KRYPTON: Accelerating Occlusion based Deep CNN Explainability Workloads

Anonymous Author(s)

ABSTRACT

Deep Convolution Neural Networks (CNNs) have revolutionized the field of computer vision with even surpassing human level accuracy in some of the image recognition tasks. Thus they are now being deployed in many real-world use cases ranging from health care, autonomous vehicles, and e-commerce applications. However, one of the major criticisms pointed against Deep CNNs is the black-box nature of how they make predictions. This is a critical issue when applying CNN based approaches to critical applications such as in health care where the explainability of the predictions is also very important. For interpreting CNN predictions several approaches have been proposed and one of the widely used method in image recognition tasks is occlusion experiments. In occlusion experiments one would mask the regions of the input image using a small gray or black patch and record the change in the predicted label probability. By changing the position of the occlusion patch, a sensitivity map can be generated from which the regions in the input image which influence the predicted class label most can be identified. However, this method requires performing multiple forward passes of CNN inference for explaining a single prediction and hence is very time consuming. We present KRYPTON, the first data system to elevate occlusion experiments to a declarative level and enable database inspired automated *incremental* and *approximate* inference optimizations. Experiments with real-world datasets and deep CNNs show that KRYPTON can enable speedups over 10X in certain cases.

ACM Reference Format:
Anonymous Author(s). 2018. KRYPTON: Accelerating Occlusion based Deep CNN Explainability Workloads. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

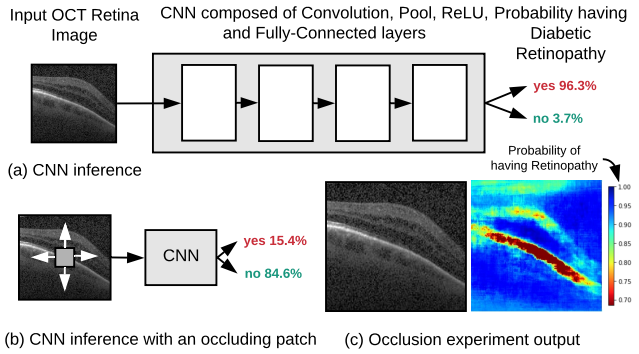


Figure 1: (a) Using CNNs for predicting Diabetic Retinopathy from OCT images. (b) Occluding parts of the OCT image changes the predicted probability for the disease. (c) By changing the position of the occlusion patch a sensitivity heat map is produced.

1 INTRODUCTION

Deep Convolution Neural Networks (CNNs) [1–4] have revolutionized the computer vision field with even surpassing human level accuracy in some of the image recognition challenges such as ImageNet [5]. As a result, there is wide adoption of deep CNN technology in a variety of real world image recognition tasks in several domains including healthcare [6, 7], agriculture [8], security [9], and sociology [10]. Remarkably, United States Food and Drug Administration Agency (US FDA) has already approved the use of deep CNN based technologies for identifying diabetic retinopathy, an eye disease found in adults with diabetes [11]. It is expected that this kind of decision support systems will help the human radiologists in fulfilling their workloads efficiently, such as operating as a cross-checker for the manual decisions and also to prioritize potential sever cases for manual inspection, and provide a remedy to the shortage of qualified radiologists globally [12].

However, despite their many success stories, one of the major criticisms for deep CNNs, and deep neural networks in general is the black-box nature of how they make predictions. In order to apply deep CNN based techniques in critical applications such as health care, the decisions should be explainable so that the practitioners can use their human judgment to decide whether to rely on those predictions or not [13].

In order to improve the explainability of deep CNN predictions several approaches have been proposed. One of the most widely used approach in image recognition tasks is occlusion experiments [14]. In occlusion experiments, as shown in Figure 1 (b), a square patch usually of black or gray color is used to occlude parts of the image and record the change in the predicted label probability. By changing the position of the occlusion patch, usually by a small fixed number of pixels called stride, a sensitivity heat map for the predicted label can be generated (similar to one shown in Figure 1 (c)). If the occlusion experiment is performed in interactive mode, the human operator has the option of picking the occlusion patch positions by marking a region on a visual interface. For example, if the scenario shown in Figure 1 is performed in interactive mode, a human operator who understands OCT images will start evaluating the image from the central region where she expects the pathological region to be. In the non-interactive mode, which is also the most common mode of performing occlusion experiments due to the high runtimes which are not amenable for interactive performance, the heat map values are evaluated for all possible occlusion patch positions. Using this heat map, the regions in the image which are highly sensitive (or highly contributing) to the predicted class can be identified (corresponds to red color regions in the sensitivity heat map shown in Figure 1 (c)). This localization of highly sensitive regions then enable the practitioners to get an idea of the the prediction process of the deep CNN.

However, occlusion experiments are highly compute intensive and time consuming as each occlusion position has to be treated as a new image and requires a separate CNN inference. In this work, our goal is to apply database inspired optimizations to the occlusion based explainability workload to reduce both the computational cost and the runtime. This will also make occlusion experiments more amenable for interactive diagnosis of CNN predictions. Our main motivation is based on the observation that when performing CNN inference corresponding to each individual patch position, there is a significant portion of redundant computations which can be avoided. To avoid redundant computations we introduce the notion of *incremental inference* of deep CNN which is inspired by the incremental view maintenance technique studied in the context of relational databases.

