter,
                a_above_400x400, bbox_counter))
print("[INFO] Average bbox per image: {0}\n".format(
        bbox_counter/img_counter))
print("[INFO] Toycar bbox: {0}, Dice bbox: {1}, Battery bbox: {2}, "
      "Spoon bbox: {3}, Candle bbox: {4}, Highlighter bbox: {5},"
      "All bbox: {6}\n".format(toycar_counter, dice_counter,
                battery_counter, spoon_counter, candle_counter,
                highlighter_counter, bbox_counter))
print("[INFO] Toycar in: {0}, Dice in: {1}, Battery in: {2}, "
      "Spoon in: {3}, Candle in: {4}, Highlighter in: {5}, "
      "All images: {6}\n".format(toycar_img, dice_img, battery_img,
                  spoon_img, candle_img, highlighter_img,
                  img_counter))

# Save and plot the figure.
#plt.show()

mean_std_width_all = \
    "Width\nMean={0:.0f}\nStd={1:.0f}".format(df["width"].mean(), df["width"].std())

ax_main.text(0.92, 0.20, mean_std_width_all, horizontalalignment='center',
             verticalalignment='center', transform=ax_main.transAxes, fontsize=18)

mean_std_height_all = \
    "Height\nMean={0:.0f}\nStd={1:.0f}".format(df["height"].mean(), df["height"].std())

ax_main.text(0.92, 0.10, mean_std_height_all, horizontalalignment='center',
             verticalalignment='center', transform=ax_main.transAxes, fontsize=18)



mean_height = \
    "Mean\ndice={0:.0f}\ntoycar={1:.0f}\nbattery={2:.0f}\nothers={3:.0f}".format(
            df[dice_cond]["height"].mean(),
            df[toycar_cond]["height"].mean(),
```

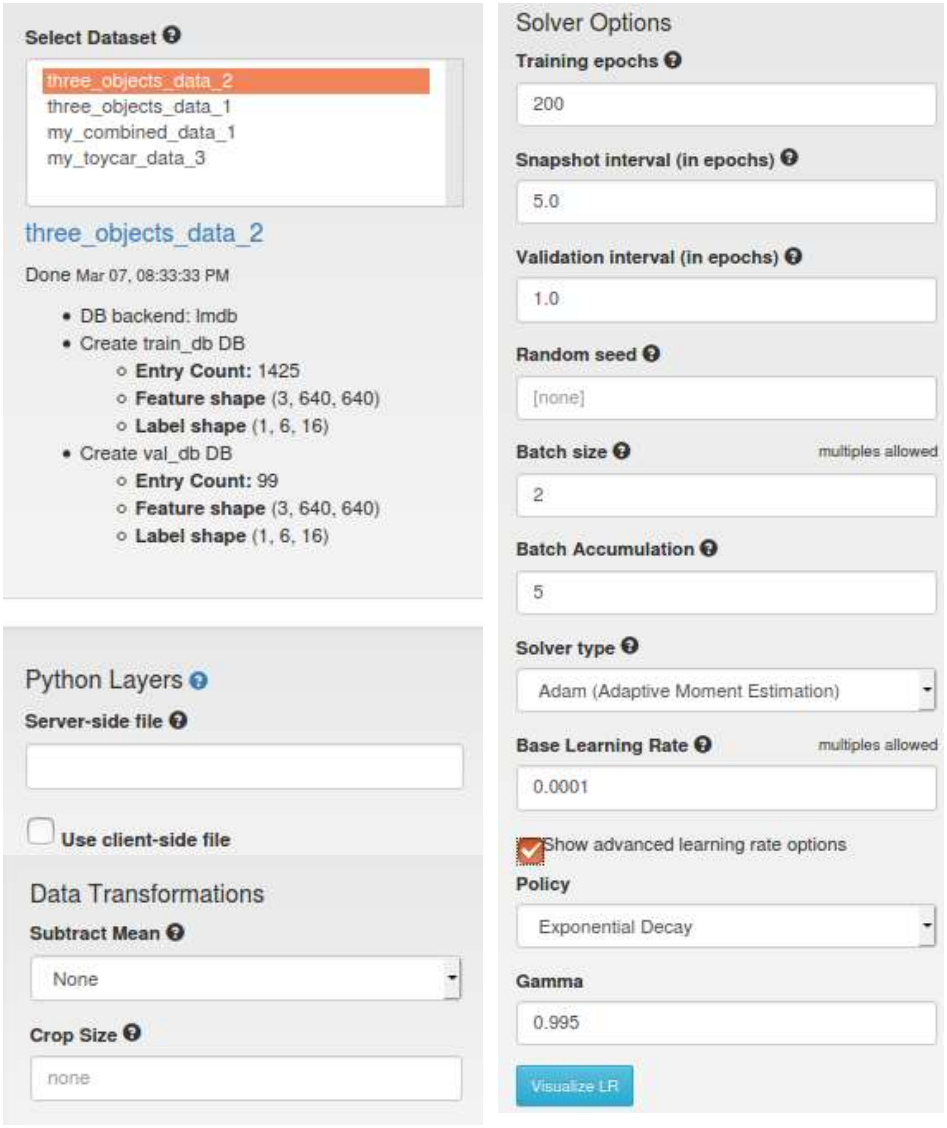
```
            df[battery_cond]["height"].mean(),
            df[others_cond]["height"].mean())

ax_right.text(0.78, 0.90, mean_height, horizontalalignment='center',
     verticalalignment='center', transform=ax_right.transAxes, fontsize=18)

std_height = \
 "Std\ndice={0:.0f}\ntoycar={1:.0f}\nbattery={2:.0f}\nothers={3:.0f}".format(
            df[dice_cond]["height"].std(),
            df[toycar_cond]["height"].std(),
            df[battery_cond]["height"].std(),
            df[others_cond]["height"].std())

ax_right.text(0.78, 0.75, std_height, horizontalalignment='center',
     verticalalignment='center', transform=ax_right.transAxes, fontsize=18)

mean_width = \
    "Mean\ndice={0:.0f}\ntoycar={1:.0f}\nbattery={2:.0f}\nothers={3:.0f}".format(
            df[dice_cond]["width"].mean(),
            df[toycar_cond]["width"].mean(),
            df[battery_cond]["width"].mean(),
            df[others_cond]["width"].mean())

ax_bottom.text(0.80, 0.45, mean_width, horizontalalignment='center',
     verticalalignment='center', transform=ax_bottom.transAxes, fontsize=18)

std_width = \
 "Std\ndice={0:.0f}\ntoycar={1:.0f}\nbattery={2:.0f}\nothers={3:.0f}".format(
            df[dice_cond]["width"].std(),
            df[toycar_cond]["width"].std(),
            df[battery_cond]["width"].std(),
            df[others_cond]["width"].std())

ax_bottom.text(0.92, 0.45, std_width, horizontalalignment='center',
     verticalalignment='center', transform=ax_bottom.transAxes, fontsize=18)

fig.savefig(
        os.path.join(opt.path_to_save_viz,
                     'Bounding box sizes.png'))


print("[INFO] Plotting stuff.\n")

plt.rcParams.update({'font.size': 18})

# All objects per image
fig_1 = plt.figure(figsize=(6, 6), dpi= 80)

ax_1 = fig_1.add_subplot(111)
ax_1.grid(alpha=0.3, color='k', linestyle='-',
             linewidth=0.5, which="both")

bins = [1,2,3,4,5,6]

bbox_per_image_df = df.set_index(["img", "area", "height", "width"]).count(level="img")

ax_1.hist(np.array(bbox_per_image_df["object_type"]), bins=bins,
          histtype="bar", align="left", rwidth=0.9, color="grey",
          density=False)
ax_1.set_xticks(bins[:-1])
ax_1.set_title("Frequency distribution of objects per image")
ax_1.set_xlabel("Number of objects per image")
ax_1.set_ylabel("Number of images")

fig_1.savefig(
        os.path.join(opt.path_to_save_viz,
                     'Frequency distribution of objects in the images.png'))

# per category per image
fig_2 = plt.figure(figsize=(12, 12), dpi= 80)

ax_00 = fig_2.add_subplot(221)
ax_00.grid(alpha=0.3, color='k', linestyle='-',
             linewidth=0.5, which="both")

bins = [1,2,3,4,5,6]
```

```
bbox_per_image_dice_df = \
    df[dice_cond].set_index(["img", "area", "height", "width"]).count(level="img")

ax_00.hist(np.array(bbox_per_image_dice_df["object_type"]), bins=bins,
          histtype="bar", align="left", rwidth=0.9, color="b",
          density=False)
ax_00.set_xticks(bins[:-1])
ax_00.set_title("Frequency distribution of \ndices per image")
ax_00.set_xlabel("Number of dices per image")
ax_00.set_ylabel("Number of images")

ax_01 = fig_2.add_subplot(222)
ax_01.grid(alpha=0.3, color='k', linestyle='-',
             linewidth=0.5, which="both")

bins = [1,2,3,4,5,6]

bbox_per_image_toycar_df = \
    df[toycar_cond].set_index(["img", "area", "height", "width"]).count(level="img")

ax_01.hist(np.array(bbox_per_image_toycar_df["object_type"]), bins=bins,
          histtype="bar", align="left", rwidth=0.9, color="g",
          density=False)
ax_01.set_xticks(bins[:-1])
ax_01.set_title("Frequency distribution of \ntoy cars per image")
ax_01.set_xlabel("Number of toy cars per image")
ax_01.set_ylabel("Number of images")

ax_10 = fig_2.add_subplot(223)
ax_10.grid(alpha=0.3, color='k', linestyle='-',
             linewidth=0.5, which="both")

bins = [1,2,3,4,5,6]

bbox_per_image_battery_df = \
    df[battery_cond].set_index(["img", "area", "height", "width"]).count(level="img")

ax_10.hist(np.array(bbox_per_image_battery_df["object_type"]), bins=bins,
          histtype="bar", align="left", rwidth=0.9, color="r",
          density=False)
ax_10.set_xticks(bins[:-1])
ax_10.set_title("Frequency distribution of \nbatteries per image")
ax_10.set_xlabel("Number of batteries per image")
ax_10.set_ylabel("Number of images")

ax_11 = fig_2.add_subplot(224)
ax_11.grid(alpha=0.3, color='k', linestyle='-',
             linewidth=0.5, which="both")

bins = [1,2,3,4,5,6]

bbox_per_image_others_df = \
    df[others_cond].set_index(["img", "area", "height", "width"]).count(level="img")

ax_11.hist(np.array(bbox_per_image_others_df["object_type"]), bins=bins,
          histtype="bar", align="left", rwidth=0.9, color="y",
          density=False)
ax_11.set_xticks(bins[:-1])
ax_11.set_title("Frequency distribution of \nother objects per image")
ax_11.set_xlabel("Number of other objects per image")
ax_11.set_ylabel("Number of images")

fig_2.subplots_adjust(left=None, bottom=None,
                      right=None, top=None,
                      wspace=0.4, hspace=0.4)

fig_2.savefig(
        os.path.join(opt.path_to_save_viz,
                     'Frequency distribution of objects in the images per category.png'))

