Localizing Little Landmarks with Transfer Learning

by
Sharad Kumar

A thesis submitted in partial fulfillment of the
requirements for the degree of
Master of Science
in
Computer Science

Thesis Committee:
Melanie Mitchell, Chair
Bart Massey
Kristin Tufte

Portland State University
2018

Abstract

Locating a small object in an image — like a mouse on a computer desk or the door handle of a car — is an important computer vision problem to solve because in many real life situations a small object may be the first thing that gets operated upon in the image scene. While a significant amount of artificial intelligence and machine learning research has focused on localizing prominent objects in an image, the area of small object detection has remained less explored. In my research I explore the possibility of using context information to localize small objects in an image. Using a Convolutional Neural Network (CNN), I create a regression model to detect a small object in an image where model training is supervised by coordinates of the small object in the image. Since small objects do not have strong visual characteristics in an image, it's difficult for a neural network to discern their pattern because their feature map exhibits low resolution rendering a much weaker signal for the network to recognize. Use of context for object detection and localization has been studied for a long time. This idea is explored by Singh *et al.* [25] for small object localization by using a multi-step regression process where spatial context is used effectively to localize small objects in several datasets. I extend the idea in this research and demonstrate that the technique of localizing in steps using contextual information when used with transfer learning can significantly reduce model training time.

Acknowledgement

This thesis would not be possible without Professor Melanie Mitchell's guidance through the research. Her savvy idea to use deep learning for identifying a small object helped me gain more insight into machine learning in general and deep learning in particular. I also want to acknowledge Sheng Lundquist's help with TensorFlow for design, development and debugging of the model. Achnowledgement is in order for Saurabh Singh also who provided datasets and original code for this work.

Contents

List of Figures

List of Tables

Dedicated to my son, for his patience while daddy worked...

Chapter 1

Introduction

We use myriad small objects everyday without noticing their presence. We manipulate an inconspicuous car door handle when interacting with the car, use a barely visible light switch as we enter a dark room and a computer mouse may be the first object we operate when we sit down to use a computer. Despite the usefulness of small objects, their detection in an image is a relatively unexplored area in computer vision. Detection of small objects in an image is a difficult task, since these objects often do not have distinctive appearance. While Convolutional Neural Network (CNN) architectures have demonstrated superior performance for classification or localization of salient objects in an image, these architectures have not been very effective for identifying objects or relationships that occupy a small part of an image, since low resolution of small objects provide weak signal to the network for recognition. Another difficulty in small object detection, as noted by Chen *et al.* [3], is that the precision requirement for accurate localization of small objects is several magnitudes higher as compared to large objects. The reason for this predicament is that the bounding box of a small object is much smaller as compared to the bounding box of a large object in an image and as a result, to meet acceptable performance requirement, the available pixels that the predicted bounding box of a small object in an image can deviate from the actual bounding box is much smaller when compared to the number of pixels the predicted bounding box of a large object can deviate from the actual one.

Singh *et al.* [25] propose a recurrent CNN architecture that uses successively more relevant contextual information in a sequence of steps to localize small objects. Singh *et al.* call these small objects *little landmarks*. The training in this regression model is supervised by coordinates of the little landmark. The final step in this scheme predicts the small object's location, while prior steps predict where to look next for localization. The learning therefore discovers globally distinctive

patterns to start the sequence and conditionally distinctive ones, called *latent landmarks*, to get closer to the target in discrete steps. Figure 1 illustrates this process.

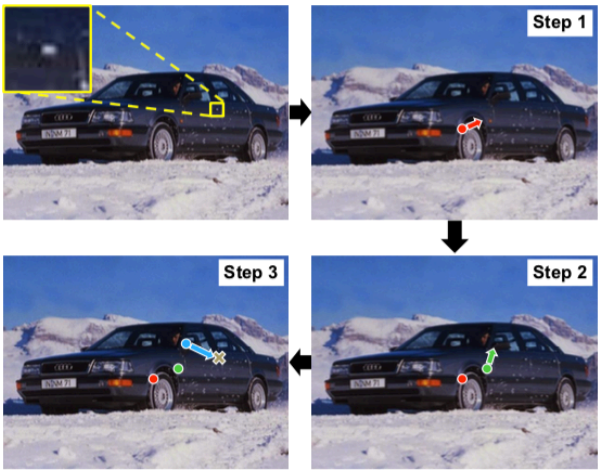


Figure 1: This figure illustrates the process of detecting the car door handle (barely recognizable in top left corner of the figure) in the image of a car. Step 1 finds the first *latent landmark*, which predicts location of the second *latent landmark*. The latent landmark found in step 2 predicts the location of the car door handle, which the model learns to localize in the third step of the process. Image source: [25]. *This figure is best viewed in color.*

In this thesis I recreate the model proposed by Singh *et al.* using TensorFlow [8]. I train the model on the same dataset that was used by Singh *et al.* The parameters in the original model are randomly initialized and the model is trained end-to-end. I propose a modification to the model by incorporating *transfer learning*. Transfer learning is a process in which image features learned by a trained network can be used by another network for its own training. This techniques may result in reducing training time and may also be useful when there's is dearth of training data, as I explain in section 5.1 of this thesis. My goals in this thesis are following: i) test transfer learning as a way to speed up training of this system and observe the effect of fewer training examples on test accuracy of the model; and ii) to test generality of Singh *et al.*'s model by training it on a new dataset whose spatial characteristics are different from the datasets used by Singh *et al.*

Singh *et al.* demonstrate the robustness of their model by localizing small objects in several datasets, such as an electrical switch on a wall, a car door handle and the beak of a bird. Their results and the design philosophy that exploits context information for small object detection provides strong evidence that this technique can also be used for localization of other small objects in an image, since context is an important component that often helps in localizing an object in an image. However, as Singh *et al.* train their model from scratch, their model takes a long time to train and needs a large amount of data for training. These limitations of their model was my motivation to modify their design and use transfer learning to build a more efficient model. Using transfer learning, parameters learned by a model that has been trained on a large dataset can

be effectively used as the initial parameters for some other model. To exploit transfer learning, I modify the original model by repurposing a pre-trained VGG-16 [24] model where I use activations from *pool4* layer of the VGG-16 model to initialize part of the model and with experiments I demonstrate that incorporating transfer learning in the model in such a way significantly reduces training time without affecting the model accuracy. VGG-16 is a deep CNN model created by K. Simonyan and A. Zisserman from University of Oxford that achieved 92.7% accuracy in top-5 test categories and 70.5% in top-1 test category on the Imagenet dataset [23]. Imagenet is a dataset of over 14 million images belonging to 1000 categories. I demonstrate significant improvement in training time with my approach and report results in chapter 6.

Researchers at Portland State University have created a labeled dataset of "dog-walking" images as part of a project on recognizing visual situations in images. Chapter 5 provides statistics of this dataset. I use this dataset to test the effectiveness of little landmark CNN model to localize the "hand-holding-leash" relationship in a "dog-walking" image. Successful localization of this little landmark would be useful for the overall task of detecting relevant parts of the "walking a dog" visual situation (Figure 2), such as likely location of the dog and the person walking the dog in the image.

Figure 2: Dog Walking Image from PSU Dog Walking dataset. Detection of the "hand-holding-leash" little landmark can help localize the dog walker and the dog in the image, and give evidence that this image is an example of the "dog-walking" situation. *This figure is best viewed in color.*

I performed four main experiments on the little landmark CNN model. In the first experiment, I recreated the model using TensorFlow [8] and trained it from scratch using the car door handle dataset to localize a car door handle in the image of a car. The purpose of this experiment was to demonstrate that my model faithfully replicates the original model. The test accuracy for this

model was similar to that reported by Singh *et al.* In the second experiment, I used transfer learning to modify the model and observed significant improvement in training time for the same dataset and achieved the same accuracy as in the first experiment. In the third experiment I trained the same model as in the second experiment, but localized on "hand-holding-leash" landmark in dog-walking images. The localization in the third experiment performed poorly. The size of data available for experiment 3 was about 20% of the size of data used in experiment 2. In the last experiment, I repeated experiment 2, but with same size of data that was used in experiment 3. The test accuracy in this case was inferior than that of of experiment 2, but better than experiment 3. I discuss these experiments and results in more detail in chapter 6.

The remainder of this thesis is organized as follows. Chapter 2 contains a brief discussion of CNNs, the mechanics of training a CNN model and how regularization is used in finding a balance between bias and variance to minimize generalization error. I discuss related work in chapter 3. In chapter 4 I explain the CNN architecture for the little landmark localization task and also discuss the prediction model for little and latent landmarks. In chapter 5 I discuss my model modifications and in chapter 6 I present results of my experiments. I conclude in chapter 7 where I summarize my research findings and and discuss possible extensions to this work.

Chapter 2

Background

Since its resurgence in 2012, when a neural network based architecture called AlexNet was proposed by Krizhevsky *et al.* [17] for image classification of the ImageNet [23] dataset, a variant of neural network architecture called Convolutional Neural Networks (CNN) has achieved significant improvement in image classification as well as regression tasks. Another influential work in this area was done by Girshick *et al.* [9] where they propose an architecture that uses region proposals with CNN for object localization. Their method is called R-CNN. Understanding of the CNN architecture and the object detection scheme in the R-CNN architecture is necessary for understanding the little landmark localization system proposed by Singh *et al.* and my extension to their architecture. In this chapter I provide an overview of CNN and R-CNN architectures.