Due to the overlapping nature of how the Convolution kernel operates (details to follow in Section 3), the size of the modified patch will start growing as it progresses through more layers in a CNN and the amount of redundant computations will reduce. However, at deeper layers, the effect over the patch coordinates which are radially further away from the center of the occlusion patch position will be diminishing. Our second optimization is based on this observation where we apply a form of *approximate inference* which applies a threshold to limit the growth of the updating patch. By applying propagation thresholds, a significant amount of computation redundancy can be retained. We refer to this optimization as *projective field thresholding*.

The third optimization is also a form of *approximate inference* which is applicable only in the context of non-interactive mode. In most occlusion experiment use cases, such as in medical imaging, the object or pathological region of interest is contained in a relatively small region of the image. In such situations, it is unnecessary to inspect the original image at the same high resolution of striding the occluding patch few pixels at a time, at all possible occlusion patch positions. In this approach first, a low-resolution heat map is generated using a larger stride value with a relatively low computational cost. Only the interesting regions will be then inspected further with a smaller stride to produce a higher resolution output. In the interactive mode as the human operator will be actively picking a set of occlusion patch positions for the system to evaluate this optimization will not be applicable. We refer to this optimization as *adaptive drill-down*.

Unlike the *incremental inference* approach which is exact, *projective field thresholding* and *adaptive drill-down* are approximate approaches. They essentially trade-off accuracy of the generated sensitivity heat map compared to the original, in favor of faster execution. These changes in accuracy in the generated heat map will be visible all the way from quality differences which are almost indistinguishable to the human eye to drastic structural differences, depending on the level of approximation. This opens up an interesting trade-off space of quality/accuracy versus runtime. Krypton provides user configurable tuning parameters for easily picking an operational point on this quality-runtime trade-off space.

Finally, we have implemented Krypton on top of PyTorch deep learning toolkit by adding custom implementations for incremental and approximate inference operations. It currently supports VGG16, ResNet18, and InceptionV3 both on CPU and GPU environments, which are three widely used deep CNN architectures. We evaluate our system on three real-world datasets, 1) retinal optical coherence tomography dataset (OCT), 2) chest X-Ray, and 3) more generic ImageNet dataset, and show that Krypton can result in speedups over 10X. While we have implemented Krypton on top of PyTorch toolkit, our work is largely orthogonal to the choice of the deep learning toolkit; one could replace PyTorch with TensorFlow, Caffe2, CNTK, MXNet, or implement from scratch using C/CUDA and still benefit from our optimizations. Overall, this paper makes the following contributions:

- To the best of our knowledge, this is the first paper to study

Outline. The rest of this paper is organized as follows.

2 BACKGROUND

Deep CNNs. Deep CNNs are a type of neural networks specialized for image data. They exploit spatial locality of information in image pixels to construct a hierarchy of parametric feature extractors and transformers organized as layers of various types: *Convolution*, which use image filters from graphics, except with variable filter weights, to extract features; *Pooling*, which subsamples features in a spatial locality-aware way; *Batch-Normalization*, which normalizes the output of the layer; *Non-Linearity*, which applies a non-linear transformation (e.g., ReLU); *Fully Connected*, which is the building block of a multi-layer perceptron; and *Softmax*, which emits predicted probabilities to each class label. In most deep CNN architectures, the above layers are stacked together with oneś output being fed as the input to the other. Adding multiple layers element-wise or stacking multiple layers together depth-wise to produce a new layer is also present in some architectures. Popular deep CNN model architectures include AlexNet [1], VGG [2], Inception [4], ResNet [3], SqueezeNet [15], and MobileNet [16].

Deep CNN Explainability Various approaches used to explain CNN predictions can be broadly divided into two categories, gradient-based and perturbation based approaches. Gradient-based approaches generate a sensitivity heat map by computing the partial derivatives of model output with respect to every input pixel via backpropagation. In perturbation based approaches the output of the model is observed by modifying regions on the input image and thereby identifying the sensitive regions. Despite being time-consuming, in most real world use cases such as in medical imaging, practitioners tend to use occlusion experiments [14], a perturbation approach, as the preferred approach for explanations as they produce high quality fine grained sensitivity heat maps [13] using a process which is very intuitive to the human observer [17].

3 PRELIMINARIES AND OVERVIEW

In this section, we first formally state the problem and explain our assumptions. Then we formalize the internals of critical layers in a Deep CNN for the purpose of proposing our *incremental inference* approach in Section 4.