plt.show()
```

8.11. Appendix K: Training and Validation Setup of the Final Model in DIGITS

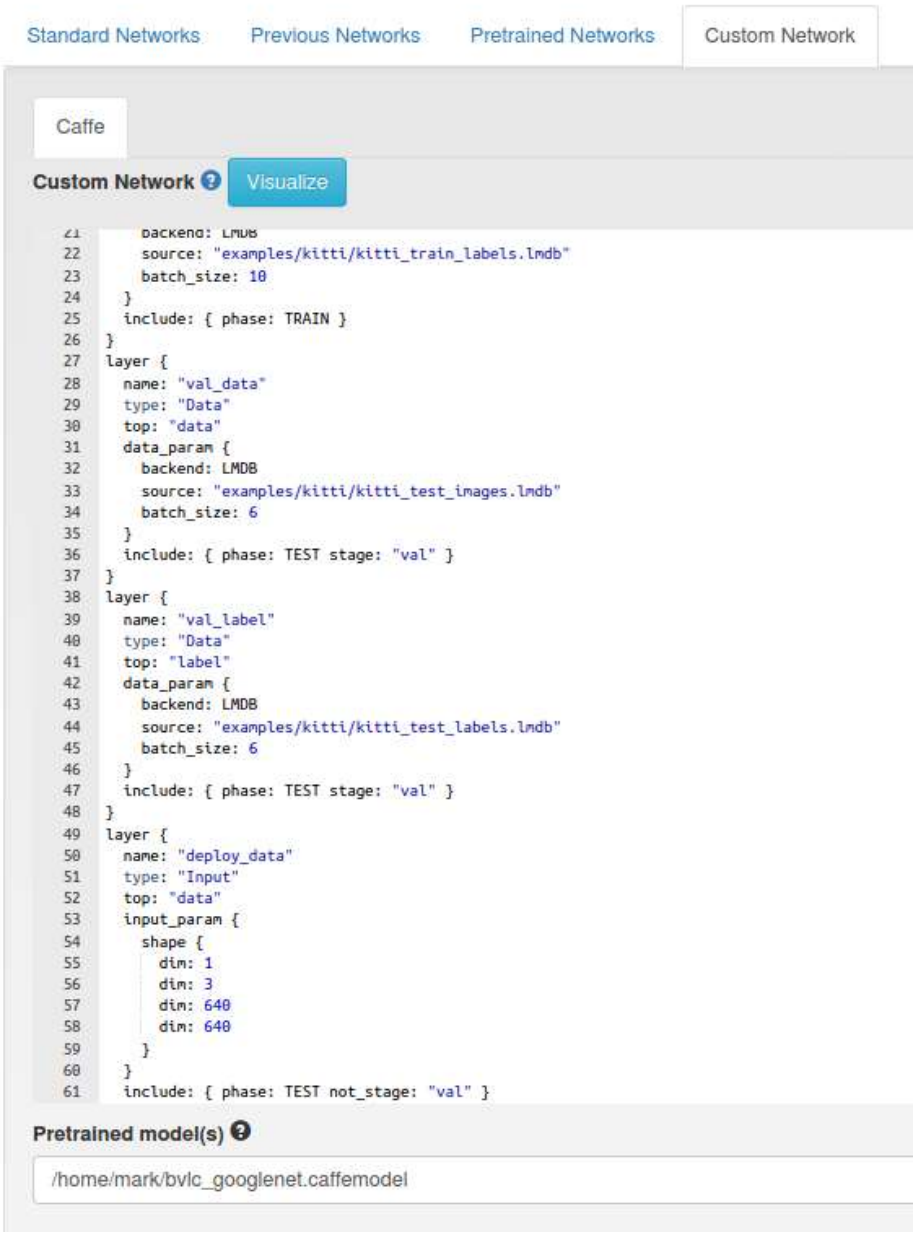


This appendix describes the training and validation setup of the final object detection model within DIGITS. Firstly, the annotated dataset is imported and an *Object Detection Dataset* is created within DIGITS as shown in the figure on the left. The paths to the training and validation images and labels are specified in the *Training Image folder* and the *Validation Image folder* fields. The paths to the corresponding training and validation image annotation files are specified in the *Training label folder* and the *Validation label folder*. The *Pad Imag*e and *Resize Image* fields are left blank, as the images are already cleaned and pre-processed as discussed section 3.3. The *Channel conversion* parameter specifies that the images are in RGB form. The *Minimum box size (in pixels) for validation set* field specifies the minimum bounding box size for the bounding box thresholding and clustering functions during validation. As shown in Figure 10, the majority of the bounding box sizes are above 40 x 40 pixels, which is why this field is left as default. In the *Custom classes* field, the listed object classes have direct correspondence with the object types in the object annotation files as described in section 3.3. The rest of the fields are left as default and the DIGITS dataset is given a name in the *Dataset Name* field.



Once the training and the validation databases are loaded into DIGITS, a new object detection model is created as shown in the figure above. The object detection dataset created before is declared as the training and validation dataset in the *Select Dataset* subwindow. The Python Layers subwindow remains intact as in the project no additional layers were added to the original DetectNet model. The *Data Transformations* subwindow remains untouched as well. However, the *Subtract Mean* field could be used to normalize the image data – a common technique in machine learning to speed up learning. In the Solver Options subwindow, the learnable parameter optimization algorithm is defined. The number of *Training epochs* is set to 200. The *Snapshot interval (in epochs)* specifies the rate at which DIGITS saves the trained model parameters into a snapshot file. These files can be transferred onto the Jetson Nano for inference as described in section 3.4.1. In the project, the snapshot files were saved only every fifth epoch to spare disk space. The *Validation interval (in epochs)* specifies the repetition of model validation. In the project, the

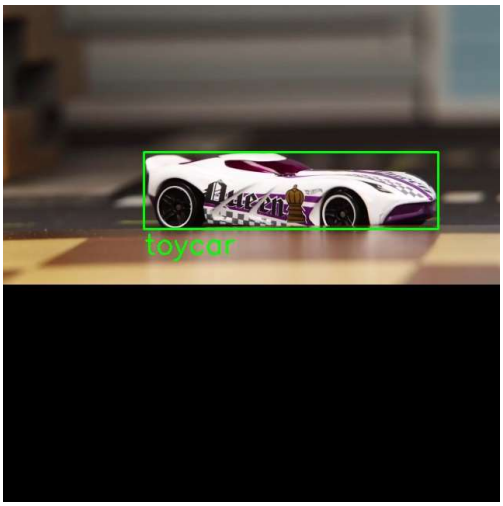
trained model was validated after every epoch to efficiently anticipate under-fitting or over-fitting. Since CNN-based object detection models are likely to learn different convolutional features, the *Random seed* parameter may assist in reproducing training experiments. In the project, the random seed was no used. The *Batch size* and the *Batch Accumulation* parameters are restricted by the host machine's GPU Compute Capability and memory. As mentioned in section 3.2, the GPU of the host machine has a Compute Capability of 5.0 and a memory of 4 GB. These parameters correspond to the maximum capability to process two training images in parallel at the same time and to accumulate gradients over a maximum of five images for mini-batch gradient descent based optimizers. In the project, the Adam optimizer is selected in the *Solver type* field, which is discussed in section 2.1.8. As a result of the hyperparameter search described in section 3.4.2, the initial learning rate is set to 10^{-4} in the *Base Learning Rate* field. The base learning rate was decreased using an exponential learning rate decay policy with a 0.995 decay parameter, which is specified in the *Policy* and *Gamma* fields.



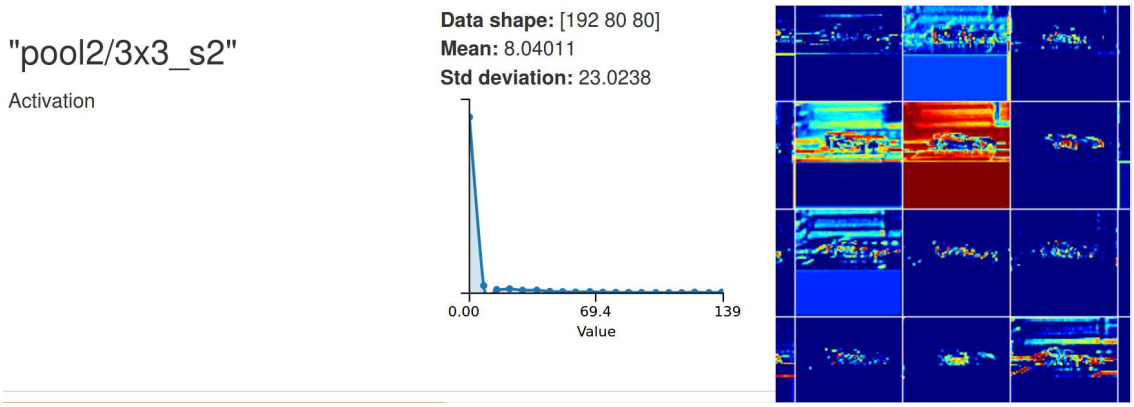
Once the *Solver Type* field is filled out, the network architecture of the object detection model is defined. In the project, the DetectNet object detection model is used which is discussed in section 3.1. The DetectNet description file for training is copied into the *Custom Network* field. In the project, the BVLC ImageNet pre-trained GoogLeNet parameters [29] are used for fine-tuning the model in the custom object detection task. The path to the downloaded pre-trained parameters is specified in the *Pretrained model(s)*. In the project, only one GPU is used therefore it is automatically assigned to the model training and the validation. However, If there were multiple GPUs available for the host machine, some or all of them may be assigned for distributed training.

8.12. Appendix L: Visualization of Layers During Inference within DIGITS

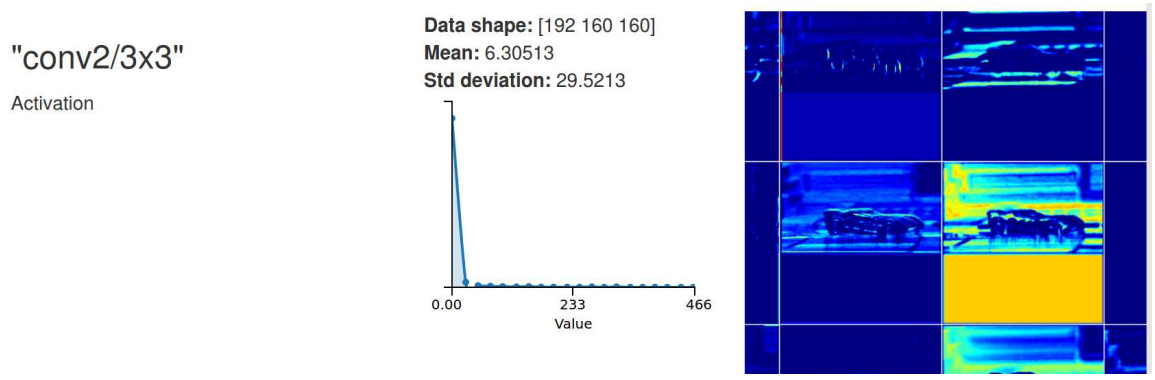
Some of the layers mentioned in section 3.1 are visualized for network inference on the test image shown below. The visualizations are provided within DIGITS.



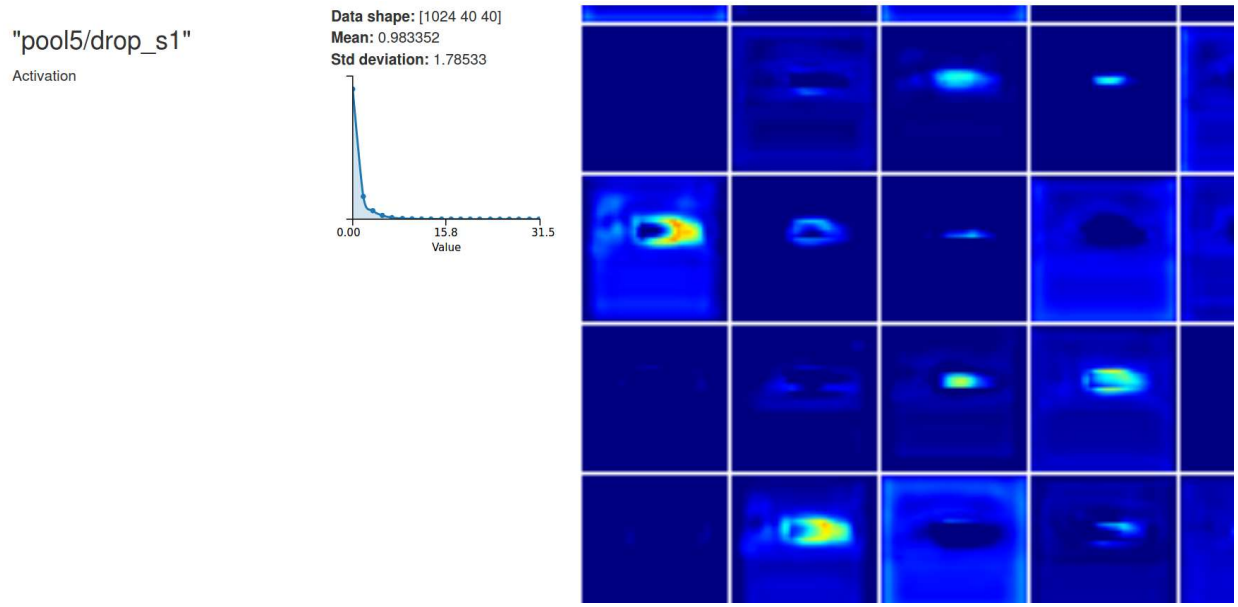
The visualization of the exemplary pooling layer “pool2/3x3_s2” is shown below.



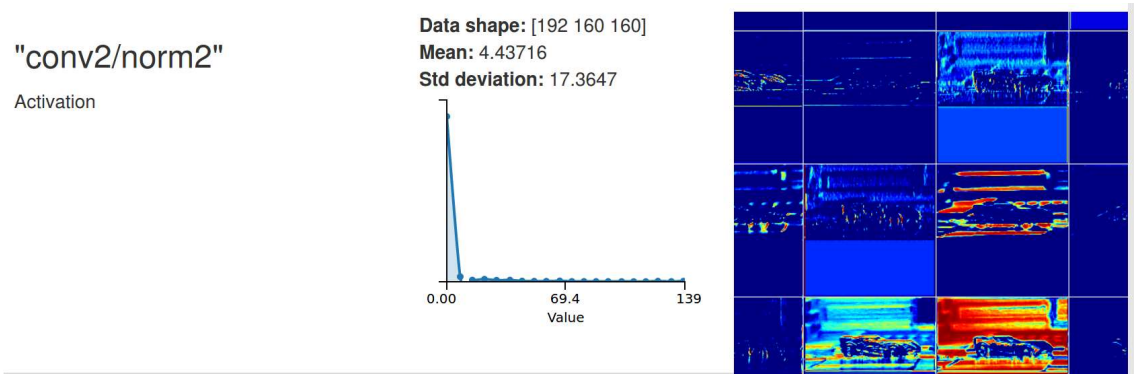
The visualization of the exemplary activation layer “conv2/relu_3x3” is shown below.



The visualization of the exemplary dropout layer “pool5/drop_s1” is shown below.



The visualization of the exemplary LRN layer “conv2/norm2” is shown below.



8.13. Appendix M: DetectNet Inference Python Script

This Python script was used to launch real-time object detection with DetectNet on the Jetson Nano embedded AI computing platform.