2.1 Convolutional Neural Networks (CNNs)

A CNN is a deep learning model [1] and is essentially a special kind of neural network that can be used to solve classification or regression problems. In a tradition neural network, the parameter values (weights) of the model are stored in a two-dimensional matrix and the interaction between input and output layers of the models is encoded as the result of multiplication operations between the parameter matrix and a one-dimensional input vector, whereas the connectivity between input and output units in a CNN is partial in the sense that an input unit is connected only to a few output units — via a weight vector (called a kernel or filter). Sparse interactions between input

[1] Deep Learning AI systems can be represented as a deep graph with many layers that represent hierarchical structure of the learning network. These systems understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts [11].

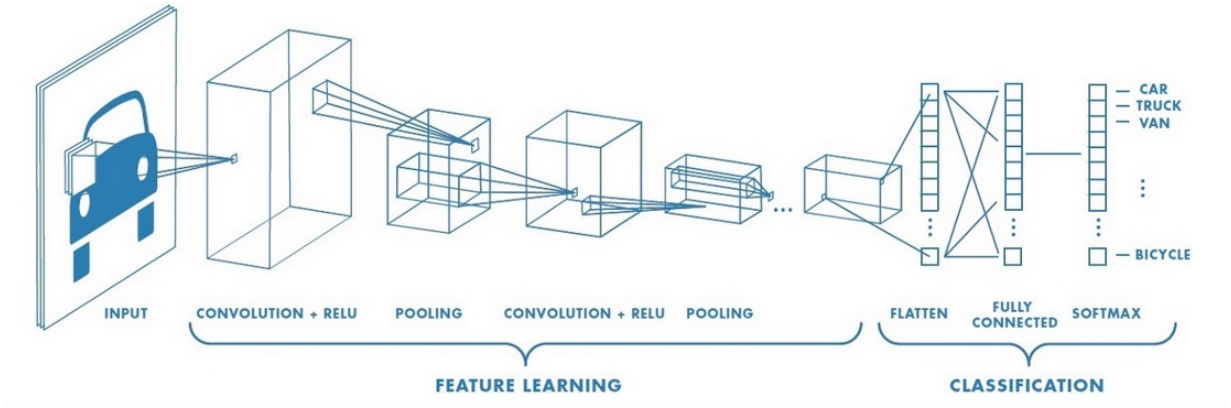


Figure 3: Schematic diagram of a Convolutional Neural Net (CNN). A typical layer of a CNN has three components; convolution (CONV), non-linearity (ReLU) and pooling (POOL). Many such layers are stacked together to create a deep architecture. Input enters the architecture from the left (image of the car) and after going through many layers of CONV-ReLU-POOL transformations produces output (image classification) in the rightmost layer. In front part of the architecture (feature learning), inputs and outputs are partially connected, whereas towards the end learned features of the model are passed thru a fully connected layer to generate a probability distribution for classification classes or a real value for a regression problem. Image source: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html. *This figure is best viewed in color.*

and output units in a CNN results in an architecture where training is much more efficient because of reduced number of parameters in the model as compared to the number of parameters in a traditional neural network of the same depth. Another important design characteristic of a CNN is that weights in a kernel are shared across all interactions, which further reduces the number of parameters in the model. I explain CNN architecture in more detail next.

Figure 3 shows a schematic diagram of a general CNN architecture. A layer of CNN typically has convolution, non-linearity (Rectified Linear Unit — ReLU) and pooling stages. CNN layers are arranged in a sequence and, in this hierarchical model, each layer learns features or representations at increasing levels of abstraction. The input enters the model from left and in this example is classified as one of several image classes by the rightmost layer of the model. However, this architecture can also be used to solve a regression problem where the output is a single continuous value. As depicted in Figure 3, a small patch of input generates a small patch of output in the CONV-ReLU stage. The generated output is called a feature map of activations. These activations are generated using a kernel as shown in Figure 4. Figure 4 shows a *5 X 5* input when processed with a *3 X 3* filter produces an output of dimension *3 X 3*. The output is generated by taking the

dot product of the kernel with the input as it slides (convolves) over the input, in increments of one cell in this example. In Figure 4 the dot product of nine cells in top left corner of the input with the kernel produces the top left cell of the output. The rest of the values in the output are produced as the the kernel convolves horizontally and vertically one cell at a time. This one cell movement of the kernel is called a *stride*. The value of the stride can be anything between one and the length of the kernel. Sometimes the input is zero padded so that the kernel aligns with the input as is convolves horizontally and vertically across the input.

The weight values in a kernel do not change as it convolves across the input. The height and length of feature map produced by the CONV stage of a layer is determined by the the kernel size and the stride. However, the depth of the feature map is determined by the architecture designer. The depth of the convolution dictates how many filters there are in what's called a filter bank. The rational for this design is that different filters in the filter bank of the convolution can learn different characteristics of the input. The fact that different filters in a filter bank learn different features of the input image and also that higher layers of the model learn features by using features learned by lower layers was demonstrated by Fergus *et al.* [29] using a *Deconvolutional Network* that maps activations back to the input pixel space.

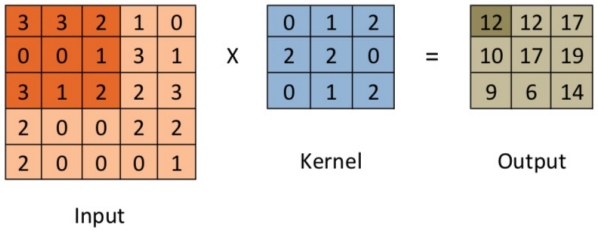


Figure 4: Dot product of the *3 X 3* kernel with an input of dimension *5 X 5* produces an output of *3 X 3* dimension. Dot product of the kernel with nine cells in top left corner of the input produces the output in top left cell of the output. *This figure is best viewed in color.*

The CONV stage in a CNN is followed by the ReLU stage that introduces non-linearity into the model by applying $max(0, x)$ element-wise to the activations produced by kernels in the convolution layer. Introducing non-linearity with ReLU ensures that the CNN model does not collapse into a large linear model [13, 12, 18]. A CNN model (and neural networks in general) can therefore be thought of as a nonlinear generalization of a linear model, which by introducing the nonlinear transformation greatly enlarges the class of the linear model [12]. This feature of CNN allows it to fit the model using a non-linear function that is more suitable (in terms of lower test error) for the domain represented by the training dataset. Use of the the $max(0, x)$ non-linear function

results in accelerated training as compared to logistic or hyperbolic tangent function as explained by Krizhevsky *et al.* [17]. The gradient in the saturating part of logistic or hyperbolic tangent activation functions becomes very small (for very large or very small weights) resulting in sluggish training because of minuscule weight change during training.

The ReLU stage in the CNN layer is followed by the pooling (POOL) stage. Activations from the ReLU are downsampled by the POOL stage. As shown in Figure 4, pooling layer downsamples the activations spatially and is usually done either by averaging or by computing the max over a small window of activations produced by the ReLU stage. Downsampling further reduces the number of parameters in the model and hence the amount of computation in the network. It also results in small shift invariance, since a small amount of change in the input will not change the output of a CNN layer, irrespective of whether average or max pooling is used. The use of pooling can be viewed as adding a strong prior that the function the layer learns must be invariant to small translations [11]. Because of the downsampling at this stage, there's a progressive reduction in size of the feature map generated by layers of the CNN, as shown in Figure 3.

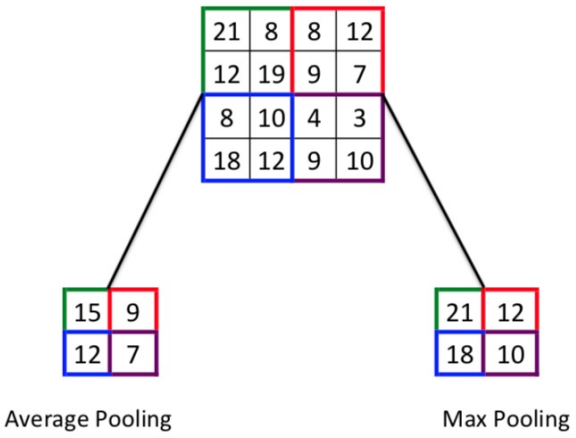


Figure 5: Pooling is used to create summary of activations generated by the ReLU layer. This figure shows two types of pooling commonly used in CNN. In average pooling, the average value of some output activations are used as a representative and the maximum value is used in max-pooling. *This figure is best viewed in color.*

Forward pass

As shown in Figure 3, each CNN layer transforms one volume of activations into another volume during the forward pass of the processing and the system makes a prediction at the rightmost layer on the model. As discussed above, these activations are generated by doing a dot product of the kernel with part of the input as the kernel convolves over the input. In this scheme, the weight values in a kernel do not change as it convolves across the input. This is a critical design decision of

CNNs, because this scheme ensures that any useful features identified in one part of the image can be re-used everywhere else without having to be independently learned, as explained by Murphy [18]. Using the kernel in such a fashion introduces a strong prior probability distribution over the parameters of a layer, which warrants that the weights for one hidden unit must be identical to the weights of its neighbor but shifted in space [11].

Model Training using Back-Propagation

Weights in the filters are the parameters of a CNN model and model fitting involves learning these weights using a method called back-propagation. Back-propagation is an optimization algorithm where the goal is to find a set of values for model parameters that generalizes the learning in such a way that the model achieves superior classification or regression accuracy on data that it has not seen before. One of the ways to achieve this goal is by minimizing the error on training dataset by using gradient descent (or its variant) method for optimization. In this mechanism many forward and backward passes are made using mini-batches of training data. A mini-batch is set of input data taken together for making several forward passes and using the average of their error in the back-propagation process for weight adjustments. Use of mini-batches in gradient descent accelerates the training and also helps to obtain an unbiased estimate of the gradient by taking the average gradient on a mini-batch.