3.1 Problem Statement and Assumptions

We are given a CNN f, an image ${}^{img}\mathcal{I}$ on which the occlusion experiment needs to be run, the predicted class label L for ${}^{img}\mathcal{I}$, an occluding patch $\mathcal{P}$ in RGB format, and occluding patch striding amount ${}^{\mathcal{P}}S$. In the interactive case Krypton expects the user to also provide a set of interested occluding patch positions G. In the non-interactive scenario Krypton uses the user provided ${}^{\mathcal{P}}S$ value to initialize G to the all possible occluding positions. The occlusion experiment workload is to generate a 2-D heat map M with values corresponding to the coordinates in G contain the predicted probability for L by f for the occluded image ${}^{img}\mathcal{I}'_{x,y}$ and zero otherwise. More precisely, we can state the workload using the following set of logical statements:

Table 1: Symbols used in the Preliminaries Section

Symbol	Meaning
${}^{l}\mathcal{I}({}^{img}\mathcal{I})$	Input activation volume of the l^{th} layer (Input Image)
${}^{l}\mathcal{O}$	Output activation volume of the l^{th} layer
${}^{l}C_I, {}^{l}H_I, {}^{l}W_I$	Depth, height, and width of l^th layer Input
${}^{l}C_O, {}^{l}H_O, {}^{l}W_O$	Depth, height, and width of l^{th} layer Output
${}^{l}\mathcal{K}_{conv}$	Convolution filter kernels for the l^{th} layer
${}^{l}\mathcal{B}_{conv}$	Convolution bias value vector for the l^{th} layer
${}^{l}\mathcal{K}_{pool}$	Pooling filter kernel for the l^th layer
${}^{l}H_{\mathcal{K}}, {}^{l}W_{\mathcal{K}}$	Height and width of the filter kernel for the l^{th} layer
${}^{l}S{\equiv}({}^{l}S_x, {}^{l}S_y)$	Filter kernel patch striding amount for the l^{th} layer (${}^{l}S_x$ and ${}^{l}S_y$ corresponds to width and height dimensions)
${}^{l}P{\equiv}({}^{l}P_x, {}^{l}P_y)$	Padding amount for the l^{th} layer (${}^{l}P_x$ and ${}^{l}P_y$ corresponds to padding along width and height dimensions)
$\mathcal{P}$	Occluding patch in RGB format
${}^{\mathcal{P}}S$	Occluding patch striding amount
M	Heat map produced by the occlusion experiment
H_M, W_M	Height and width of M
f	Fine-tuned CNN which takes in an input image and outputs a probability distribution over the class labels
L	Class label predicted by f for the original image ${}^{img}\mathcal{I}$
G	Set of occluding patch superimposition positions on ${}^{img}\mathcal{I}$ in (x,y) format
$\circ_{x,y}$	Superimposition operator. $A \circ_{x,y} B$, superimposes B on top of A starting off at (x, y) position

$$W_M = \lfloor(\texttt{width}(^{img}\mathcal{I}) - \texttt{width}(\mathcal{P}) + 1)/^{\mathcal{P}}S\rfloor \quad (1)$$

$$H_M = \lfloor(\texttt{height}(^{img}\mathcal{I}) - \texttt{height}(\mathcal{P}) + 1)/^{\mathcal{P}}S\rfloor \quad (2)$$

$$M \in \mathbb{R}^{H_M \times W_M} \quad (3)$$

$$\forall\, x, y \in G: \quad (4)$$

$$^{img}\mathcal{I}'_{x,y} \leftarrow {}^{img}\mathcal{I} \circ_{x,y} \mathcal{P} \quad (5)$$

$$M[x, y] \leftarrow f(^{img}\mathcal{I}'_{x,y})[L] \quad (6)$$

Step (1), and (2) calculates the dimensions of the generated heat map M which is dependent on the dimensions of $^{img}\mathcal{I}$, $\mathcal{P}$, and $^{\mathcal{P}}S$. Step (5) superimposes $\mathcal{P}$ on $^{img}\mathcal{I}$ with its top left corner placed on the (x,y) location of $^{img}\mathcal{I}$. Step (6) calculates the output value at the (x,y) location by performing CNN inference for $^{img}\mathcal{I}'_{x,y}$ using f and taking the predicted probability for the label L. Step 5 and 6 are run for all occluding patch position values in G. In the non-interactive case G is initialized to $G = [0, H_M) \times [0, W_M)$. In the interactive case it is possible that human operator would provide multiple Gs, one after the other, for which the system has to evaluate iteratively.

We assume that f is a CNN from a roster of well-known CNNs (currently, VGG 16 layer version, ResNet 18 layer version, and Inception version 3). This is a reasonable start since most recent CNN based image recognition applications use only such well-known CNNs from model zoos [18, 19]. Nevertheless, our work is orthogonal to the specifics of a particular architecture and the proposed approaches can be easily extended to any architecture. We leave support for arbitrary CNNs to future work.