```
#!/usr/bin/python
#
# Copyright (c) 2019, NVIDIA CORPORATION. All rights reserved.
#
# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.
#
# NOTE:
# This script has been modified from the original detectnet-camera.py format
# in order to be used in a special use case. The special use case is part of
# Mark Csizmadia's final year BEng Electronic Engineering project, and is
# discuseed below.
#
# This script runs inference with a pretrained model using the camera of the
# Jetson Nano. The new functionality furnishes the user with the option to
# extract a rectangular region of interest (ROI) marked by four red dots.
# If the ROI option is selected (see argparse), then the inference will only
# consider that region. Otherwise, the script can perform the original
# functionality.


# Import packages.
import jetson.inference
import jetson.utils
import argparse
# My imports
import ctypes
import numpy as np
import cv2
import os
import sys
from pymodbus.client.sync import ModbusTcpClient
import time


# Positive number : 0 - 32767
# Negative number : 32768 - 65535
def number_to_raw_data(val):
    if val < 0:
        val = (1 << 15) - val
    return val

def raw_data_to_number(val):
    if(val>>15) == 1:
        val = -(val & 0x7FFF)
    return val


def hconcat_cv(im_list):
    """
    Concatenates images horizontally. Can handle images width different widths.
```

```
    Inputs:
        - im_list: a list of np.array representing images

    Outputs:
        - hconcat_imgs: horizontally concatenated images
    """
    # Calculate the minimum heigth of all of the images.
    height_min = min(im.shape[0] for im in im_list)

    # Resize the widths of the images in order to concatenate them horizontally.
    im_list_resized = [cv2.resize(im, (int(im.shape[1] * height_min / im.shape[0]),
        height_min), interpolation=cv2.INTER_CUBIC) for im in im_list]

    # Concatenate images.
    hconcat_imgs = cv2.hconcat(im_list_resized)

    return hconcat_imgs

def vconcat_cv(im_list):
    """
    Concatenates images vertically. Can handle images width different heights.

    Inputs:
        - im_list: a list of np.array representing images

    Outputs:
        - vconcat_imgs: vertically concatenated images
    """
    # Calculate the minimum width of all of the images.
    width_min = min(im.shape[1] for im in im_list)

    # Resize the heights of the images in order to concatenate them vertically.
    im_list_resized = [cv2.resize(im, (width_min, int(im.shape[0] * width_min / im.shape[1])),
        interpolation=cv2.INTER_CUBIC) for im in im_list]

    # Concatenate images.
    vconcat_imgs = cv2.vconcat(im_list_resized)

    return vconcat_imgs

def calibration_feedback(im_list):
    """
    Creates a feedback window for the calibration step. Shows the raw camera image, the cropped ROI,
    the red filtered and the morphoogically transformed images.
    Used to assess whether the calibration is successful.

    Inputs:
        - im_list: a list of np.array representing images

    Outputs:
        - vconcat_imgs: vertically concatenated images
    """
    # Pad each image for aesthetic window.
    padding = 20
    im_list_padded = [cv2.copyMakeBorder(im, padding, padding, padding, padding,
cv2.BORDER_CONSTANT,
        value=(255,255,255)) for im in im_list]

    # Concatenate the first two images horizontally.
    h1 = hconcat_cv(im_list_padded[:2])

    # Concatenate the first two images horizontally.
    h2 = hconcat_cv(im_list_padded[2:])

    # Concatenate the two results of the individual horizontal concatenation steps.
    v = vconcat_cv([h1,h2])

    # Get the height and the width of the resultant image.
    height, width = v.shape[0], v.shape[1]

    # Set coordinates and font properties for overlaying text on the image.
    font = cv2.FONT_HERSHEY_DUPLEX
    vertical_percentage = 0.025
    horizontal_percentage = 0.3
    top_left = (int(horizontal_percentage * width), int(vertical_percentage * height))
    bottom_left = (int(horizontal_percentage * width), int((1-vertical_percentage) * height))
    top_right = (int((1 - horizontal_percentage) * width), int(vertical_percentage * height))
```

```
    bottom_right = (int((1 - horizontal_percentage) * width), int((1-vertical_percentage) * height))
    font_scale = 0.8
    font_color = (120,120,120)
    line_type = 1

    # Put texts in the image.
    cv2.putText(v, "Raw camera image", top_left, font, font_scale, font_color, line_type)
    cv2.putText(v, "ROI", top_right, font, font_scale, font_color, line_type)
    cv2.putText(v, "Red filtered", bottom_left, font, font_scale, font_color, line_type)
    cv2.putText(v, "After morphological tansform", bottom_right, font, font_scale, font_color,
line_type)

    # Show the feedback.
    cv2.imshow("Calibration", v)
    print("Inspect the calibration feedback, and close the feedback window to continue.\n")
    cv2.imwrite("calibration.png",v)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def morphological_img_proc(initial_mask):
    """
    Morphological image processing step. Performs a sequence of closing and opening operations.

    Inputs:
        - initial_mask: np.array, 3 dims, the red masked image. Has spurious small islands that
            ruin the perspective trans.

    Outputs:
        - fully_masked: np.array, 3 dims, cleaned up version without the spurious islands.

    """
    # Create structrual elements for opening and closing.
    rect_close = (2,2) #2,2
    rect_open = (5,5) #5,5
    struct_el_close = cv2.getStructuringElement(cv2.MORPH_RECT, rect_close)
    struct_el_open = cv2.getStructuringElement(cv2.MORPH_RECT, rect_open)

    # Perform the closing and the opening.
    closed = cv2.morphologyEx(initial_mask, cv2.MORPH_CLOSE, struct_el_close)
    opened = cv2.morphologyEx(initial_mask, cv2.MORPH_OPEN, struct_el_open)
    #opened = cv2.morphologyEx(initial_mask, cv2.MORPH_OPEN, struct_el_open)

    # Mask the initial mask with the imroved morphological transform mask.
    # Convert the colors to black for masking?
    final_mask = opened / 255
    fully_masked = initial_mask * opened

    # Debug.
    #cv2.imshow("opened",opened)
    #cv2.waitKey(0)
    # Debug.
    #cv2.imshow("final_mask",final_mask)
    #cv2.waitKey(0)

    return fully_masked

def get_coordinates_of_vertices(img, color="red"):
    """
        Get the coordinates of the middle of the blobs that specify the rectangular
        Region of Interest (ROI).

        Inputs:
        - img: An image, from opencv, np.ndarray
        - color: green, red in str

    Outputs:
        - centers: Center of the blobs, ndarrayy, 2d, 4 rows of x,y pairs

        Resources:
        https://docs.opencv.org/3.0-
beta/doc/py_tutorials/py_ml/py_kmeans/py_kmeans_opencv/py_kmeans_opencv.html
        """
        # Create hsv colormap.
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

        # Select mask color (in project, red).
        if color == "green":
```

```
            mask = cv2.inRange(hsv, (36, 25, 25), (70, 255,255))
        elif color == "red":
            mask_1 = cv2.inRange(hsv, (0, 70, 50), (10, 255, 255))
            mask_2 = cv2.inRange(hsv, (170, 70, 50), (180, 255, 255))
            mask = mask_1 | mask_2
        else:
            raise Exception("Invalid mask color.")