Activations are generated in the forward pass and the error is calculated by measuring the difference between prediction and target using an error function at the last layer of the model. Gradient descent is used to nudge the prediction close to the target by adjusting weights of the model in the backward pass from the output to the input layer using chain rule of derivatives. Values of the parameters of the model are adjusted in small increments, over many iterations, with the opposite sign of the derivative.

The hope that a model that has low training error will also have small test error is based upon the assumption that the examples in each dataset are independent from each other, and that the training set and test set are identically distributed, drawn from the same probability distribution, in other words examples in training and test datasets are drawn in identically and independently distributed (IID) fashion.

Validation Set and Hyper-Parameter Tuning

Like most statistical learning models, CNNs also use hyper-parameters to fine-tune the model. The objective of using optimally calibrated hyper-parameters is to minimize the test error of the model. This ensures that the model learns good generalizations and does not just memorize patterns it sees in the training dataset — a problem called *overfitting*. Two commonly used hyper-parameter of the model are *learning rate* and *weight decay*. Learning rate determines how big or small adjustment to make to model parameter values during the back-propagation process. A high learning rate may help decrease the model error fast, but may also result a sub-optimal solution. A low learning rate on the other hand may achieve optimal model fitting, but training the model may take a long time. A validation set is used in this case to find a learning rate that provides a good compromise between training time and model accuracy. A validation set is a dataset that is reserved to test model accuracy as the model is trained using different hyper-parameters. In this case the training learning rate that results in best accuracy on the validation set is chosen as the final learning rate. Depending upon the availability of data, the validation set may be an independent dataset or a subset of the test dataset. The selection of the weight decay hyper-parameter is discussed in next section where I explain the issue of overfitting the model and how to address it using regularization.

Overfitting and Regularization

Because of their large numbers of parameters, CNN models have a proclivity to overfit when not designed with suitable regularization parameters or when there is not sufficient data to train the model. Overfitting results in a model that shows high variance when the model is trained on different training datasets (Hastie *et al.* [13]). Because of overfitting in a CNN, the training fails to achieve good model generalizations resulting in good training accuracy, but poor performance on test datasets. Bias in a model is another source of prediction error which is usually caused by wrong selection of model for the task. In case of overfitting, variance rather than bias dominates the estimation error. Regularization addresses the problem of overfitting by reducing variance significantly at the cost of increasing the bias only a little bit (Goodfellow *et al.* [11]).

L_1 and L_2 are two commonly used regularization techniques in machine learning in general, including neural networks. Both of these are ways to introduce parameter norm penalty (the regularization term in the error function) to the error function. If $J(w; X, y)$ represents the original

error function, where w represents parameters of the model, X represents training data and y represents expected output vector for the training data, then the following represents error function with the regularization term

$$\widetilde{J}(w; X, y) = J(w; X, y) + \alpha\Omega(w)$$

Here $\alpha\Omega(w)$ is the regularization term. The regularization term limits the capacity of the model by ensuring that model parameters values do not become very large or very small between iterations during training. α in the regularization term is the weight hyper-parameter that determines contribution of the norm penalty term Ω relative to the error function. L_2 regularization (also called weight decay) drives the model weights closer to the origin by adding a regularization term $\Omega(w) = 1/2 \|w\|_2^2$ to the error function. In L_1 regularization $\Omega(w)$ is calculated as the sum of absolute values of the individual parameters, $\|w\|_1$.

The dropout technique proposed by Srivastava *et al.* [26] is another regularization technique used specifically with neural networks. The basic design of using dropout involves dropping some neural network units (along with their incoming and outgoing connections) while training. This results in a smaller network that is less prone to overfitting. In each training step, whether or not a neural network unit is retained in the network is probabilistically determined. This setup effectively creates many different training models that are derived from the original model, but are much smaller. At the time of testing however, the entire network is used for the prediction and this technique effectively results in "averaging" weights of all training models and thereby reduces variability in the model.

In my implementation of a CNN for the "localizing little landmarks" task, I explored L_1 and L_2 regularizations as well as dropout. I report my findings in the experiments chapter 6. Singh *et al.* [25] use L_2 regularization as I will explain in section 4.3.

2.2 Object Detection in R-CNN (Regions with CNN features)

R-CNN [9], a model proposed by Girshick *et al.*, was one of the first approaches to show that a model based on CNN can be effectively used for object localization in an image (Figure 6). In this approach, Girshick *et al.* generate 2000 region proposals (using selective search [27]) for the input

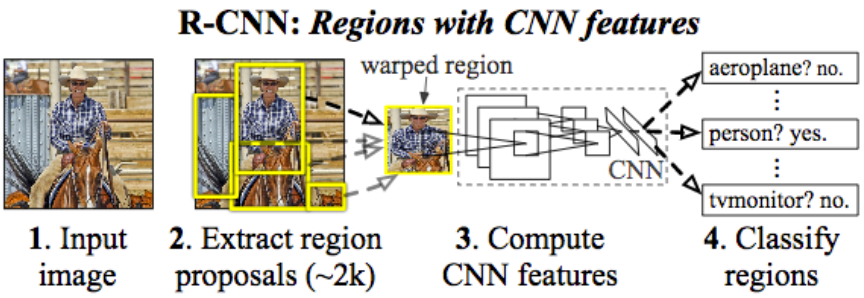


Figure 6: R-CNN [9] process for object localization. *This figure is best viewed in color.*

image and then use the CNN model of Krizhevsky *et al.* [17] to generate 4096 dimensional feature maps for these regions. These activations are then used for region classification using a Support Vector Machine (SVM).

While R-CNN has reported impressive performance for prominent object detection, it has not been very successful in detecting small objects in an image. I discuss some variants of R-CNN technique that other researchers have used for small object localization [3, 6] in the next chapter.

Chapter 3

Related Work

The problem of localization of small objects in an image using CNNs has not been widely studied. Chen *et al.* [3] extended the R-CNN algorithm to detect small objects in an image, such as a computer mouse on a desk, or a faucet in a kitchen. They used a modified Region Proposal Network [22] by choosing object proposals many times smaller than used in the original Region Proposal Network. To use context information, they also cropped a region centered at the region proposal, but bigger than the region proposal. The region proposal and the context proposal are fed to two parallel CNNs and their concatenated activation are used as an input to a third CNN to make predictions. Chen *et al.* used intersection over union (IoU) as a performance metric, whereas in my work I use the Euclidian distance between original and predicted object coordinates normalized by the bounding box of the object, as I will explain in section 4.3.

Another important work for small object identification was done by Eggert *et al.* [6]. These authors modified the Faster R-CNN model [22] to leverage higher-resolution feature maps for brand logo detection. Their work qualifies as small object detection, since they are trying to locate small brand logos in pictures such as images of a soft drink brand in a picture taken at an outdoor concert venue, or images of a sport brand on a person's shirt. In their work they attempt to generate better region proposals and assume a perfect classifier. They compare performances of region proposal generators using activations from the *conv3*, *conv4* and *conv5* layers of a pre-trained VGG-16 model and discover that the *conv3* and *conv4* layers' activations performed better than the activations of the *conv5* layer.

The research that I extend [25] in this work proposes an architecture that is recurrent in the

sense that the feature map generated by one step of the model is encoded as contextual information and is fed as input to the next step in the sequence along with the feature map generated by the convolutional layer.

Another important work that explores this idea of using contextual information is by Zuo *et al.* [30]. In their work they argue that convolutional and pooling layers in a CNN are performed locally without considering other regions of the image and therefore fail to capture contextual dependencies for better representation. They propose a model that encodes this correlation for better performance.

In this thesis I use transfer learning to demonstrate that instead of training a network from scratch, using a pre-trained network may reduce the training time significantly. Pan *et al.* [20] do an in depth study of the feasibility of transfer learning and show that the knowledge learned by a model in one domain can be transferred to another machine learning model in a different domain even when the feature space and/or the data distribution of source and target systems is not the same. Shin *et al.* [14] also employ transfer learning to fine-tune a CNN model pre-trained on a natural image dataset (RGB) to a medical image (monochrome) classification task. Transfer learning of this kind has been successfully used by numerous researchers and practitioners in image classification and localization tasks by exploiting patterns learned by deep CNN models pre-trained on large image datasets.

Chapter 4

Detecting a Small Object in an Image

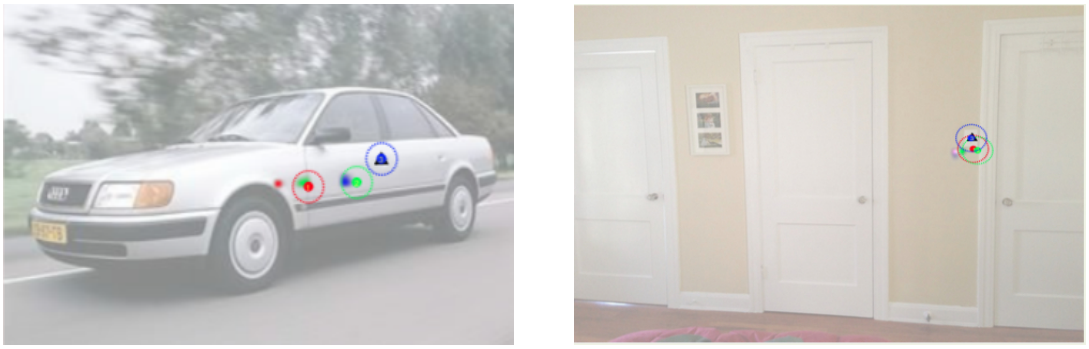
An efficient object detection model may use region proposals, as one used by Girshick *et al.* in R-CNN [9, 22]. Their two stage architecture is current state-of-the-art, where the first stage extracts region proposals and the second stage uses these region proposals as input to a CNN which is pre-trained on an auxiliary dataset. This setup has shown remarkable performance improvement for a prominent object detection in an image, but is not suitable for little landmarks, as little landmarks do not have very distinctive features. A simple solution to address the issue of lack of distinctness of little landmark in an image can be either to magnify the image or take a high resolution image. As explained by Eggert *et al.* [6], this simple approach may not be very effective since computing convolutions in a CNN grows quadratically with image dimensions and will result in unnecessary computation. Moreover, as explained by Chen *et al.* [3], low resolution inputs for small objects are deeply embedded in the nature of visual perception and a robust vision system should be able to deal with it.