3.2 Deep CNN Internals

Input and output of individual layers in a Deep CNN except for Fully-Connected ones are arranged into three-dimensional volumes which have a width, height, and depth. For example an RGB input image of 224×224 spatial sizes can be considered as an input volume having a width and height of 224 and a depth of 3 (corresponding to 3 color channels). Every non Fully-Connected layer will take in an input volume and transform it into another volume. A Fully-Connected layer takes in a vector or flattened activation volume as input and transforms it into another vector. For our purpose, these transformations can be broadly divided into three categories based on how they operate spatially:

- Transformations that operate at the granularity of a global context.
  - E.g. Fully-Connected
- Transformations that operate at the granularity of individual spatial locations.
  - E.g. ReLU, Batch Normalization
- Transformations that operate at the granularity of a local spatial context.
  - E.g. Convolution, Pooling

Transformations that operate at the granularity of a global context. These transformations operate on a global context or in other words, does not take into account the spatial information. Fully-Connected layer, which is the only global context transformation in a CNN, takes in an input vector and performs a vector-matrix multiplication with a weight matrix and produces an output. As they perform one bulk transformation there is no opportunity for exploiting redundancies. The computational cost of a Full-Connected transformation is proportional to the product of the size of the input and output vectors. Fully-Connected layers are used as the last or last few layers in a CNN and only accounts for a small fraction of the total computational cost.

Transformations that operate at the granularity of individual spatial locations. These transformations essentially perform a *map*(.) function on each individual activation value (see Figure 2 (b)). Hence the output will have the same dimensions as input. The computational cost incurred by these transformations is proportional to the volume of the input (or output). However, with incremental spatially localized updates in the input, such as placing an occlusion patch, only the updated region is needed to be recalculated. Extending these transformations to become change aware is straightforward. The computational cost of the change aware incremental transformation is proportional to the volume of the modified region.

Transformations that operate at the granularity of a local spatial context. With incremental spatially localized updates in the input, transformations that operate at the granularity of a local spatial context also provide opportunities for exploiting redundancy and can be made change aware. However, with local context transformations, such as Convolution and Pooling, this extension is non-trivial due to the overlapping nature of the spatial contexts. Each Convolution layer can have ${}^{l}C_O$ 3-D filter kernels organized into a 4-D array ${}^{l}\mathcal{K}_{conv}$ with each having a smaller spatial width ${}^{l}W_{\mathcal{K}}$ and height ${}^{l}H_{\mathcal{K}}$ compared to the width ${}^{l}W_I$ and height ${}^{l}H_I$ of the input volume ${}^{l}\mathcal{I}$, but has the same depth ${}^{l}C_I$. During inference, c^{th} filter kernel is strided along the width and height dimensions of the input and a 2-D activation map ${}^{c}A = ({}^{c}a_{y,x}) \in \mathbb{R}^{{}^{l}H_O \times {}^{l}W_O}$ is produced by taking elementwise product between the kernel and the input and adding a bias value as per Equation 7. The computational cost of each of these individual element-wise product is proportional to the volume of the filter kernel. Finally, these 2-D activation maps are stacked together along the depth dimension to produce an output volume ${}^{l}O \in \mathbb{R}^{{}^{l}C_O \times {}^{l}H_O \times {}^{l}W_O}$ as

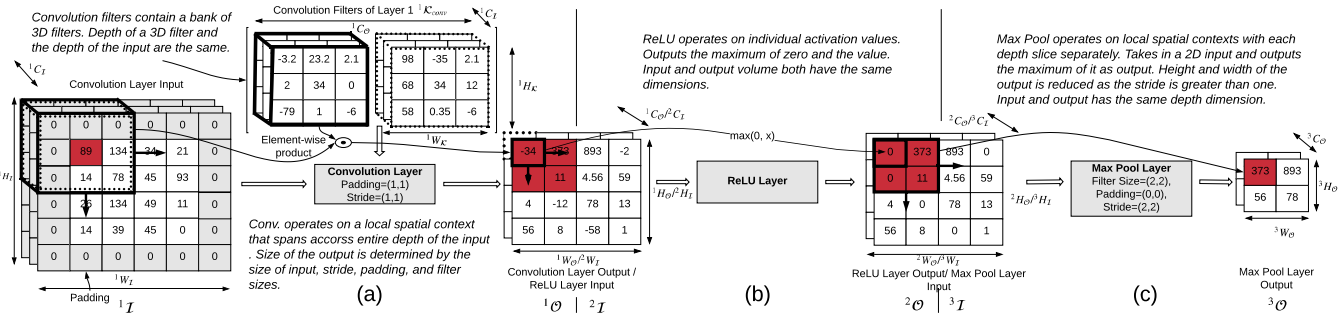


Figure 2: Simplified representation of selected layers of a Deep CNN. For simplicity addition of bias is not shown in the Convolution transformation. The values marked in red show how a small spatial update in the first input would propagate through subsequent transformation. Notation is explained in Table 1.

per Equation 8. A simplified representation of Convolution transformation is shown in Figure 2 (a).