        # Create boolean mask.
        imask = mask > 0

        # Create masked image array in memory, initialize to zeros (black pixels).
        masked = np.zeros_like(img, np.uint8)

        # Mask the image.
        masked[imask] = img[imask]

    # Debug.
    #cv2.imshow("masked",masked)
    #cv2.waitKey(0)

    # Morphological image processing.
    fully_masked = morphological_img_proc(initial_mask=masked)

    #Debug. Has to be perfectly clean as opposed to masked previously.
    #cv2.imshow("fully_masked",fully_masked)
    #cv2.waitKey(0)

        # Get the indices of the masked blobs.
        indices = np.where(fully_masked > 0)
        # Use the indices to create a list of tuples of the x,y coordinates of
        # the blob pixels.
        coordinates = list(zip(indices[0], indices[1]))

        # Since RGB, eliminate duplicate coordinate pairs in channels.
        unique_coordinates = list(set(coordinates))
    print(unique_coordinates)

        # Convert the coordinates to a numpy array where one row is an x,y pair.
        unique_coordinates_np_array = \
            np.array(unique_coordinates, dtype=np.float32)

        # Have to flip the array since at some point it got transformed.
        # Essesntailly setting the origin back to the top-left of the array.
        unique_coordinates_np_array = np.flip(unique_coordinates_np_array,1)

        # K-means clustering with K=4 to find the centroids of the blobs.
        # Define criteria = ( type, max_iter = 10 , epsilon = 1.0 )
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

        # Set flags (Just to avoid line break in the code)
        flags = cv2.KMEANS_RANDOM_CENTERS

        # Apply K-Means.
        compactness,labels,centers = \
            cv2.kmeans(unique_coordinates_np_array,4,None,criteria,10,flags)

        # Debug code. OK if the fully_masked blobs's centorids coincide with the k-means.
        #plt.imshow(fully_masked)
        #plt.scatter(centers[:,0], centers[:,1], c="r")

    # Get the ordered set of x,y coordinates of the vertices of the ROI.
        ordered_centers = order_points(centers=centers)

        return ordered_centers, masked, fully_masked

def order_points(centers):
        """
        Order the centers as follows: (1) top-left, (2) top-right,
        (3) bottom-right, (4) bottom-left

        Inputs:
            - centers: 2d numpy array of 4 x,y pairs

        Outputs:
            - ordered_centers: 2d numpy array of the 4 ordered x,y pairs
```

```
        Adopted from:
        https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/
        """
        # Create an array to hold the ordered center coordinates.
        ordered_centers = np.zeros((4, 2), dtype = "float32")

        # Sum calucations for finding the order.
        s = centers.sum(axis = 1)

        # The top-left has the min sum of x and y coordinates.
        ordered_centers[0] = centers[np.argmin(s)]

        # The bottom-right has the max sum of x and y coordinates.
        ordered_centers[2] = centers[np.argmax(s)]

        # Difference calucations for finding the order.
        diff = np.diff(centers, axis = 1)

        # The top-right has the min difference of x and y coordinates.?
        ordered_centers[1] = centers[np.argmin(diff)]

        # The bottom-left has the max difference of x and y coordinates.?
        ordered_centers[3] = centers[np.argmax(diff)]

        return ordered_centers

def perspective_transform(image, ordered_centers):
        """
        Perform perspective transform on the image using four coordinate paris.
        Extract the rectangular region of interest (ROI).

        Inputs:
            - image: An image, from opencv, np.ndarray
            - pts: An ordered set of x,y coordinates of the vertices of the ROI.
                    2d numpy array of 4 x,y pairs

        Outputs:
            - warped: the transformed image, shows the rectangular ROI
        """
        # Get the ordered set of x,y coordinates of the vertices of the ROI.
        (tl, tr, br, bl) = ordered_centers

        # Calculate the maximum width of the ROI. The maximum width will be the
        # width of the transformed image.
        widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
        widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
        maxWidth = max(int(widthA), int(widthB))

        # Calculate the maximum height of the ROI. The maximum height will be the
        # height of the transformed image.
        heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
        heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
        maxHeight = max(int(heightA), int(heightB))

        # Create the desination coordinates of the vertices of the image resulting
        # from transformation operation.
        dst = np.array([[0, 0],[maxWidth - 1, 0],[maxWidth - 1, maxHeight - 1],
            [0, maxHeight - 1]], dtype = "float32")

        # Perform the perspective transformation.
        # Get the transformation matrix using the ordered set of x,y coordinates
        # of the ROI in the original image, and the destination coordinates.
        M = cv2.getPerspectiveTransform(ordered_centers, dst)

        # Transform the original image matrix using the perspective transformation
        # matrix and the destination width and height.
        warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))

        return warped


def connect_to_arm(use_arm):
    client = None
    if use_arm:
        client = ModbusTcpClient('10.10.10.10', port=5020)
        client.connect()
        print("Connected to arm via modbus server.\n")
```

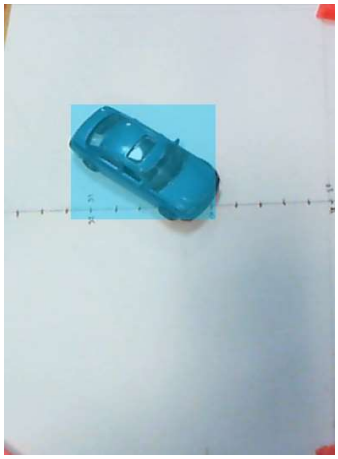
```
                client.write_registers(250, number_to_raw_data(1))
    return client

# Parse the command line. Custom argument: analyse_ROI.
parser = argparse.ArgumentParser(
    description="Locate objects in a live camera stream using an object detection DNN.",
    formatter_class=argparse.RawTextHelpFormatter, epilog=jetson.inference.detectNet.Usage())

parser.add_argument("--network", type=str, default="pednet",
    help="pre-trained model to load, see below for options")
parser.add_argument("--threshold", type=float, default=0.5,
    help="minimum detection threshold to use")
parser.add_argument("--camera", type=str, default="0",
    help="index of the MIPI CSI camera to use (NULL for CSI camera 0)\nor for VL42 cameras the
/dev/video node to use.\nby default, MIPI CSI camera 0 will be used.")
parser.add_argument("--width", type=int, default=1280,
    help="desired width of camera stream (default is 1280 pixels)")
parser.add_argument("--height", type=int, default=720,
    help="desired height of camera stream (default is 720 pixels)")
parser.add_argument("--analyse_ROI", type=int, default=0,
    help="whether to analyse the ROI (used for common space with robotic arm)")
parser.add_argument("--use_arm", type=int, default=0,
    help="whether to use the robotic arm")

opt, argv = parser.parse_known_args()

# Load the object detection network.
net = jetson.inference.detectNet(opt.network, argv, opt.threshold)

# Create the camera and display.
camera = jetson.utils.gstCamera(opt.width, opt.height, opt.camera)
display = jetson.utils.glDisplay()

print("\n" * 5)
print("Starting calibration...\n")

# Calibration Boolean flag.
calibrated = False

# Wait to for some time to start the calibration.
# E.g.: camera might need time to adjust to the lighting.
camera_smooth_time_idx = 0

####
client = connect_to_arm(use_arm=opt.use_arm)

##send_once = True
#inputs = [number_to_raw_data(0.25*1000),number_to_raw_data(0*1000),number_to_raw_data(1)]
#client.write_registers(200, inputs)
####

# Start calibration.
while display.IsOpen() and opt.analyse_ROI and not calibrated:
    # Capture RGBA camera image.
    img, width, height = camera.CaptureRGBA(zeroCopy=1)

    if camera_smooth_time_idx < 30:
        camera_smooth_time_idx = camera_smooth_time_idx + 1
        continue

    # Sync. CUDA with the camera for subsequent image processing steps. THIS IS IMPORTANT!
    jetson.utils.cudaDeviceSynchronize()

    # Convert CUDA GPU memory pointer PyCapsule to numpy array.
    img_np_rgba = jetson.utils.cudaToNumpy(img, width, height, 4)

    # Convert RGBA to RGB.
    img_np_rgb = cv2.cvtColor(img_np_rgba.astype (np.uint8), cv2.COLOR_RGBA2BGR)

    # Find the red vertices of the ROI (and also return mask images for calibration feedback).
    ordered_centers, masked, fully_masked = get_coordinates_of_vertices(img=img_np_rgb, color="red")

    # Perspective transform to see if well calibrated.
    transformed_img_np = perspective_transform(image=img_np_rgb, ordered_centers=ordered_centers)

    # List of images for calibration feedback.
    im_list = [img_np_rgb, transformed_img_np, masked, fully_masked]
```

```
    # Show calibration feedback.
    calibration_feedback(im_list)

    # Wait on user's response.
    cal_feedback = raw_input("Is the calibration good [y/n]?\n")

    # If user replies "y" for yes, then calibration is successful and the execution breaks
    # out of the calibration loop.
    if cal_feedback == "y":
        calibrated = True
        print("Calibration completed!\n")
        break
    # If user replies "n" for no, then the calibration was not successful and the
    # calibration loop is restarted.
    else:
        print("Restarting calibration...\n")
        continue

# Process frames until user exits.

print("Starting inference...\n")

while display.IsOpen():
    # Capture the image. zeroCopy=1 (otherwise PythonCapsule probelms)
    img, width, height = camera.CaptureRGBA(zeroCopy=1)

    # If the analyse_ROI script argument is true, then extract ROI and pass that
    # onto the network and the display.
    if opt.analyse_ROI:
        # This line is really important to be able to perfrom the
        # subsequent operations!!!!!!!!!!!!
        jetson.utils.cudaDeviceSynchronize()

        # Convert the CUDA GPU memory pointer PyCapsule to numpy array.
        img_np_rgba = jetson.utils.cudaToNumpy(img, width, height, 4)
        img_np_rgb = cv2.cvtColor(img_np_rgba.astype (np.uint8), cv2.COLOR_RGBA2BGR)

        # Transform the image and extract the ROI via perspective transform.
            transformed_img_np = perspective_transform(image=img_np_rgb,
                ordered_centers=ordered_centers)

        # Get the height and the width of the ROI (for the network).
        height = transformed_img_np.shape[0]
        width = transformed_img_np.shape[1]

        # Convert RGB back to RGBA for network and display.
        transformed_img = cv2.cvtColor(transformed_img_np, cv2.COLOR_BGR2RGBA)

        # Convert back to CUDA GPU memory pointer PyCapsule form numpy.
        img = jetson.utils.cudaFromNumpy(transformed_img)

    # Detect objects in the image (with overlay).
    detections = net.Detect(img, width, height)


    # Print the detections.
    print("Detected {:d} objects in image.\n".format(len(detections)))
    print("Listing detections:\n")

    roi_width = 150
    roi_height = 200



    if (len(detections) > 0 and opt.use_arm and (client.read_holding_registers(250, 1))):
        #client = ModbusTcpClient('10.10.10.10', port=5020)
        #client.connect()
        #print("Connected to arm via modbus server.\n")
                #client.write_registers(250, number_to_raw_data(1))

        confidence = detections[0].Confidence
        class_id = detections[0].ClassID

        y_min = (detections[0].Top / height) * roi_height
        x_min = (detections[0].Left / width) * roi_width
        y_max = (detections[0].Bottom / height) * roi_height
```

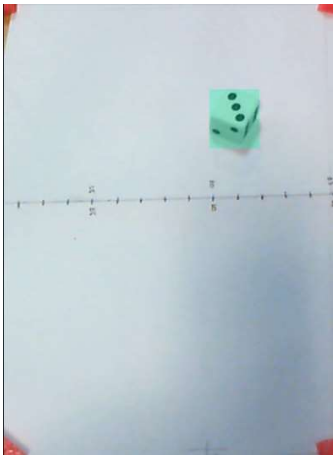
```
    x_max = (detections[0].Right / width) * roi_width

    inputs = [number_to_raw_data(x_min*100), number_to_raw_data(y_min*100),
          number_to_raw_data(x_max*100), number_to_raw_data(y_max*100),
          number_to_raw_data(class_id)]

    client.write_registers(200, inputs)
    client.write_registers(250, 0)

for detection in detections:
    print(detection)
    print(detection.Left)
    print("\n")