This limitation of R-CNN beckons a different deep learning architecture to localize small objects in an image. Singh *et al.* [25] propose a stepwise regression model for this task and they call this model an architecture for "localizing little landmarks". They propose a recurrent model that is trained end-to-end, supervised by location of the little landmark in an image. The authors demonstrate the robustness of this model by testing and training it on several datasets. One contribution of their work was to annotate a dataset that contains images with little landmarks. The authors use Stanford's car dataset [16] for this task and annotate images in the dataset with the location of the car door handle. I use this Car Door Handle (CDH) dataset and Portland State

Dog Walking Images (DWI) dataset for training and testing our models. Details of these datasets are provided in chapter 5.

4.1 Architecture Overview of Little Landmark CNN Model

The CNN architecture for the detection of little landmarks [25] exploits the fact that a little landmark is defined by its context. Authors define a *latent landmark* as a location that is more prominent than the little landmark and can be used as context to help detect the little landmark conditionally. The localization in this architecture happens in a sequence of steps where a series of latent landmarks lead to the target little landmark by providing contextual information. Figure 7 illustrates this process with two examples. In the image on the left, the car door handle is detected



(a) Car Door Handle Localization. (b) Light Switch Localization.

Figure 7: Localization of the *little landmark* in three steps. Ground truth locations are shown as a black triangle. Step 1, Step 2 and Step 3 are color coded as Red, Green and Blue respectively. Colored blobs show the locations of the latent landmarks for each step. Solid circles show the predicted location of the next latent landmark by each step; dotted circles around solid circles show the bandwidth of the radial basis kernel used to encode the predictions. The model discovers *latent landmarks* in the first two steps that help find coordinates of the target in the third step. For the car door handle localization, in the first step, the model localizes near the front wheel of the car, gets little closer to the car door handle in the second step and lands on the target in the third step. For the light switch localization, the process starts from the edge of the light-switch panel and the model detects the light switch in three steps. Image source: [25]. *This figure is best viewed in color and with zoom.*

in a sequence of three steps; a trained network finds the first latent landmark (in red) near the front wheel, the second latent landmark in the bottom half of the front door (in green) which helps find the target location (in blue) in the third step. A similar pattern can be seen in the image on the right side of Figure 7 where the edge of the light-switch panel is used as the starting point to

localize the light switch on the wall in three steps.

Figure 8 shows a schematic diagram of the little landmark localization architecture that localizes a car door handle in three steps. Layers in this architecture have CONV and ReLU stages only. Absence of pooling stage can be attributed to the fact that since pooling results in a lower resolution feature map than the original one, it may make detection of the little landmark more difficult as it already has non-distinctive characteristics.

An image enter this system from CONV1 layer and the prediction of car door handle is made in step 3. $Conv1(3, 3, 1, 32)$ indicates a filter size of *3* x *3*, stride of 1 and a filter bank of depth 32. ReLU introduces non-linearity into the system. In this recurrent CNN architecture, convolutional layers $Conv1$ to $Conv6$ are used to extract a feature map of the input image which becomes the common input to step 1, step 2, and step 3 on the top. The three columns in the upper part of figure 8 correspond to the stepwise procedure that detects little landmark in the third step. Each prediction step s in the model generates location of the latent landmark (red blob) and predicts the location of next latent landmark (blue blob). The prediction for the next step is encoded as a feature map $P^{(s)}$ (marked as 1 and 2 in Figure 8) and along with the image feature map — output of $Conv6$ layer generated in the bottom part of Figure 8 — is used as input to the next step. The loss calculation for the model is explained later in section 4.3, but it's worth noting here that the difference between prediction (blue blob) in step *s-1* and the generated latent landmark location (red blob) in step s contributes to total error of an iteration.

4.2 Image Pre-processing

A batch of ten images is created as the input to the model. The height and width of images are adjusted so that all images in a batch have same dimensions. When necessary, images are zero padded to match the longest width and height among all images in a batch. Images are normalized by performing mean subtraction and standard deviation scaling. Random scale jitter is added to images and images are also randomly flipped to introduce noise for minimizing the problem of overfitting on training data.

A location grid, *out_locs*, is created for each batch which corresponds to spatial dimension of the output from the CONV6/RELU6 layer of the model (figure 8). This location grid is used for

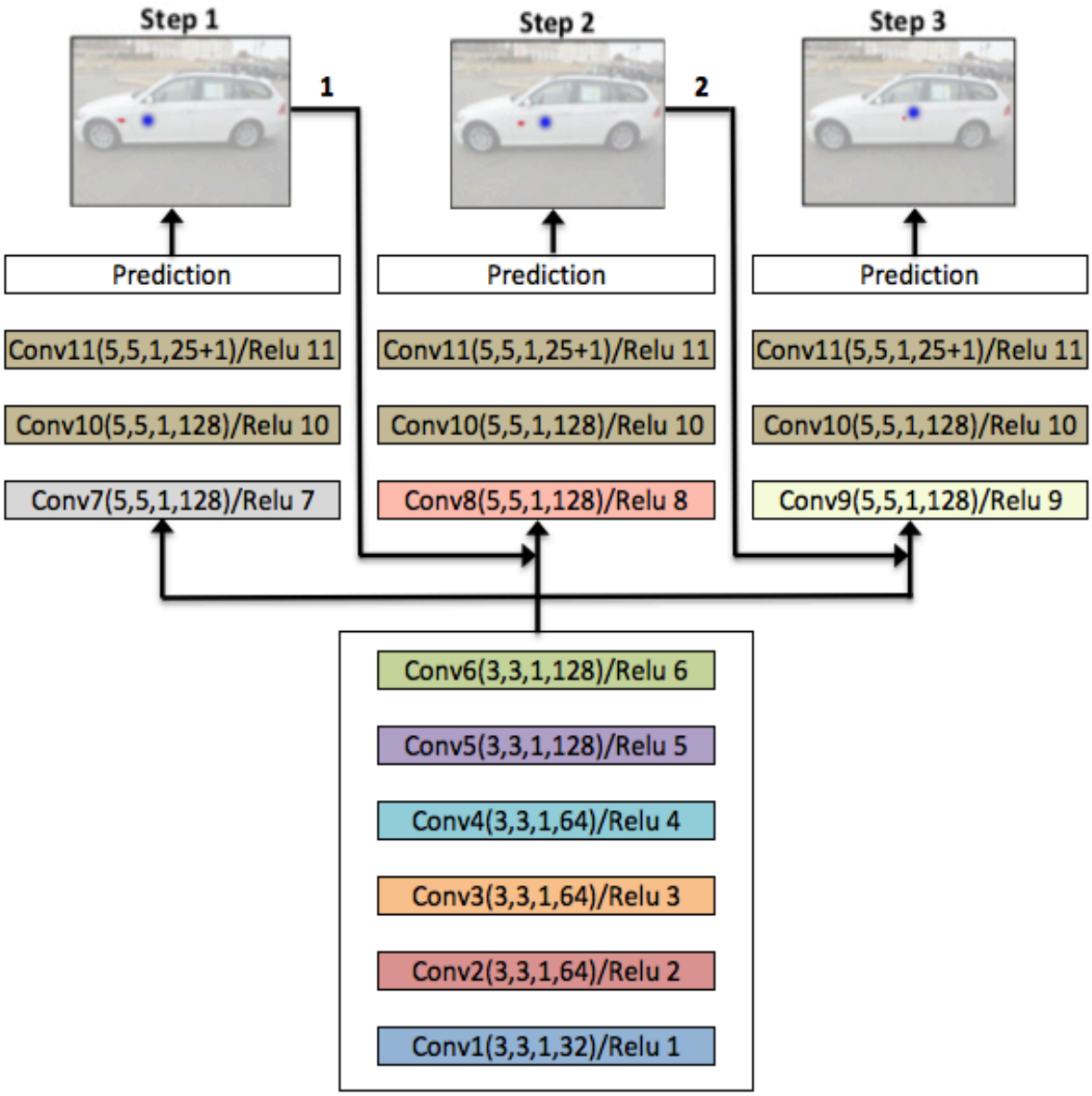


Figure 8: Little Landmark CNN Architecture. In this architecture, an image enters the system from the bottom and the final prediction for the door handle of the car is made in step 3 of the process. The input image goes thru six layers of CONV-ReLU transformations. The output from CONV6/ReLU6 layer is an input to all three prediction steps above. The red blob is the prediction for a latent landmark in that step and is used to predict the location of next latent landmark. This information is encoded as a feature map, marked as 1 and 2, with radial basis kernel (the blue blob) and is passed as a feature to the next step. Coordinates of the little landmark are predicted in the last step. CONV/ReLU boxes of the same color shows that parameters are shared in those layers. Image source: [25]. *This figure is best viewed in color and with zoom.*

generating the latent landmarks (red blobs in Figure 8) as explained in the next section.