$$
\begin{aligned}
{}^{c}a_{y,x} = \sum_{k=0}^{{}^{l}C_I} \sum_{j=0}^{{}^{l}H_{\mathcal{K}}-1} \sum_{i=0}^{{}^{l}W_{\mathcal{K}}-1} {}^{l}\mathcal{K}_{conv}[c,k,j,i] \\
\times {}^{l}\mathcal{I}[k, y - \lfloor \frac{{}^{l}H_{\mathcal{K}}}{2} \rfloor + j, x - \lfloor \frac{{}^{l}W_{\mathcal{K}}}{2} \rfloor + i] \\
+ {}^{l}\mathcal{B}_{conv}[c]
\end{aligned} \tag{7}
$$

$$
{}^{l}\mathcal{O} = [{}^{0}A, {}^{1}A, ..., {}^{{}^{l}C_O-1}A] \tag{8}
$$

Pooling can also be thought as a Convolution operation with a fixed (i.e. not learned) 2-D filter kernel ${}^{l}\mathcal{K}_{pool}$. But unlike Convolution, Pooling operates independently on each depth slice of the input volume. A Pooling layer takes a 3-D activation volume ${}^{l}\mathcal{O}$ having a depth of ${}^{l}C$, width of ${}^{l}W_I$, and height of ${}^{l}H_I$ as input and produces another 3-D activation volume ${}^{l}\mathcal{O}$ which has the same depth of ${}^{l}C$, width of ${}^{l}W_O$, and height of ${}^{l}H_O$ as the output. Pooling kernel is generally strided with more than one pixel at a time and hence ${}^{l}W_O$ and ${}^{l}H_O$ are generally smaller than ${}^{l}W_I$ and ${}^{l}H_I$. A simplified representation of Pooling transformation is shown in Figure 2 (c).

Relationship between Input and Output Spatial Sizes. The output volume's spatial sizes ${}^{l}W_O$ and ${}^{l}H_O$ are determined by the spatial sizes of the input volume ${}^{l}W_I$ and ${}^{l}H_I$, spatial sizes of the filter kernel ${}^{l}W_{\mathcal{K}}$ and ${}^{l}H_{\mathcal{K}}$ and two other parameters: **stride** ${}^{l}S$ and **padding** ${}^{l}P$. Stride is the amount of pixel values used to slide the filter kernel at a time when producing a 2-D activation map. It is possible to have two different values with one for the width dimension ${}^{l}S_x$ and one for the height dimension ${}^{l}S_y$. Generally ${}^{l}S_x \leq {}^{l}W_{\mathcal{K}}$ and ${}^{l}S_y \leq {}^{l}H_{\mathcal{K}}$. In Figure 2 Convolution transformation uses a stride value of 1 and Pool transformation uses a stride value of 2 for both dimensions. Sometimes in order to control the spatial size of the output activation map to be same as the input activation map, one needs to pad the input feature map with zeros around the spatial border. Padding ${}^{l}P$ captures the amount of zeros that needs to be added. Similar to the stride ${}^{l}S$, it is possible to have two separate values for padding with one for the width dimension ${}^{l}P_x$ and one for the height dimension ${}^{l}P_y$. In Figure 2 Convolution transformation pads the input with one line of zeros from both dimensions. With these parameters defined, the width (similarly height) of the output activation volume can be defined as follows:

$$
{}^{l}W_O = ({}^{l}W_I - {}^{l}W_{\mathcal{K}} + 1 + 2 \times {}^{l}P_x)/{}^{l}S_x \tag{9}
$$

Estimating the Computational Cost of Deep CNNs. Deep CNNs are highly compute intensive and out of the different types of layers Convolution layers contribute to 90% (or more) of the computation. Hence we can estimate the computational cost of a Deep CNN by counting the number of fused multiply-add (FMA) floating point operations (FLOPs) required by Convolution layers for a single forward inference.

For example, applying a Convolution filter having the dimensions of $({}^{l}C_I, {}^{l}H_{\mathcal{K}}, {}^{l}W_{\mathcal{K}})$ to a single spatial context will require ${}^{l}C_I \times {}^{l}H_{\mathcal{K}} \times {}^{l}W_{\mathcal{K}}$ many FLOPs, each corresponding to a single element-wise multiplication. Thus, the total amount of computations ${}^{l}Q$ required by that layer in order to produce an output $\mathcal{O}$ having dimensions ${}^{l}C_O \times {}^{l}H_O \times {}^{l}W_O$, and the total amount of computations Q required to process the entire set of convolution layers L in the CNN can be calculated as per Equation 10 and 11. However, in the case of incremental updates only a smaller spatial patch having a width ${}^{l}W_{\mathcal{P}}$ (${}^{l}W_{\mathcal{P}} <= {}^{l}W_O$) and height ${}^{l}H_{\mathcal{P}}$ (${}^{l}H_{\mathcal{P}} <= {}^{l}H_O$) is needed to be recomputed. The amount of computations required for the incremental computation ${}^{l}Q_{inc}$ and total amount of incremental computations Q_{inc} required for the

entire set of convolution layers L will be smaller than the above full computation values and can be calculated as per Equation 12 and 13. Notice that in both of the above formulations computational cost calculation takes into account the spatial sizes of the output or the output patch. The output spatial sizes can be determined using the spatial sizes of the input and other parameters as shown in the previous Section. In Section 4 we show that the spatial sizes of the output patch can also be predetermined using the input patch spatial sizes.