# Render the image.
display.RenderOnce(img, width, height)

# Update the title bar.
display.SetTitle("{:s} | Network {:.0f} FPS".format(opt.network,
    1000.0 / net.GetNetworkTime()))

# Synchronize with the GPU.
if len(detections) > 0:
    jetson.utils.cudaDeviceSynchronize()

# Print out performance info.
print("Network performance information:\n")
net.PrintProfilerTimes()
print("\n")
```

8.14. Appendix N: Examples of Real-Time Object Detection

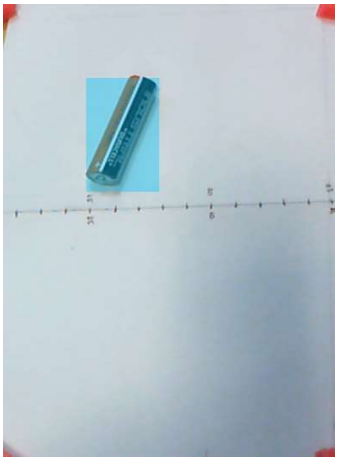
This appendix includes some real-time object detection snapshots of the Jetson Nano interface. Toy cars and batteries were successfully differentiated, but unfortunately, there was no time to change the color of their bounding boxes as described in Appendix E: The Effects of Coronavirus (COVID-19) on the Project



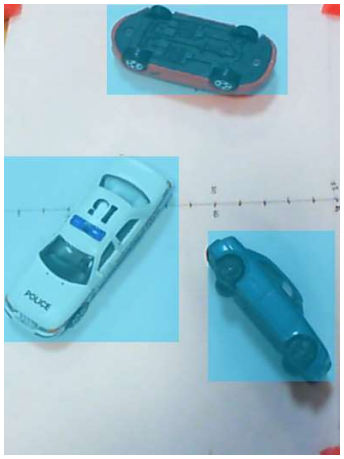
toycar_sinlge



dice_single

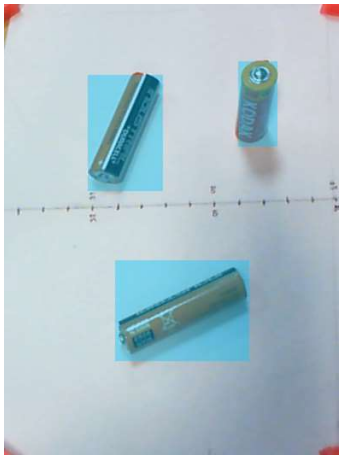


battery_single

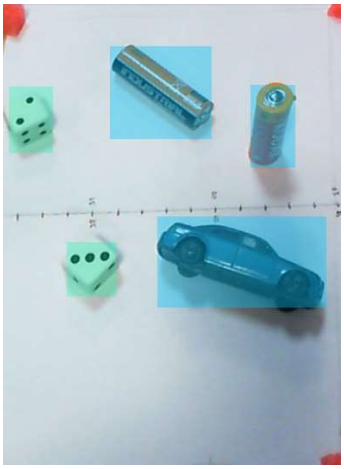


toycar_multi

dice_multi

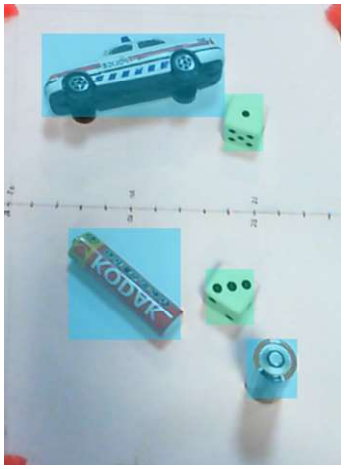


battery_multi



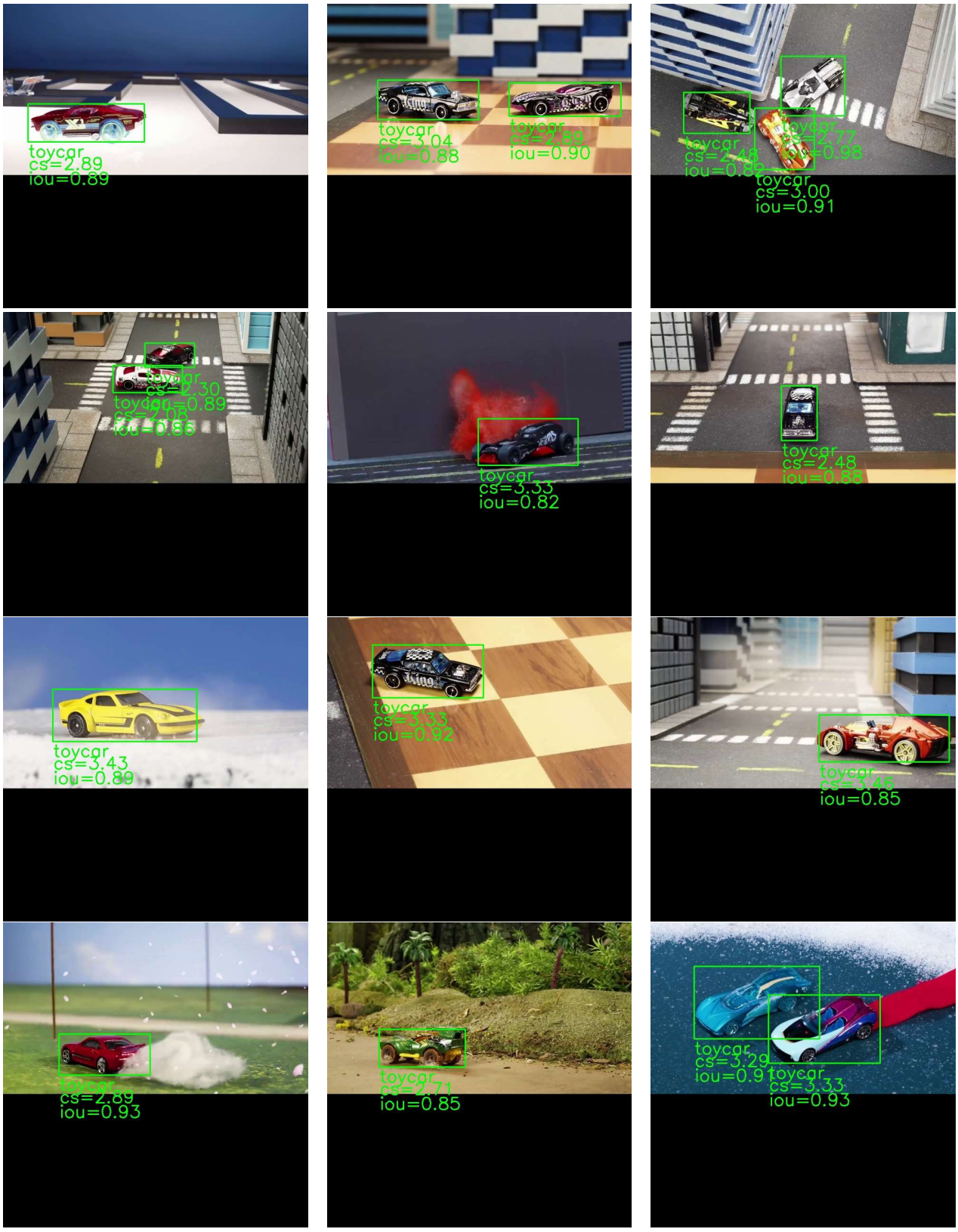
mixed_1

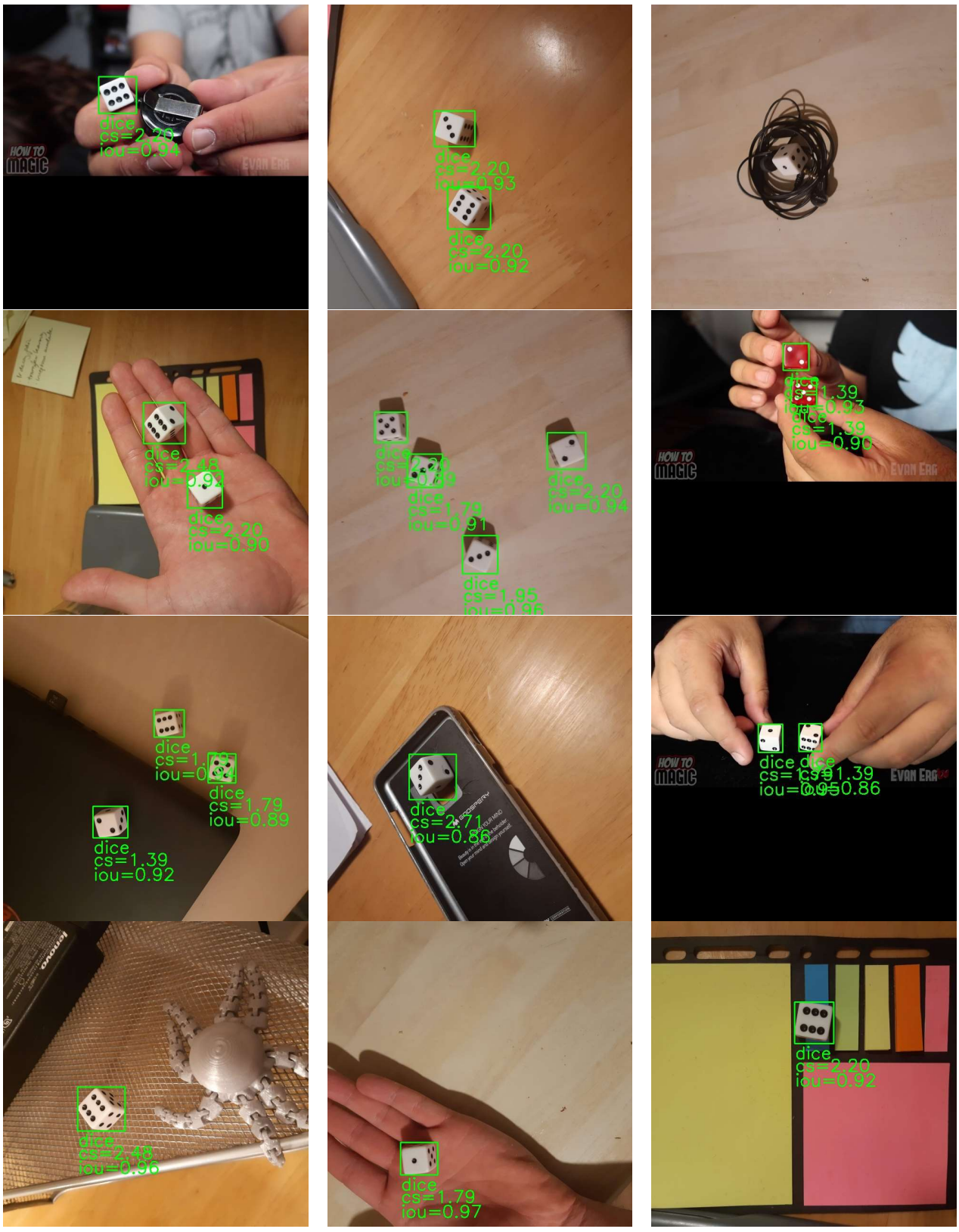
mixed_2

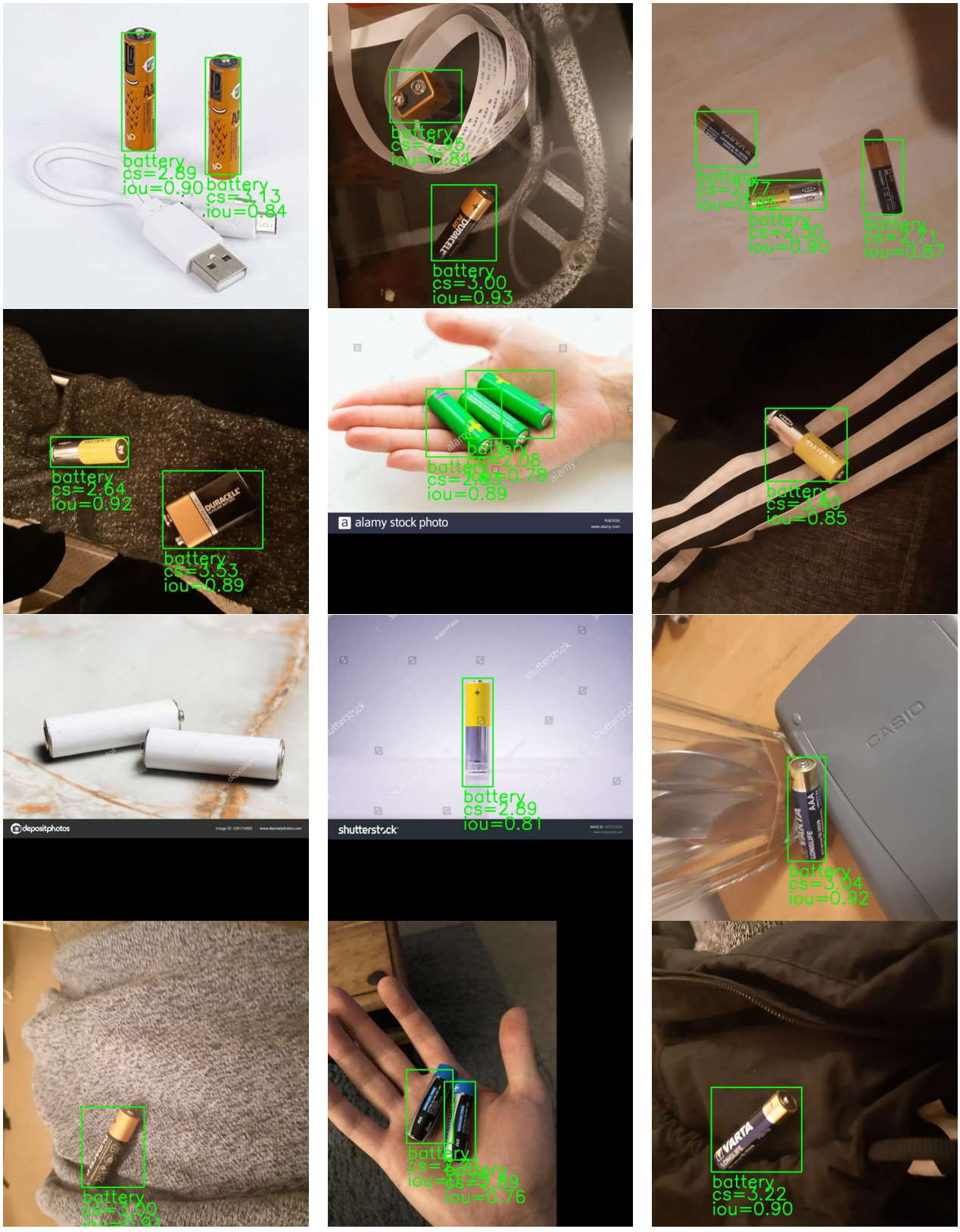


mixed_3

8.15. Appendix O: Examples of Detected Objects in Images




dice
cs=2.20
iou=0.94
dice
cs=2.20
iou=0.93
dice
cs=2.20
iou=0.92
dice
cs=2.48
iou=0.92
dice
cs=2.20
iou=0.90
dice
cs=1.79
iou=0.91
dice
cs=2.20
iou=0.94
dice
cs=1.95
dice
cs=1.39
iou=0.93
dice
cs=1.39
iou=0.90
dice
cs=1.39
iou=0.92
dice
cs=1.79
iou=0.89
dice
cs=2.71
iou=0.86
dice
cs=1.39
iou=0.86
dice
cs=2.48
iou=0.96
dice
cs=1.79
iou=0.97
dice
cs=2.20
iou=0.92


battery
cs=2.89
iou=0.90
battery
cs=3.13
iou=0.84
battery
cs=3.00
iou=0.93
battery
cs=2.64
iou=0.92
battery
cs=3.53
iou=0.89
alamy stock photo
battery
cs=2.89
iou=0.81
shutterstock
battery
cs=3.64
iou=0.92
battery
cs=3.22
iou=0.90
battery
iou=0.76

8.16. Appendix P: Performance Analysis and Visualization Script

This Python script was used to analyze and visualize the performance of the trained model.