4.3 Prediction Model

The coordinates of the target little landmark is the only supervision used in the model and the model learns to predict latent landmarks. As I explained above, each step in the model learns to find its latent landmark and predicts the location of the latent landmark for the next step. I explain this process next.

Latent and Little Landmark Centroid Generation (red blob in Figure 8)

As evident from Figure 3, the size of an activation volume decreases as an image is transformed through various layers of a CNN. The CONV11/ReLU11 layer of a step produces 26 output activation layers of size w x h, where values of w and h are determined by the width and height of images in the input batch. The first of 26 w x h output activation layers is used for predicting centroids of little and latent landmarks (red blobs in Figure 8); in steps 1 and 2 latent landmarks are predicted whereas in step 3 the little landmark (coordinates of car door handle in figure 8) is predicted. The softmax of w x h dimensional output activation is multiplied with w x h dimensional *out_locs* location grid (location grid is created during preprocessing, as explained above in section 4.2, and values of grid cells remain constant during training) to generate coordinate values of the centroid of little and latent landmarks (pc).

Latent Landmark Prediction for Next Step (blue blob in figure 8)

CONV11/ReLU11 layer of each step produces l_i (where, $i = w$ x h) activations as shown in bottom left image on right side of figure 9. These activations are used to produce an estimate of the next step's latent landmark, $P^{(s)}$ (blue blobs in figure 8 and figure 9). Each one of l_i activations influences prediction of next step's latent landmark's centroid (poc). To calculate poc, a logical grid of *5* x *5* cells is placed over each location l_i. The last 25 layers of w x h x *26* dimensional output from CONV11/RELU11 is used for this calculation. This scheme is illustrated in Figure 9.

The image on bottom left of right side of figure 9 shows l_i locations generated by CONV11/ReLU11 layer. The left side of figure 9 shows 25 logical grid points g_j (for $j \in \{1, ..., 25\}$) corresponding to a locations l_i, where $g_j(x), g_j(y) \in \{-50, -25, 0, 25, 50\}$ pixels. Grid points g_j, therefore take values in the range {(-50,-50), (-50, -25), ..., (25,50), (50,50)}, where values in the pair corresponds to x

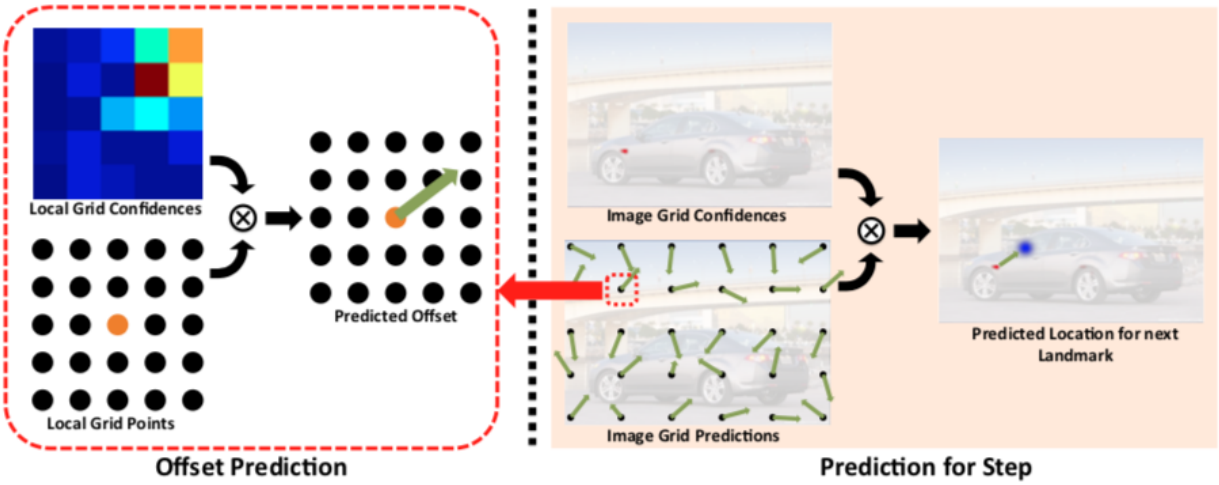


Figure 9: Latent Landmark Prediction Grid. A separate logical grid (*5* x *5* in size shown on the left) is placed over each coordinate produced by CONV6/RELU6 layer (shown on the right). The grid has fixed values for x and y coordinated of each 25 grid points. The grid values when multiplied by the confidence value generated by the network predicts the *latent landmark* for the next step. Image source: [25]. *This figure is best viewed in color.*

and y coordinates, and these values remain constant during training. The network produces 25 layers of confidence values of size w x h from the last 25 layers of the output from CONV11/RELU11 layer. These 25 layers of w x h activation values are multiplied with grid points g_j and then added together to generate x and y coordinate values for poc for each location l_i. These i coordinate values are finally multiplied by network confidence values (first layer of w x h x *26* dimensional output from CONV11/RELU11) to predict the latent landmark for the next step (this is shown as multiplication between values in top left and bottom left images on the right side in figure 9). Finally, to generate prediction $P^{(s)}$ for the latent landmark of the next step, encoding is done by placing a radial basis kernel with $\beta = 15$, centered at $pc + poc$. Prediction $P^{(s)}$ is shown as the blue blob on right side of figure 9.

Predicting Coordinates of the little Landmark

As explained above pc along with the prediction poc made by each location l_i is used for generating $P^{(s)}$. The sum $pc + poc$ predicts the centroid for the next step's latent landmark, except for the last step. In the last step, $pc + poc$ predicts the location of the little landmark — for example, coordinates of the car door handle in Figure 9.

Training (and Loss Calculation)

For my implementation of Singh *et al.*'s work I implemented their model in TensorFlow and trained it using the Adam optimizer [15]. L_2 loss between the predicted location and the ground truth was used for gradient calculation. However, there are several components in the loss value; L_2 loss between the ground truth and the location predicted by the last step of the model and the L_2 loss between latent landmarks predicted in step s and estimated in step *s-1* contribute to the total loss of the model in one iteration, which is averaged over the minibatch of ten images to perform back-propagation of error through the network. The weightage given to the step losses is less than the weightage given to L_2 loss between the ground truth and the predicted location in the last step as shown below

$$L = \sum_{s=1}^{S} \lambda_s L^{(s)} + R(\theta)$$

I used $\lambda_s = 0.1$, except for the final step S where $\lambda_s = 1$. $R(\theta)$ is a regularizer for the parameters of the network. I used L_2 regularization (weights decay) with a multiplier of 0.00005.

Incorporating losses from all steps of the model in the loss function encourages earlier steps to be informative for the later steps by penalizing disagreement between the predicted and later detected latent landmark locations. Also, encoding predictions as a feature map instead of as a rigid constraint for the next step allows it to ignore the prediction from the previous step, if necessary. This flexibility is helpful in early stages of the training when the latent landmark estimates are not very reliable.

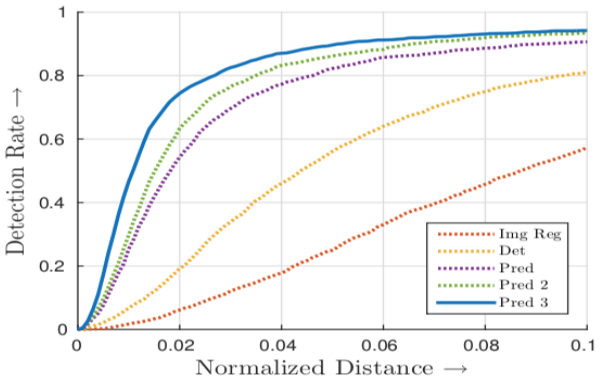
Performance Metric

To evaluate the model's accuracy, Singh *et al.* used a generally accepted metric of plotting detection rate against normalized distance [10]. The normalized distance is calculated by dividing the L_2 norm between actual and predicted locations by the height of the bounding box of the car in the car door handle dataset, and by height of the light switch panel in the light switch dataset. The detection rate is the ratio between number of images for which the little landmark was correctly located to the total number of test examples. For performance evaluation I create a plot of the

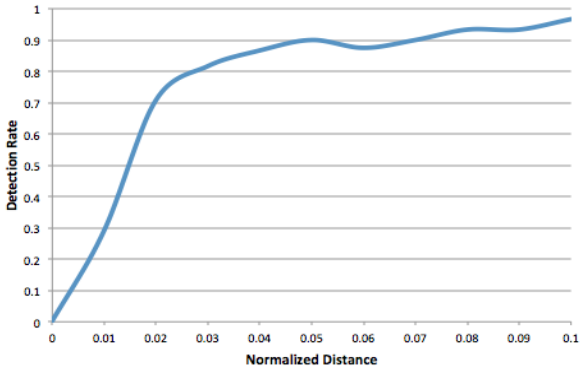
normalized distance vs. detection rate for normalized distances in the range 0.01 to 0.1 (figure 10).

Chapter 5

Methods



(a) Singh *et al.*'s result [25]



(b) Result from my TensorFlow implementation

Figure 10: Original vs. my model's test accuracy comparison. The graph in (a) shows a plot of normalized distance vs. detection rate published by Singh *et al.* The curve marked as Pred 3 shows the accuracy of the three step process. My implementation's test accuracy is plotted in (b) that shows comparable performance to the original Pred 3 curve. *This figure is best viewed in color.*

The authors of [25] provided me with their Matlab implementation for the model. I used that codebase to create my own implementation of the model using TensorFlow and Python. I tested the fidelity of my implementation by comparing my test results with results published in [25]. I used the same dataset as in [25] for training and testing the model and I found that my model performed similar to the results reported in [25]. Figure 9 shows a comparison between the models.