Based on the above quantities we define a new metric named *theoretical speedup* R, which is the ratio between the total full computational cost Q and total incremental computation cost Q_{inc} (see Equation 14). This ratio essentially acts as a surrogate for the theoretical upper-bound for computational and runtime savings that can be achieved by applying incremental computations to deep CNNs.

$$^{l}Q = (^{l}C_{I} \times {}^{l}H_{\mathcal{K}} \times {}^{l}W_{\mathcal{K}}) \times (^{l}C_{O} \times {}^{l}H_{O} \times {}^{l}W_{O}) \quad (10)$$

$$Q = \sum_{l \in L} {}^{l}Q \quad (11)$$

$$^{l}Q_{inc} = (^{l}C_{I} \times {}^{l}H_{\mathcal{K}} \times {}^{l}W_{\mathcal{K}}) \times (^{l}C_{O} \times {}^{l}H_{\mathcal{P}} \times {}^{l}W_{\mathcal{P}}) \quad (12)$$

$$Q_{inc} = \sum_{l \in L}^{l} Q_{inc} \quad (13)$$

$$R = \frac{Q}{Q_{inc}} \quad (14)$$

Table 2: Additional symbols used in the Optimizations Section

Symbol	Meaning
$^{l}x^{I}_{\mathcal{P}}, {}^{l}y^{I}_{\mathcal{P}}$	Starting coordinates of the input patch for the l^{th} layer
$^{l}x^{\mathcal{R}}_{\mathcal{P}}, {}^{l}y^{\mathcal{R}}_{\mathcal{P}}$	Starting coordinates of the patch that needs to be read in for the l^{th} layer transformation
$^{l}x^{O}_{\mathcal{P}}, {}^{l}y^{O}_{\mathcal{P}}$	Starting coordinates of the output patch for the l^{th} layer
$^{l}H^{I}_{\mathcal{P}}, {}^{l}W^{I}_{\mathcal{P}}$	Height and width of the input patch for the l^{th} layer
$^{l}H^{\mathcal{R}}_{\mathcal{P}}, {}^{l}W^{\mathcal{R}}_{\mathcal{P}}$	Height and width of the patch that needs to be read in for the l^{th} layer transformation
$^{l}H^{O}_{\mathcal{P}}, {}^{l}W^{O}_{\mathcal{P}}$	Height and width of the output patch for the l^{th} layer
τ	Projective field threshold
$r_{drill-down}$	Stage two drill-down fraction used in *adaptive drill-down*

4 OPTIMIZATIONS

In this section we explain *exact inference* and *approximate inference* optimization in detail. In Krypton these optimizations are applied on top of the currently dominant approach of performing CNN inference on batches of images, with batch size selected to optimize the hardware utilization, where each image corresponds to an occluded instance of the original image. The batched inference is important as it reduces per image inference time by amortizing the fixed overheads. In our experiments we found that this simple optimization alone can give up to 1.4X speedups on CPU environments and 2X speedups on the GPU environment compared to the per image inference approach. Finally we explain how Krypton configures its internal system configurations for *approximate inference*.

4.1 Exact: Incremental Inference

As explained earlier occlusion experiments in its naive form perform many redundant computations. In order to avoid these redundancies layers in a CNN have to be change aware and operate in an incremental manner i.e. reuse previous computations as much as possible and compute only the required ones. In this section, we focus on transformations that operate at the granularity of a local spatial context (i.e. Convolution and Pooling) as other types either have no redundancies (global context transformations) or are trivial to make incremental aware (point transformations).

Incremental Convolution and Pooling.

In Section 3 we explained that both Convolution and Polling transformations can be cast into a form of applying a filter along the spatial dimensions of the input volume. However, how each transformation operates along the depth dimension is different. For our purpose, we are only interested in finding the spatial propagation of the patches in the input through the consecutive layers and hence both these transformations can be treated similarly. The coordinates and the dimensions (i.e. height and width) of the modified patch in the output volume caused by a modified patch in the input volume are determined by the coordinates and the dimensions of the input patch, sizes of the filter kernel $^{l}H_{\mathcal{K}}$ and $^{l}W_{\mathcal{K}}$, padding values $^{l}P_{x}$ and $^{l}P_{y}$, and the strides $^{l}S_{x}$ and $^{l}S_{y}$. For example consider the simplified demonstration showing a cross-section of input and output in Figure 3. We use a coordinate system whose origin is placed at the top left corner of the input. A patch marked in red is placed on the input starting off at $^{l}x^{I}_{\mathcal{P}}, {}^{l}y^{I}_{\mathcal{P}}$ coordinates and has a height of $^{l}H^{I}_{\mathcal{P}}$ and width of $^{l}W^{I}_{\mathcal{P}}$. The updated patch in the output starts off at $^{l}x^{O}_{\mathcal{P}}, {}^{l}y^{O}_{\mathcal{P}}$ and has a height of $^{l}H^{O}_{\mathcal{P}}$ and width of $^{l}W^{O}_{\mathcal{P}}$. Note that due to the overlapping nature of filter positions, to compute the output patch transformations have to read a slightly larger context than the updated patch. This read in context is shown by the blue shaded area in Figure 3. The starting coordinates of this