```
# -*- coding: utf-8 -*-
"""
This Python script was produced by Mark Antal Csizmadia for the BEng Electronic
Engineering final year project titled Embedded Object Detection with Deep
Learning.
Author: Mark Antal Csizmadia
Institution: The Universtiy of Manchester

This script extracts the predicted bounding boxes produced by an object
detection model trained within the NVIDIA DIGITS environment. The predicted
bounding box coordinates are used to derive the following metrics: true
positive detection, false positive detection, false negative detection,
precision, recall, average precision (AP), and mean Average Precision (mAP).
"""


# Import packages.
from bs4 import BeautifulSoup
import numpy as np
import re
import os
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 22})
import cv2
import argparse


# Construct the argument parser and parse the arguments.
parser = argparse.ArgumentParser(
        description="Analyse and visualise the detecions.",
        formatter_class=argparse.RawTextHelpFormatter)

parser.add_argument("--path_to_dataset",
                    type=str,
                    required=True,
                    help="Give the path to the dataset directory.")

parser.add_argument("--path_to_htmls",
                    type=str,
                    required=True,
                    help="Path to the directory of the DIGITS HTMLs and save
destination.")

parser.add_argument("--mode",
                    type=str,
                    required=True,
                    help="train, val or test")

parser.add_argument("--lr",
                    type=str,
                    required=True,
                    help="Learning rate.")

# Functions.
def get_ground_truth_annotation(lines):
    """ Returns the upper left and the lower right coordinates of the ground
    truth bounding box for an image, read from the annotation files the in
    KITTI format.

    Inputs:
        -  lines: list of strs that define the ground truth bounding boxes
        in the KITTI data format
```

```
    Outputs:
        - ground_truth_bboxes: a dict of 2D lists. The keys are the object
        classes, and the values are 2D lists. Each 2D list contains
        the ground truth bounding boxes as a list of 4 floats (that
        define the top left and the bottom right corners of the
        bounding boxes).
        E.g.:
        {'toycar': [],
         'dice': [['123', '104', '289', '290'],
                  ['227', '284', '415', '506']],
         'battery': []}

    """
    ground_truth_bboxes = {"toycar":[], "dice":[], "battery":[],
                           "highlighter":[], "candle":[], "spoon":[]}
    for line in lines:
        ground_truth_bboxes[line.split(" ")[0]].append(
                [float(line.split(" ")[4]), float(line.split(" ")[5]),
                float(line.split(" ")[6]), float(line.split(" ")[7])])

    return ground_truth_bboxes

def get_iou(detection, ground_truth):
    """ Returns the Intersection over Union (IoU) of two rectangular bounding
        boxes.

        Inputs:
            - detection: the upper left and the lower right coordinates of the
            predicted bounding box
            - ground_truth: the upper left and the lower right coordinates of
            the ground truth bounding box

        Outputs:
            - intersection_over_union: the IoU of the two bounding boxes

    """
    # Get top left nad bottom right coordinates of the
    # detected bbox.
    top_left_d = (float(detection[0]), float(detection[1]))
    bottom_right_d = (float(detection[2]), float(detection[3]))
    # Get top left nad bottom right coordinates of the
    # ground truth bbox.
    top_left_gt = \
        (float(ground_truth[0]), float(ground_truth[1]))
    bottom_right_gt = \
        (float(ground_truth[2]), float(ground_truth[3]))
    # Calcualte the overlap of the two bounding boxes on the x-axis.
    x_overlap = \
        max(0, min(bottom_right_d[0], bottom_right_gt[0]) - \
            max(top_left_d[0], top_left_gt[0]))
    # Calcualte the overlap of the two bounding boxes on the y-axis.
    y_overlap = \
        max(0, min(bottom_right_d[1], bottom_right_gt[1]) - \
            max(top_left_d[1], top_left_gt[1]))
    # Calcualte the intersection of the area of the two bounding boxes..
    area_intersection = x_overlap * y_overlap
    # Calcualte the area of the bounding boxes.
    area_d = \
        (bottom_right_d[0] - top_left_d[0]) * \
        (bottom_right_d[1] - top_left_d[1])
    area_gt = \
        (bottom_right_gt[0] - top_left_gt[0]) * \
        (bottom_right_gt[1] - top_left_gt[1])
    # Calcualte the union of the area of the two bounding boxes.
    area_union = area_d + area_gt - area_intersection
    # Calcualte the IoU of the two bounding boxes.
    intersection_over_union = area_intersection / area_union

    return intersection_over_union
```

```
def process_digits_html(path_to_html, object_names):
    """ Turn the poorly formatted DIGITS HTML file into a pandas DataFrame
        for subsequent analysis.