The plot of normalized distance vs. detection rate from my implementation is depicted in Figure 7(b). The Pred 3 plot in Figure 7(a) shows the accuracy of Singh *et al.*'s model. The Pred 2 and Pred 1 plots in Figure 7(a) are of model with two steps and one step respectively.

Datasets

Car Door Handle (CDH) Dataset

The authors of [25] provided me with car images dataset that contained annotated coordinates of the car door handle. There were 1920 training images and 1200 testing images in the dataset. All images had the coordinates of the car door handle annotated. Car bounding box details were also available.

Dog Walking Images (DWI) Dataset

I used the Portland State Dog-Walking Images dataset [2] to test the model's effectiveness on a different dataset. I used 310 images for training and 70 for testing. PSU researchers annotated the coordinates of the hand-holding-leash little landmark of the image and also the dog, dog walker and the leash bounding boxes in each image.

Training Detail

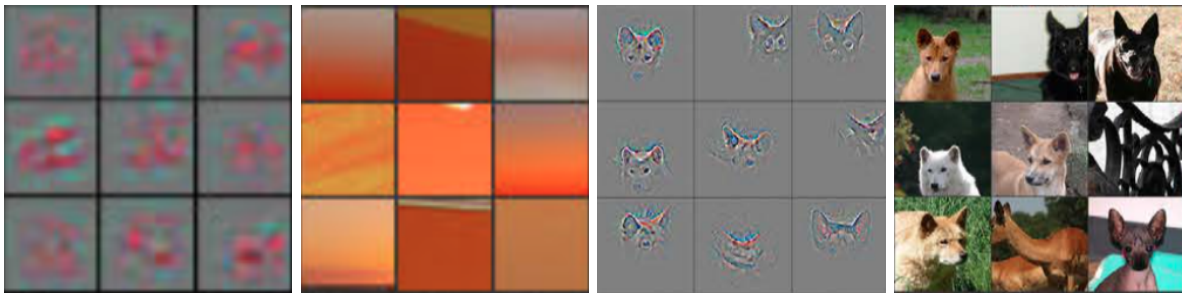
I trained my implementation of the model on above mentioned datasets. For the Car Door Handle dataset, I trained the model for 3100 epochs where each epoch was trained on a shuffled set of 1920 images in batches of ten images each. Similarly for the Dog-Walking Images dataset, I trained the model for 3200 epochs where each epoch was trained on a shuffled set of 310 images in batches of ten images each. The testing performance of these trained models is discussed in chapter 6.

5.1 Model Modifications

Transfer Learning

Fergus *et al.* [29] and many other researchers [2, 19] have demonstrated that, in a well-trained CNN model, lower layers get attuned to more generic features of an image like edges, corners or color blobs, whereas higher layers learn more specific features like a person's face or wheels of a car. Bengio [2] argues that a deep learning algorithm seeks to discover good representations at multiple layers in such a way that features learned in a higher level can be composed of features

[2] http://web.cecs.pdx.edu/ mm/PortlandStateDogWalkingImages.html



(a) (b) (c) (d)

Figure 11: Visualization of features learned by a CNN model using a Deconvolutional Network [29]. (a) shows layer 2 activations mapped to its pixel space and (b) shows image patches corresponding to (a); (c) shows layer 5 activations mapped to its pixel space and (d) shows image patches corresponding to (c). Image source: [29]. *This figure is best viewed in color.*

learned in lower layers of the network. Another important characteristic of learning in this scheme is that features learned in higher layers are invariant to variations in training distribution, such as background, viewpoint, scene context etc. Fergus *et al.* [29], by mapping activations in a CNN model back to its pixel space, experimentally demonstrate this fact as shown in Figure 11. Properties of a deep CNN discussed above make transfer learning possible where representations learned with a CNN trained on a large image dataset can be effectively used to initialize another network. Oquab et al. [19] show that, despite differences in image statistics and tasks in two different datasets, transferred representations lead to significant improvement in image detection and classification and they demonstrate that incorporation of prior knowledge via transfer learning can boost the performance of CNNs by a large margin. Because of these advantages the two most common reasons to employ transfer learning are that it can reduce training time and that it can overcome overfitting when there is a paucity of training data.

My motivation to use transfer learning was to exploit generalizations learned by the VGG-16 network [24] pre-trained on ImageNet [23] dataset to reduce the training time of for little landmark localization. VGG-16 [24] is a CNN model proposed by K. Simonyan and A. Zisserman from the University of Oxford that achieved 92.7% accuracy in top-5 test category and 70.5% accuracy in top-1 category on the Imagenet dataset. The Imagenet dataset [23] is a dataset of over 14 million images belonging to 1000 categories.

As I discussed above, the model proposed by Singh *et al.* [25] takes more than 100 hours to train. I used transfer learning to improve the training time for this model by using activations

from another state-of the-art CNN model, pre-trained on an auxiliary dataset, as the starting point for training the model. I observed significant improvement in training time in this modified implementation that used transfer learning; my implementation of the original model needed 107 hours of training and the modified model needed about 46 hours to train to achieve similar level of performance.

Another goal of my thesis was to localize the "hand-holding-leash" part of a dog-walking image. The rational was that detection of "hand-holding-leash" situation in the image will give more evidence that a person may be walking a dog in the image. This detection will help localize other objects or relationships in the image, which may help to identify the visual situation in an image [21]. To this end, I used the design proposed to detect little landmarks [25] to localize "hand-holding-leash" part of the dog walking image.

I used transfer learning to overcome the problem of overfitting because of scarce training data in the Portland State Dog-Walking Images dataset. I did encounter overfitting problem when I trained the model from scratch with limited data in dog walking images dataset.

Little Landmarks Model Modification using Transfer Learning

As I explained above, I used generalizations learned by the VGG-16 model for training the car door handle as well as the dog walking image datasets. The VGG-16 CNN architecture has sixteen weight layers and a filter of size *3* x *3* is used to convolve in all layers. The input to the network is a fixed size *224* x *224* RGB image. Mean image subtraction is the only preprocessing performed on input images to center images. I used activations from *pool4* layer of the model as input to the modified little landmarks model (Figure 12).

In the modified architecture I used a bridge layer that takes *pool4* activations from the VGG-16 model as input and performs a layer of convolutions to modify output dimensions so that it conforms with rest of the architecture. I used a publicly available pre-trained VGG-16 model [4] as the basis of my model. I used same image preprocessing technique that I used for the little landmark model I discussed in chapter 4, and fed a mini-batch of ten images to VGG-16 model, as before. Internally the VGG-16 implementation warped images to *224* x *224* x *3* before feeding it to the network.

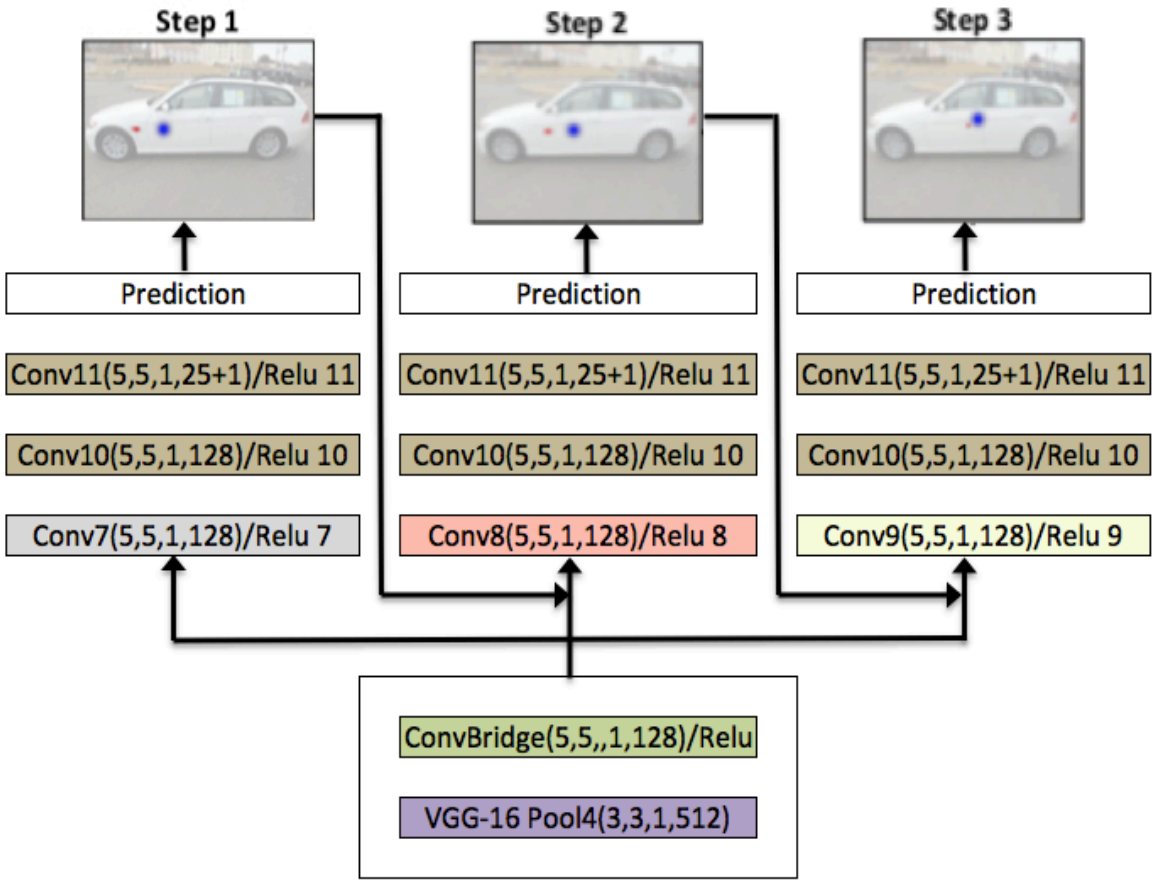


Figure 12: Little landmarks architecture modified using transfer learning. This model used features learned by the VGG-16 network as an input to rest of the model. As in figure 8, an image enters the system from the bottom and the final prediction for the door handle of the car is made in step 3 of the process. The bridge layer is use to transform output dimensions of VGG-16 model so that it dovetails with the upper part of Little Landmarks' architecture. *This figure is best viewed in color and with zoom.*

Chapter 6

Experiments and Results

In this chapter I discuss the experiments I performed and their results. Table 1 provides a comparison of detection rates between experiments for various normalized distances and figure 13 shows normalized distance vs. detection rate plots for all experiments.