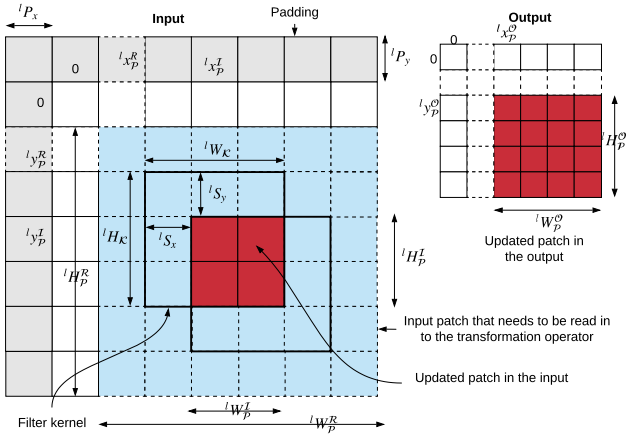


Figure 3: Simplified representation of input and output patch coordinates and dimensions of Convolution and Pool transformations.

read-in patch are denoted by ${}^{l}x^{\mathcal{R}}_{\mathcal{P}}, {}^{l}y^{\mathcal{R}}_{\mathcal{P}}$ and the dimensions are denoted by ${}^{l}W^{\mathcal{R}}_{\mathcal{P}}, {}^{l}H^{\mathcal{R}}_{\mathcal{P}}$. The relationship between the coordinates and dimensions in the horizontal axis (similarly in the vertical axis) can be expressed as follows:

$$ {}^{l}x^{\mathcal{O}}_{\mathcal{P}} = max\Big(\lceil({}^{l}P_x + {}^{l}x^{\mathcal{I}}_{\mathcal{P}} - {}^{l}W_{\mathcal{K}} + 1)/{}^{l}S_x\rceil, 0\Big) \tag{15} $$

$$ {}^{l}W^{\mathcal{O}}_{\mathcal{P}} = min\Big(\lceil({}^{l}W^{\mathcal{I}}_{\mathcal{P}} + {}^{l}W_{\mathcal{K}} - 1)/{}^{l}S_x\rceil, {}^{l}W_O\Big) \tag{16} $$

$$ {}^{l}x^{\mathcal{R}}_{\mathcal{P}} = {}^{l}x^{\mathcal{O}}_{\mathcal{P}} \times {}^{l}S_x - {}^{l}P_x \tag{17} $$

$$ {}^{l}W^{\mathcal{R}}_{\mathcal{P}} = {}^{l}W_{\mathcal{K}} + ({}^{l}W^{\mathcal{O}}_{\mathcal{P}} - 1) \times {}^{l}S_x \tag{18} $$

Equation 15 calculates the starting coordinates of the output patch. Use of padding effectively shifts the coordinate system and therefore ${}^{l}P_x$ is added to correct it. Due to the overlapping nature of filter kernels the affected span of the updated patch in the input will be increased by ${}^{l}W_{\mathcal{K}} - 1$ amount and hence needs to be subtracted from the input coordinates ${}^{l}x^{\mathcal{I}}_{\mathcal{P}}$ (a filter of size ${}^{l}W_{\mathcal{K}}$ which is placed starting at ${}^{l}x^{\mathcal{I}}_{\mathcal{P}} - {}^{l}W_{\mathcal{K}} + 1$ will see the new change at ${}^{l}x^{\mathcal{I}}_{\mathcal{P}}$). Equation 16 calculates the width of the output patches. Once the output patch coordinate and width are calculated it is straightforward to calculate the read-in patch coordinate as per Equation 17 and the width as per Equation 18.

With all the geometric mappings defined we now explain the complete incremental inference approach for a single layer. Algorithm 1 presents it formally. The IncrementalInference procedure takes in the original transformation ${}^{l}T$ of the l^{th} layer, pre-materialized input for the layer corresponding to the original image, a batch of updated patches which are 3D volumes of activation values and their geometric properties as input. Notice that for the first layer all

Algorithm 1 Incremental Inference Transformation

Input:
${}^{l}T$: *Original Transformation*
${}^{l}\mathcal{I}$: *Pre-materialized input from original image*
$[{}^{l}\mathcal{P}^{\mathcal{I}}{}_1, ..., {}^{l}\mathcal{P}^{\mathcal{I}}{}_n]$: *Input patches*
$[({}^{l}x^{\mathcal{I}}_{\mathcal{P}_1}, {}^{l}y^{\mathcal{I}}_{\mathcal{P}_1}), ..., ({}^{l}x^{\mathcal{I}}_{\mathcal{P}_n}, {}^{l}y^{\mathcal{I}}_{\mathcal{P}_n})]$: *Input patch coordinates*
${}^{l}W^{\mathcal{I}}_{\mathcal{P}}, {}^{l}H^{\mathcal{I}}_{\mathcal{P}}$: *Input patch dimensions*