        Inputs:
            - path_to_html: the path to the DIGITS HTML
            - object_names: ordered list (same as in DIGITS) of objects
            - confidence_score: the confidence score of the detection

        Outputs:
            - df: data processed and put into a pandas DataFrame

    """
    print("[INFO] Processing the DIGITS HTML file.\n")
    # Create a BeautifulSoup object of the HTML detection results.
    soup = BeautifulSoup(open(path_to_html), "html.parser")
    # Extract the detections stored in a table in the HTML file.
    table = soup.find("table", attrs={"class": "table"})
    # Extract the headings of the table.
    # NOTE: <th>Idx</th> has to be manually added to the table headnings. Fors
    # some reason, DIGITS does not name the column storing the predicted
    # detection lists. Adding this piece of code helps read that column for
    # further processing.
    # After adding the piece of code, the heading code should look like:
    # <td></td>
       # <th>Idx</th>
    # <th>Image</th>
    # <th>Data</th>
    headings = [th.get_text() for th in table.find("tr").find_all("th")]
    # Process the table.
    datasets = []
    for row in table.find_all("tr")[1:]:
        dataset = zip(headings, (td.get_text() for td in row.find_all("td")))
        datasets.append(dataset)

    d = {"Idx":[], "Image":[], "Data":[]}


    # datasets store the data in columns -Idx, Image and Data
    # The first column (Idx) is a unique index for the image (not the same as
    # the image file name) detections.
    # The second column (Image) is stores the image file names.
    # The third (Data) column stores the detection lists, e.g.:
    # bbox-list-class2. This means that the following coordinates are the bbox
    # coordinates for class 2.
    # The rule to extract the bbox coordinates: there are 50 elements in each
    # object's list. An element is list of 4 coordinates corresponding to one
    # detection. No detection is listed as 4 0s.
    for dataset in datasets:
        for idx, field in enumerate(dataset):
            # For the zeroth (Idx) and first (Image) colmun, add the dict
            # entry: key=column name, value=entry in table.
            # E.g.: Idx, 1 and Image=/path/to/dataset/00002792.jpg
            if idx != 2:
                d[field[0]].append(field[1])
            # For the second column (Data).
            # Extract the non-zero prediced bounding boxes of each object
            # class for each image.
            elif idx == 2:
                els = field[1]
                results = re.findall("\d+\.?\d*", els)
                bbox_delimiter = 5
                class_delimiter = 251
                class2_location = (0 * class_delimiter, 1 * class_delimiter)
                class1_location = (1 * class_delimiter, 2 * class_delimiter)
                class0_location = (2 * class_delimiter, 3 * class_delimiter)
                class2_list = results[class2_location[0]:class2_location[1]]
                class1_list = results[class1_location[0]:class1_location[1]]
```

```
                class0_list = results[class0_location[0]:class0_location[1]]
                class_lists = [class2_list, class1_list, class0_list]
                rand_d = {"toycar":[], "dice":[], "battery":[]}
                for class_list in class_lists:
                    class_tag = object_names[int(class_list[0])]
                    for idx_bbox in range(1,
                                          len(class_list[1:])+1-bbox_delimiter,
                                          bbox_delimiter):
                        bbox_with_str = \
                            class_list[idx_bbox:idx_bbox+bbox_delimiter]
                        bbox = [float(i) for i in bbox_with_str]
                        all_non_zero = True
                        for i in bbox:
                            if i == 0.0:
                                all_non_zero = False
                        if all_non_zero:
                            rand_d[class_tag].append(bbox)
                        else:
                            continue
                d["Data"].append(rand_d)
    # The d dicitonary now contains data as follows:
    # 3 columns: Idx, Image, Data
    # Idx is the unqiue detecion index (not same as image file name)
    # Image: image file path
    # Data: ONLY the non-zero bounding boxes for each object per image.
    # A Data entry is a dict with keys toycar, battery, dice, and values
    # as lists for each object class. The list inlcude sub-list of the 4
    # coordintaes of tthe upper left and the lower right vertices of the
    # predicted bboxes (only that non-zero detections).
    # Convert the dict ot pd DataFrame.
    df = pd.DataFrame(data=d)

    return df

def analyze_data(df, confidence_score_tresholds, iou_tresholds,
                 path_to_image_files, path_to_annotation_files):
    """ Turn the poorly formatted DIGITS HTML file into a pandas DataFrame
        for subsequent analysis.

        Inputs:
            - df: pandas DataFrame of detections
            - confidence_score_tresholds: list of confidence thresholds
            - iou_tresholds: list of iou thresholds

        Outputs:
            - df_results: pandas DataFrame that inlcudes information about
            detection types like true positive, and performance metrics like
            precision for each iou and condence score threshold combination

    """
    print("[INFO] Analyzing data.\n")
    # Initialize data structures.
    d_results = \
        {"cst":[], "iou_t":[],
         "tp_battery":[], "fp_battery":[], "tn_battery":[], "fn_battery":[],
         "tp_toycar":[], "fp_toycar":[], "tn_toycar":[], "fn_toycar":[],
         "tp_dice":[], "fp_dice":[], "tn_dice":[], "fn_dice":[]}

    df_results = pd.DataFrame(data=d_results)

    tp_d = {"battery":0, "toycar":0, "dice":0}

    fp_d = {"battery":0, "toycar":0, "dice":0}

    tn_d = {"battery":0, "toycar":0, "dice":0}

    fn_d = {"battery":0, "toycar":0, "dice":0}

    # Iterate over all of the confidence score treshold combination.
```

```
for cst_idx, confidence_score_treshold in enumerate(
        confidence_score_tresholds):
    for iou_idx, iou_treshold in enumerate(iou_tresholds):
        # Iterate over the object classes.
        for obj in object_names:
            # Set the following for the object class:
            # tp: true positive
            # fp: false positive
            # tn: true negative
            # fn: false negative
            # p: precision
            # r: recall
            tp = 0
            fp = 0
            tn = 0
            fn = 0
            # Iterate over all of the images.
            for idx, image in enumerate(df["Image"]):
                image_id = image.split("/")[-1].split(".")[0]
                # Get the annotation file and read the coordinates of the
                # ground truth bounding boxes (GTbboxes).
                annotation_file = \
                    os.path.join(path_to_annotation_files, image_id + ".txt")
                image_file = \
                    os.path.join(path_to_image_files, image_id + ".jpg")
                # Get top left nad bottom right coordinates of the
                # detected bbox.
                f = open(annotation_file, "r")
                lines = f.readlines()
                ground_truth_bboxes = \
                    get_ground_truth_annotation(lines=lines)
                # The detected bounding boxes (Dbboxes).
                detections = df["Data"].iloc[idx]
                # If at least 1 detection for object class in
                # current image.
                if detections[obj]:
                    # Iterate over each detection for the image that belongs
                    # to the current object class.
                    for detection in detections[obj]:
                        # Confidence score of detection.
                        confidence_score = float(detection[4])
                        # If detection has higher confidence score
                        # than treshold.
                        if confidence_score_treshold < confidence_score:
                            # If there is at least one ground truth box for
                            # the object class in the image, then
                            # Iterate over the ground-truths of the
                            # object class for the current image.
                            if ground_truth_bboxes[obj]:
                                # Iterate over the gt bboxes for the object
                                # class, and find the one with the
                                # maximum IoU for the current detection.
                                max_iou = 0
                                for gt_idx, ground_truth in enumerate(
                                        ground_truth_bboxes[obj]):
                                    # Calculate IoU of the detected and
                                    # the ground truth bboxes.
                                    intersection_over_union = \
                                        get_iou(detection, ground_truth)
                                    if max_iou < intersection_over_union:
                                        max_iou = intersection_over_union
                                # If there was a GTbbox for which the IoU
                                # exceeds the treshold, then the detection
                                # is correct (true positive).
                                if iou_treshold < max_iou:

                                    tp = tp + 1
```

```
                                    # If max_iou stayed 0 or ended up being
                                    # lower than iou_treshold,
                                    # then there was no actual GTbbox
                                    # for the detection (false positive)
                                    else:
                                        fp = fp + 1

                                # No ground-truths of the object class in
                                # the image for detection for the object class.
                                # Falsly detected the object class
                                # for a different object class (false positive)
                                else:
                                    fp = fp + 1

                            # If detection has lower confidence score
                            #than treshold.
                            else:
                                fn = fn + 1

                        # If no detection for object class in current image.
                        else:
                            # If there is at least one ground truth box for
                            # object class in image. Couldn't detect an object
                            # that it should have (false negative)
                            if ground_truth_bboxes[obj]:
                                fn = fn + 1

                            # If no detection for non-existent ground truth
                            # bounding box. Didn't detect what it indeed
                            # shouldn't have (true negative)
                            else:
                                pass
                        f.close()

                    tp_d[obj] = tp
                    fp_d[obj] = fp
                    tn_d[obj] = tn
                    fn_d[obj] = fn
                d_new = \
                    {"cst":confidence_score_treshold, "iou_t":iou_treshold,
                     "tp_battery":tp_d["battery"], "fp_battery":fp_d["battery"],
                     "tn_battery":tn_d["battery"], "fn_battery":fn_d["battery"],
                     "tp_toycar":tp_d["toycar"], "fp_toycar":fp_d["toycar"],
                     "tn_toycar":tn_d["toycar"], "fn_toycar":fn_d["toycar"],
                     "tp_dice":tp_d["dice"], "fp_dice":fp_d["dice"],
                     "tn_dice":tn_d["dice"], "fn_dice":fn_d["dice"]}
                df_results = df_results.append(d_new, ignore_index=True)

    # Cmpute the precision and recall for the combination of the confidence
    # and IoU thresholds.
    # P = TP / (TP + FP)
    # R = TP / (TP + FN)
    # battery
    df_results["p_battery"] = \
        df_results["tp_battery"] / \
        (df_results["tp_battery"] + df_results["fp_battery"])

    df_results.loc[df_results["tp_battery"] == 0,"p_battery"] = 0

    df_results["r_battery"] = \
        df_results["tp_battery"] / \
        (df_results["tp_battery"] + df_results["fn_battery"])

    df_results.loc[df_results["tp_battery"] == 0,"r_battery"] = 0

    # toycar
    df_results["p_toycar"] = \
        df_results["tp_toycar"] / \
        (df_results["tp_toycar"] + df_results["fp_toycar"])
```

```
    df_results.loc[df_results["tp_toycar"] == 0,"p_toycar"] = 0

    df_results["r_toycar"] = \
        df_results["tp_toycar"] / \
        (df_results["tp_toycar"] + df_results["fn_toycar"])

    df_results.loc[df_results["tp_toycar"] == 0,"r_toycar"] = 0

    # dice
    df_results["p_dice"] = \
        df_results["tp_dice"] / (df_results["tp_dice"] + df_results["fp_dice"])

    df_results.loc[df_results["tp_dice"] == 0,"p_dice"] = 0

    df_results["r_dice"] = \
        df_results["tp_dice"] / (df_results["tp_dice"] + df_results["fn_dice"])

    df_results.loc[df_results["tp_dice"] == 0,"r_dice"] = 0

    # Compute the sum of the detection types.
    df_results["sum_dice"] = \
        df_results["tp_dice"] + df_results["fp_dice"] + df_results["fn_dice"]

    df_results["sum_toycar"] = \
        df_results["tp_toycar"] + df_results["fp_toycar"] \
        + df_results["fn_toycar"]

    df_results["sum_battery"] = \
        df_results["tp_battery"] + df_results["fp_battery"] \
        + df_results["fn_battery"]

    return df_results

if __name__ == "__main__":
    # Retrieve arguments.
    opt, argv = parser.parse_known_args()

    # Identifier texts for selecting dataset and saving figures.
    #mode = "test"
    identifier = "(final)"
    #lr = "0_0001"
    # The order is the same as in the DIGITS dataset creation step.
    object_names = ["toycar", "dice", "battery"]

    # Paths.
    # Path to DIGITS HTML.
    path_to_html = \
        os.path.join(opt.path_to_htmls,
                     "Infer Many Images Test lr "+opt.lr+" FINAL.html")
    # Path to the images in the dataset (train, val or test).
    path_to_annotation_files = \
        os.path.join(opt.path_to_dataset, opt.mode, "labels")
    path_to_image_files = \
        os.path.join(opt.path_to_dataset, opt.mode, "images")