Experiment 1: Transfer Learning on Car Door Handle Dataset

In this experiment I used the modified model that used VGG-16 model with transfer learning (Figure 12). The objective of this experiment was to demonstrate that the training time of the model proposed by Singh *et al.* [25] can be improved if the model's weights are initialized by weights of a pre-trained model. I trained the model on the same car door handle dataset as the original model until the performance of the modified model was comparable to Singh *et al.*ś model. During training weights of the pre-trained VGG-16 model were fixed and fine tuning was performed only for the weights in regression steps of the model. The modified model needed about 46 hours to train for 530 epochs. I observed about 57% improvement in training time of the modified model when compared to the original. Figure 13(a) shows the normalized distance vs. detection rate plot for this experiment.

Experiment 2: Transfer Learning on Dog-Walking Dataset

In this experiment I trained the same network as in experiment 1 with dog walking images. The objective of this experiment was to exploit the idea proposed by Singh *et al.* to localize on "hand-holding-leash" part of the dog-walking image. 310 training and 70 test images were used in this experiment to localize "hand-holding-leash" part of the image. I trained the model for 96 hours

Normalized Distance	Detection Rate					
	Original	Experiment1	Experiment2(a)	Experiment2(b)	Experiment2(c)	Experiment3
0.01	0.29	0.28	0.00	0.00	0.00	0.01
0.02	0.71	0.49	0.00	0.00	0.00	0.02
0.03	0.81	0.73	0.00	0.00	0.00	0.12
0.04	0.86	0.83	0.00	0.00	0.29	0.23
0.05	0.90	0.91	0.00	0.14	0.43	0.44
0.06	0.88	0.94	0.14	0.00	0.29	0.46
0.07	0.90	0.94	0.14	0.00	0.57	0.65
0.08	0.93	0.95	0.14	0.14	0.43	0.66
0.09	0.93	0.97	0.43	0.29	0.71	0.75
0.10	0.97	0.98	0.43	0.29	0.71	0.75

Table 1: This table displays test accuracies of various models discussed in chapter 5. Experiment 1 was performed on the model that used transfer learning (figure 12 shows architecture of this model). Test accuracy of this model was similar to the original model proposed by Singh *et al.* [25]. Experiment 2(a) shows test accuracy of the model depicted in figure 12, when the model is trained on dog-walking images; in this case activations from *pool4* layer of a pre-trained VGG-16 [24] model were used. Experiment 2(b) shows test accuracy of a model where a five-step training was used instead of three, on dog-walking images. Experiment 2(c) was similar to experiment 2(a) except for the fact that activations from *pool3* layer of VGG-16 model [24] were used. Model in experiment 3 was trained on the car-door-handle dataset, but the training used same size of data as in experiment 2(a).

and observed near perfect training accuracy. However, I observed very poor test accuracy as shown in Figure 13(b).

The model appeared to overfit by memorizing features of training images and it did not seem to learn any general pattern for localization. I trained the model with weight loss values of 0.005 and 0.0005 also in addition to 0.00005, which was used for L_2 regularization in the original model. I also experimented with L_1 regularization. None of these method resulted in any performance improvement. I also used dropout [26] for regularization, but did not see any performance improvement.

Since context plays an important role in little landmark's localization I changed the model to localize in five steps instead of three. My rational for this approach was that a model localized in five steps may pick up better context information which may result in better generalization. However, this approach also did not work. Plot of this experiment is shown in figure 13(c).

In another attempt to improve performance I used activations from *pool3* layer of the VGG-16 model. The rational for this approach was that, since latent and little landmarks are both not very distinctive, using filters from a lower layer, that may be more attuned to features of latent and little landmarks, may show better performance. Plot of this experiment is shown in figure 13(d).

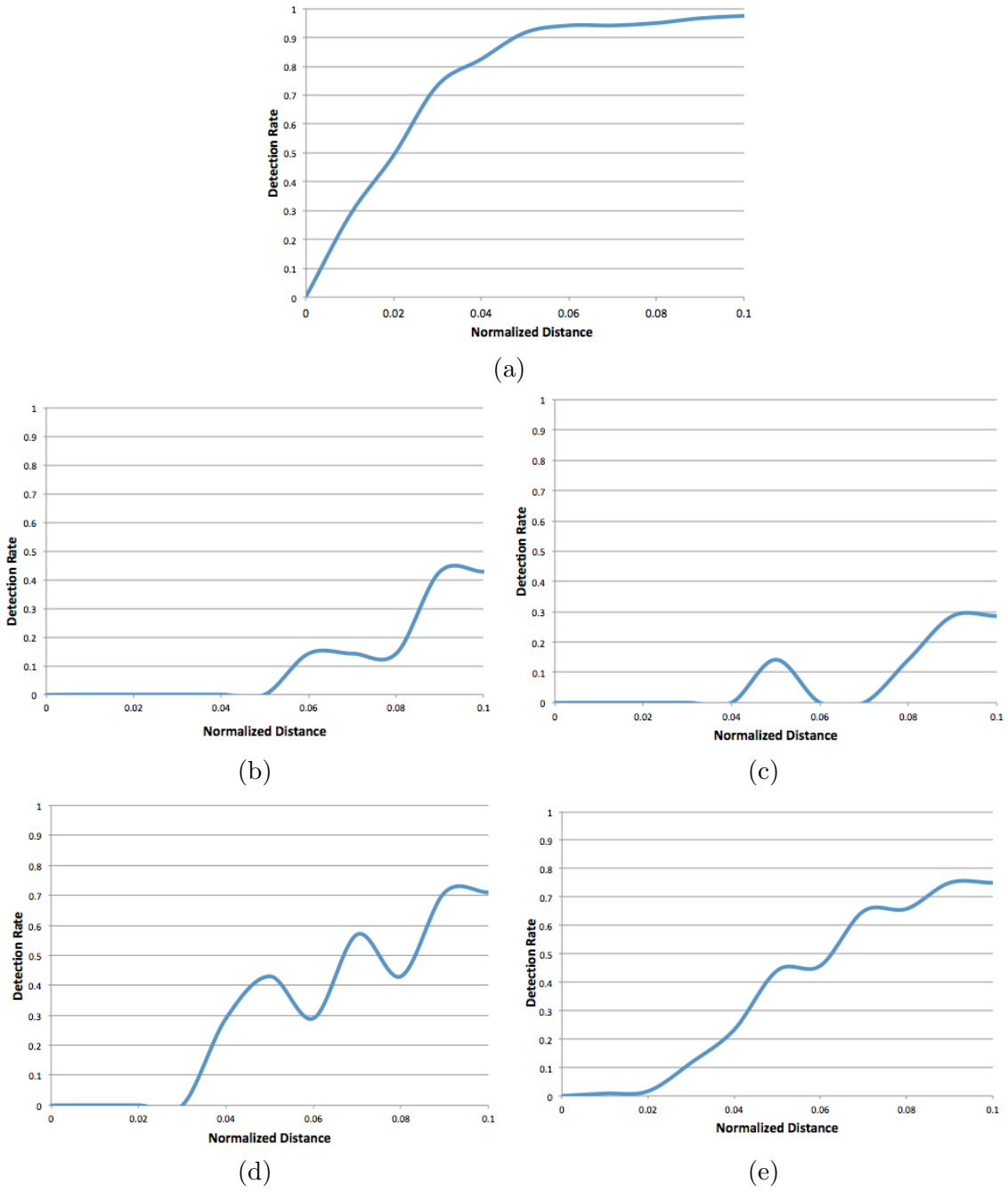


Figure 13: Normalized distance vs. detection rate plots; all plots show model accuracy for normalized distance in the range 0.01 to 0.1. Plot (a) is created for the model depicted in figure 12, that took 57% less time to train than the original model (figure 8) on car door handle dataset. (b) shows performance of figure 12 model on dog-walking images to identify "hand-holding-leash" relationship. (c) represents accuracy of a model that was trained to localize on "hand-holding-leash" relationship in five steps, instead of three. (d) represents performance of figure 12 model when trained using *pool3* layer of the VGG-16 network, instead of *pool4* that was used in (b). (e) represents performance of the figure 8 model, but trained on same size of data as in (b).