    # Process the DIGITS HTML file.
    df = process_digits_html(path_to_html=path_to_html,
                             object_names=object_names)

    # Analyse the deetctions.
    # Linaer space for confidence and IoU thresholds.
    confidence_score_tresholds = np.linspace(0, 350, 50) / 100
    iou_tresholds = np.linspace(50, 90, 5) / 100

    df_results = \
        analyze_data(df=df,
                     confidence_score_tresholds=confidence_score_tresholds,
```

```
                iou_tresholds=iou_tresholds,
                path_to_image_files=path_to_image_files,
                path_to_annotation_files=path_to_annotation_files)

################################################################################
#PLOTS
print("[INFO] Plotting data.\n")
for object_name in object_names:

    object_tag = object_name.capitalize()
    precison_tag = "p_"+object_name
    recall_tag = "r_"+object_name

    ################################################################################
    # START: Plot scatter of different ious for all confidences.

    fig0 = plt.figure(num=None, figsize=(12, 12), dpi=80, facecolor='w',
                      edgecolor='k')

    ax0 = fig0.add_subplot(111)
    ax0.grid(alpha=0.3, color='k', linestyle='-', linewidth=0.5,
             which="both")

    iou0_cond = df_results["iou_t"] == iou_tresholds[0]
    ax0.scatter(np.array(df_results[iou0_cond].loc[:, recall_tag]),
                np.array(df_results[iou0_cond].loc[:, precison_tag]),
                marker=".", c="b", label="iou={0:.2}".format(
                        iou_tresholds[0]))
    iou1_cond = df_results["iou_t"] == iou_tresholds[1]
    ax0.scatter(np.array(df_results[iou1_cond].loc[:, recall_tag]),
                np.array(df_results[iou1_cond].loc[:, precison_tag]),
                marker=".", c="r", label="iou={0:.2}".format(
                        iou_tresholds[1]))
    iou2_cond = df_results["iou_t"] == iou_tresholds[2]
    ax0.scatter(np.array(df_results[iou2_cond].loc[:, recall_tag]),
                np.array(df_results[iou2_cond].loc[:, precison_tag]),
                marker=".", c="y", label="iou={0:.2}".format(
                        iou_tresholds[2]))
    iou3_cond = df_results["iou_t"] == iou_tresholds[3]
    ax0.scatter(np.array(df_results[iou3_cond].loc[:, recall_tag]),
                np.array(df_results[iou3_cond].loc[:, precison_tag]),
                marker=".", c="c", label="iou={0:.2}".format(
                        iou_tresholds[3]))
    iou4_cond = df_results["iou_t"] == iou_tresholds[4]
    ax0.scatter(np.array(df_results[iou4_cond].loc[:, recall_tag]),
                np.array(df_results[iou4_cond].loc[:, precison_tag]),
                marker=".", c="m", label="iou={0:.2}".format(
                        iou_tresholds[4]))

    ax0.legend(loc="lower left")
    ax0.set_title("Precision-Recall scatter plot for {0}".format(
            object_tag))
    fig0.savefig(
            os.path.join(
                    opt.path_to_htmls,
                    'fig0_'+object_tag+'_'+opt.mode+'_'+opt.lr+'_.png'))

    # END: Plot scatter of different ious for all confidences.

    ################################################################################
    # START: Plot interpolated scatter of different ious for all confidences.

    fig1 = plt.figure(num=None, figsize=(14, 12), dpi=80, facecolor='w',
                      edgecolor='k')

    ax0 = fig1.add_subplot(111)

    recall0 = np.array(df_results[iou0_cond].loc[:, recall_tag])[::-1]
    precision0 = np.array(df_results[iou0_cond].loc[:, precison_tag])[::-1]
```

```
precision02=precision0.copy()
i=recall0.shape[0]-2

# interpolation...
while i>=0:
    if precision0[i+1]>precision0[i]:
        precision0[i]=precision0[i+1]
    i=i-1

precision_iou0 = precision0

for i in range(recall0.shape[0]-1):
    # vertical
    ax0.plot((recall0[i],recall0[i]),
             (precision0[i],precision0[i+1]),'k-',label='',color='b')
    # horizontal
    ax0.plot((recall0[i],recall0[i+1]),
             (precision0[i+1],precision0[i+1]),'k-',label='',color='b')

ax0.plot(recall0,precision02,'k--',label="iou={0:.2}".format(
        iou_tresholds[0]), color='b')


recall0 = np.array(df_results[iou1_cond].loc[:, recall_tag])[::-1]
precision0 = np.array(df_results[iou1_cond].loc[:, precison_tag])[::-1]
precision02=precision0.copy()
i=recall0.shape[0]-2

# interpolation...
while i>=0:
    if precision0[i+1]>precision0[i]:
        precision0[i]=precision0[i+1]
    i=i-1

precision_iou1 = precision0

for i in range(recall0.shape[0]-1):
    # vertical
    ax0.plot((recall0[i],recall0[i]),
             (precision0[i],precision0[i+1]),'k-',label='',color='r')
    # horizontal
    ax0.plot((recall0[i],recall0[i+1]),
             (precision0[i+1],precision0[i+1]),'k-',label='',color='r')

ax0.plot(recall0,precision02,'k--',label="iou={0:.2}".format(
        iou_tresholds[1]), color='r')


recall0 = np.array(df_results[iou2_cond].loc[:, recall_tag])[::-1]
precision0 = np.array(df_results[iou2_cond].loc[:, precison_tag])[::-1]
precision02=precision0.copy()
i=recall0.shape[0]-2

# interpolation...
while i>=0:
    if precision0[i+1]>precision0[i]:
        precision0[i]=precision0[i+1]
    i=i-1

precision_iou2 = precision0

for i in range(recall0.shape[0]-1):
    # vertical
    ax0.plot((recall0[i],recall0[i]),
             (precision0[i],precision0[i+1]),'k-',label='',color='y')
    # horizontal
    ax0.plot((recall0[i],recall0[i+1]),
             (precision0[i+1],precision0[i+1]),'k-',label='',color='y')
```

```
ax0.plot(recall0,precision02,'k--',label="iou={0:.2}".format(
        iou_tresholds[2]), color='y')


recall0 = np.array(df_results[iou3_cond].loc[:, recall_tag])[::-1]
precision0 = np.array(df_results[iou3_cond].loc[:, precison_tag])[::-1]
precision02=precision0.copy()
i=recall0.shape[0]-2

# interpolation...
while i>=0:
    if precision0[i+1]>precision0[i]:
        precision0[i]=precision0[i+1]
    i=i-1

precision_iou3 = precision0

for i in range(recall0.shape[0]-1):
    # vertical
    ax0.plot((recall0[i],recall0[i]),
             (precision0[i],precision0[i+1]),'k-',label='',color='c')
    # horizontal
    ax0.plot((recall0[i],recall0[i+1]),
             (precision0[i+1],precision0[i+1]),'k-',label='',color='c')

ax0.plot(recall0,precision02,'k--',label="iou={0:.2}".format(
        iou_tresholds[3]), color='c')


recall0 = np.array(df_results[iou4_cond].loc[:, recall_tag])[::-1]
precision0 = np.array(df_results[iou4_cond].loc[:, precison_tag])[::-1]
precision02=precision0.copy()
i=recall0.shape[0]-2

# interpolation...
while i>=0:
    if precision0[i+1]>precision0[i]:
        precision0[i]=precision0[i+1]
    i=i-1

precision_iou4 = precision0

for i in range(recall0.shape[0]-1):
    # vertical
    ax0.plot((recall0[i],recall0[i]),
             (precision0[i],precision0[i+1]),'k-',label='',color='m')
    # horizontal
    ax0.plot((recall0[i],recall0[i+1]),
             (precision0[i+1],precision0[i+1]),'k-',label='',color='m')

ax0.plot(recall0,precision02,'k--',label="iou={0:.2}".format(
        iou_tresholds[4]), color='m')

ax0.set_title("Interpolated Precision-Recall Curve of {0} Object Class".format(
        object_tag))
ax0.set_xlabel("Recall")
ax0.set_ylabel("Precision")
ax0.legend(loc="lower left")

# END: Plot interpolated scatter of different ious for all confidences.
#######################################################################
# START: AP calculations for all iou cond over all confs for object.
num_val_ap = 11
recall_points = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

recall_ap = np.array(df_results[iou0_cond].loc[:, recall_tag])[::-1]
precision_for_interp = \
    np.array(df_results[iou0_cond].loc[:, precison_tag])[::-1]
precision_ap = []
```

```
for recall_sample in recall_points:
    precision_ap.append(
            np.interp(recall_sample, recall_ap, precision_iou0))

ap = np.array(precision_ap).sum()/num_val_ap
txt0 = "iou={0:.2} AP={1:.2}".format(iou_tresholds[0], ap)

recall_ap = np.array(df_results[iou1_cond].loc[:, recall_tag])[::-1]
precision_for_interp = \
    np.array(df_results[iou1_cond].loc[:, precison_tag])[::-1]
precision_ap = []
for recall_sample in recall_points:
    precision_ap.append(
            np.interp(recall_sample, recall_ap, precision_iou1))

ap = np.array(precision_ap).sum()/num_val_ap
txt1 = "iou={0:.2} AP={1:.2}".format(iou_tresholds[1], ap)

recall_ap = np.array(df_results[iou2_cond].loc[:, recall_tag])[::-1]
precision_for_interp = \
    np.array(df_results[iou2_cond].loc[:, precison_tag])[::-1]
precision_ap = []
for recall_sample in recall_points:
    precision_ap.append(
            np.interp(recall_sample, recall_ap, precision_iou2))

ap = np.array(precision_ap).sum()/num_val_ap
txt2 = "iou={0:.2} AP={1:.2}".format(iou_tresholds[2], ap)


recall_ap = np.array(df_results[iou3_cond].loc[:, recall_tag])[::-1]
precision_for_interp = \
    np.array(df_results[iou3_cond].loc[:, precison_tag])[::-1]
precision_ap = []
for recall_sample in recall_points:
    precision_ap.append(
            np.interp(recall_sample, recall_ap, precision_iou3))

ap = np.array(precision_ap).sum()/num_val_ap
txt3 = "iou={0:.2} AP={1:.2}".format(iou_tresholds[3], ap)


recall_ap = np.array(df_results[iou4_cond].loc[:, recall_tag])[::-1]
precision_for_interp = \
    np.array(df_results[iou4_cond].loc[:, precison_tag])[::-1]
precision_ap = []
for recall_sample in recall_points:
    precision_ap.append(
            np.interp(recall_sample, recall_ap, precision_iou4))

ap = np.array(precision_ap).sum()/num_val_ap
txt4 = "iou={0:.2} AP={1:.2}".format(iou_tresholds[4], ap)


txt_all = "{0}\n{1}\n{2}\n{3}\n{4}".format(txt0, txt1, txt2, txt3,
          txt4)
ax0.text(1.1, 0.92, txt_all, horizontalalignment='center',
         verticalalignment='center', transform=ax0.transAxes,
         fontsize=13)

fig1.subplots_adjust(left=None, bottom=None,
                         right=0.86, top=None,
                         wspace=None, hspace=None)

fig1.savefig(os.path.join(
                opt.path_to_htmls,
                'fig1_'+object_tag+'_'+opt.mode+'_'+opt.lr+'_.png'))
######################################################################
print("[INFO] The visualizations are ready. \n")
```