Experiment 3

In the last experiment I trained the original TensorFlow model that I discussed in section 4.3, but with same amount of training data that was used for experiment 2. I anticipated that training the original little landmark model with small amount of data may render the model not as effective. For about the same duration of training that I used for the model discussed in section 4.3, I did notice inferior performance in this experiment, but the performance was not as poor as in experiment 2. Figure 13(e) shows normalized distance vs. detection rate plot for this experiment.

Chapter 7

Conclusions and Future Work

In this chapter I discuss the results of experiments from the previous chapter and draw conclusion about efficacy of various models discussed above for localization of a small object in an image. I also suggest future extensions to methods discussed in this thesis that can be used to make detection of a small object in an image more effective.

7.1 Discussion

As I discussed in the introduction, little landmarks in an image do not have very distinctive features and the use of context is an effective technique for their detection. I demonstrated this (experiment 1) by implementing the architecture originally proposed by Singh *et al.* As I explained in chapter 4, I trained and tested this model on the car door handle dataset and observed model performance similar to reported by Singh *et al.* [25]. In another implementation I modified Singh *et al.*'s model [25] by incorporating transfer learning (experiment 1, chapter 6). In this implementation I used a pre-trained VGG-16 model [24] to initialize weights of the model. By using transfer learning I was able to achieve about 57% improvement in training time without performance degradation of the model. The normalized distance vs. detection accuracy plot of this experiment is shows in figure 13(a). The result of this experiment demonstrated that generalizations learned by a well trained network can be exploited to reduce the training time of another model.

In another experiment (experiment 2(a)) I attempted to exploit transfer learning to overcome the problem of overfitting because of insufficient training data in the Portland State Dog Walking Images dataset. Although many researchers [19, 1, 5] have been successful in using similar transfer

learning methods to make up for small datasets, but this approach was not effective for localizing "hand-holding-leash" relationship in dog-walking images. Plot 13(b) shows detection rate of a model trained for experiment 2(a) for about 50 hours for different normalized distances. As explained in the previous chapter since location of small objects can be determined by their context, I performed another experiment (experiment 2(b)) where I tried to localize the "hand-holding-leash" relationship in dog-walking images in five steps instead of three. This method also did not result in performance improvement as is evident from the plot in figure 13(c). Results of experiments explained above indicated that although transfer learning may result in faster model training, but availability of more training data is still important to avoid overfitting. To test this theory I performed experiment 2(d) where I used the car door handle dataset to train the model represented in figure 8, but with same size of data as I used in experiment 2(a). The plot of this experiment is shown in figure 13(e). The performance of localizing door handle of the car in this experiment was inferior as compared to experiment 1 (plot in figure 13(a)) — underlining the fact that availability of more data for training is essential to overcome the problem of overfitting — however, the detection rate in this experiment was still much better that the detection rate in experiment 2(a) (plot in figure 13(b)). I performed one more experiment, the plot of which is shown in figure 13(d). In this experiment I used activations from *pool3* layer of the VGG-16 network for transfer learning, instead of *pool4* layer that I used in experiment 2(a). The rationale for this approach was that, since little landmarks are not characterized by their distinct features, activations from a lower layer on CNN (attuned to more generic features of an image) may perform better. The plot of this experiment is erratic and no conclusion about model accuracy can be drawn from this experiment.

Based upon the discussion above, I now explain why the approach that worked for identifying the handle of car doors in car images did not work for localizing "hand-holding-leash" relationship

Figure 14: Representative Images from Car Door Handle Dataset. Image source [25]. *This figure is best viewed in color.*

Figure 15: Representative Images from Dog Walking Images Dataset. *This figure is best viewed in color.*

in dog-walking images.

The comparison of results between experiment 1 and experiment 3 (Table 1) demonstrates that lack of data does impact a model accuracy. Experiment 1 as well as experiment 3 were performed on Singh *et al.*'s model (figure 8); whereas 1920 training images were used in experiment 1, only 310 training images were used in experiment 3. The model was trained for the same amount of time in both experiments, but the performance of the model in experiment 3 is inferior to experiment 1, underscoring the importance of availability of data. Additionally, comparison between results of experiment 2(a) and experiment 3 shows that lack of training data resulted in much worse performance in dog-walking images as compared to cars images. Figure 14 shows some representative images of cars used in the training to localize the door handle of a car. In all these images location of the door handle has such a regular spatial configuration that latent landmarks can be consistently found; in other words, the context for little and latent landmarks are consistent across images. I discovered that in most cases the first latent landmark localized either near the front or the rear wheel of the car, the second latent landmark localized somewhere on the door of the car, which helped localize the door handle of the car. Whereas in dog-walking images (Figure 15), the context around the "hand-holding-leash" landmark is not consistent between images. Dog-walking images have varied background and settings and as a result latent landmarks cannot be localized effectively in first and second step of the model. Figure 15 shows dog-walking images in four different settings and it's apparent from these representative images that latent landmarks in dog-walking images cannot be found with same consistency as in the case of car images, which is the main reason why localization of "hand-holding-leash" did not perform as well as the localization of the door handle of a car did.

7.2 Future Work

A hallmark of CNN models is their ability to learn invariant features of an image so that the prediction or classification accuracy is not impacted by variability in an image. Xu *et al.* [28] explain that representations in very top layer of a CNN distills information in an image down to the most salient objects. However this property of CNN is not very useful for tasks like small object detection since they are defined more by the context. Xu *et al.* propose an attention based model [28] where they argue that using low level representations may help preserve the semantic information in an image. In their work they draw a parallel between this approach and presence of attention in the human vision system. They use this insight to create a Long Short Term Memory (LSTM) based CNN system that automatically learns to describe the content of images. I argue that a similar approach of using attention based model can be explored for identification for small objects in an image.

The variant of transfer learning that I used is supervised transfer learning where the source network [3] was trained with supervision. Another form of transfer learning uses unsupervised pre-training, which is a possible topic for future work. Unsupervised pre-training restricts the model parameters to regions that correspond to capturing structure in the input distribution. Erhan *et al.* [7] explain that greedy layer-wise unsupervised pre-training overcomes the challenges of deep learning by introducing a useful prior to the supervised fine-tuning training procedure, which effectively puts parameter values in an appropriate range for further supervised training. Use of this approach makes unsupervised learning a special kind of regularizer that minimizes the variance at the cost of introducing some bias in the model. This is another promising method for small object localization specially in cases where training data is scarce and context around the small object is not consistent.

Another possible topic for future work is the development of a better metric to measure the effectiveness of small object detection. Chen *et al.* [3] point out that one of the challenges of small object identification is that we do not know how difficult this task is or how well existing object detectors work. I found that the metric used for measuring performance of small object detection is not consistent between research groups — some researchers use Intersection over Union (IoU),

[3] *source network* in this case is the pre-trained network whose weights are used to initialize the model under development.

whereas others use some sort of normalization for Euclidian distance between the prediction and the ground truth. Because of this it's difficult to compare performance of different small object localization models and a metric that can be used for such comparison would be very useful tool in this area of work.

Bibliography

[1] Amr Ahmed et al. "Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks". In: *European Conference on Computer Vision*. Springer. 2008, pp. 69–82.

[2] Yoshua Bengio. "Deep learning of representations for unsupervised and transfer learning". In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. 2012, pp. 17–36.

[3] Chenyi Chen et al. "R-cnn for small object detection". In: *Asian conference on computer vision*. Springer. 2016, pp. 214–230.

[4] Chris Machrisaa. *VGG-16 TensorFlow*. https://github.com/machrisaa/tensorflow-vgg. 2017.

[5] Ronan Collobert et al. "Natural language processing (almost) from scratch". In: *Journal of Machine Learning Research* 12.Aug (2011), pp. 2493–2537.

[6] Christian Eggert et al. "Improving small object proposals for company logo detection". In: *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ACM. 2017, pp. 167–174.

[7] Dumitru Erhan et al. "Why does unsupervised pre-training help deep learning?" In: *Journal of Machine Learning Research* 11.Feb (2010), pp. 625–660.

[8] Sanjay Surendranath Girija. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: (2016).

[9] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[10] Afzal Godil et al. *Performance Metrics for Evaluating Object and Human Detection and Tracking Systems*. Tech. rep. 2014.

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

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2008.

[13] Trevor Hastie et al. *An Introduction to Statistical Learning*. Springer, 2014.

[14] Shin Hoo-Chang et al. "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning". In: *IEEE transactions on medical imaging* 35.5 (2016), p. 1285.

[15] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[16] Jonathan Krause et al. "3d object representations for fine-grained categorization". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2013, pp. 554–561.

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

[18] Kevin Murphy. *Machine Learning*. MIT Press, 2012.

[19] Maxime Oquab et al. "Learning and transferring mid-level image representations using convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1717–1724.

[20] Sinno Jialin Pan, Qiang Yang, et al. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2010), pp. 1345–1359.

[21] Max H Quinn et al. "Semantic Image Retrieval via Active Grounding of Visual Situations". In: *Semantic Computing (ICSC), 2018 IEEE 12th International Conference on*. IEEE. 2018, pp. 172–179.

[22] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[23] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[24] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[25] Saurabh Singh, Derek Hoiem, and David Forsyth. “Learning to Localize Little Landmarks”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 260–269.

[26] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[27] Jasper RR Uijlings et al. “Selective search for object recognition”. In: *International journal of computer vision* 104.2 (2013), pp. 154–171.

[28] Kelvin Xu et al. “Show, attend and tell: Neural image caption generation with visual attention”. In: *International conference on machine learning*. 2015, pp. 2048–2057.

[29] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[30] Zhen Zuo et al. “Convolutional recurrent neural networks: Learning spatial dependencies for image representation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2015, pp. 18–26.