Output:
$[{}^{l}\mathcal{P}^{\mathcal{O}}{}_1, ..., {}^{l}\mathcal{P}^{\mathcal{O}}{}_n]$: *Output patches*
$[({}^{l}x^{\mathcal{O}}_{\mathcal{P}_1}, {}^{l}y^{\mathcal{O}}_{\mathcal{P}_1}), ..., ({}^{l}x^{\mathcal{O}}_{\mathcal{P}_n}, {}^{l}y^{\mathcal{O}}_{\mathcal{P}_n})]$: *Output patch coordinates*
${}^{l}W^{\mathcal{O}}_{\mathcal{P}}, {}^{l}H^{\mathcal{O}}_{\mathcal{P}}$: *Output patch dimensions*

1: **procedure** IncrementalInference
2: *Calculate* $[({}^{l}x^{\mathcal{O}}_{\mathcal{P}_1}, {}^{l}y^{\mathcal{O}}_{\mathcal{P}_1}), ..., ({}^{l}x^{\mathcal{O}}_{\mathcal{P}_n}, {}^{l}y^{\mathcal{O}}_{\mathcal{P}_n})]$
3: *Calculate* $({}^{l}W^{\mathcal{O}}_{\mathcal{P}}, {}^{l}H^{\mathcal{O}}_{\mathcal{P}})$
4: *Calculate* $[({}^{l}x^{\mathcal{R}}_{\mathcal{P}_1}, {}^{l}y^{\mathcal{R}}_{\mathcal{P}_1}), ..., ({}^{l}x^{\mathcal{R}}_{\mathcal{P}_n}, {}^{l}y^{\mathcal{R}}_{\mathcal{P}_n})]$
5: *Calculate* $({}^{l}W^{\mathcal{R}}_{\mathcal{P}}, {}^{l}H^{\mathcal{R}}_{\mathcal{P}})$
6: *Initialize* $\mathcal{R} \in \mathbb{R}^{\mathtt{depth}({}^{l}\mathcal{I}) \times {}^{l}H^{\mathcal{R}}_{\mathcal{P}} \times {}^{l}W^{\mathcal{R}}_{\mathcal{P}}}$
7: **for** `i in [1,...,n]` **do**
8: $T_1 \leftarrow {}^{l}\mathcal{I}[:, {}^{l}x^{\mathcal{R}}_{\mathcal{P}_i} : {}^{l}x^{\mathcal{R}}_{\mathcal{P}_i} + {}^{l}W^{\mathcal{R}}_{\mathcal{P}}, {}^{l}y^{\mathcal{R}}_{\mathcal{P}_i} : {}^{l}y^{\mathcal{R}}_{\mathcal{P}_i} + {}^{l}H^{\mathcal{R}}_{\mathcal{P}}]$
9: $T_2 \leftarrow T_1 \circ_{({}^{l}x^{\mathcal{I}}_{\mathcal{P}_i} - {}^{l}x^{\mathcal{R}}_{\mathcal{P}_i}), ({}^{l}y^{\mathcal{I}}_{\mathcal{P}_i} - {}^{l}y^{l}\mathcal{R}_{\mathcal{P}_i})} {}^{l}\mathcal{P}_i$
10: $R[i, :, :] \leftarrow T_2$
11: $[{}^{l}\mathcal{P}^{\mathcal{O}}_1, ..., {}^{l}\mathcal{P}^{\mathcal{O}}_n] \leftarrow T(\mathcal{R})$
12: **return** $[{}^{l}\mathcal{P}^{\mathcal{O}}_1, ..., {}^{l}\mathcal{P}^{\mathcal{O}}_n]$,
13: $[({}^{l}x^{\mathcal{O}}_{\mathcal{P}_1}, {}^{l}y^{\mathcal{O}}_{\mathcal{P}_1}), ..., ({}^{l}x^{\mathcal{O}}_{\mathcal{P}_n}, {}^{l}y^{\mathcal{O}}_{\mathcal{P}_n})], ({}^{l}W^{\mathcal{O}}_{\mathcal{P}}, {}^{l}H^{\mathcal{O}}_{\mathcal{P}})$

the elements in the batch of updated patches will be identical as the same occlusion patch (in RGB format) is used. However, at latter layers they will be different as the output of the first layer will be different due to different read-in contexts. First IncrementalInference calculates the geometric properties of the output and the read-in patches. A temporary input volume R is initialized to hold the input patches with their read-in contexts. The for loop iteratively populates R with corresponding patches. Finally, ${}^{l}T$ is applied to R to compute the output patches. In a CNN which has multiple such layers chained together, the outputs of the IncrementalInference procedure are fed as input again for the incremental inference of the next layer along with the pre-materialized input from the original image. However, at a boundary of local context transformation and a global context transformation, such as in Convolution → Fully-Connected or Pool → Fully-Connected, full updated output has to be created instead of propagating only the updated patches. The high-level steps taken by the end-to-end incremental inference approach for occlusion experiments can be summarized as follows:

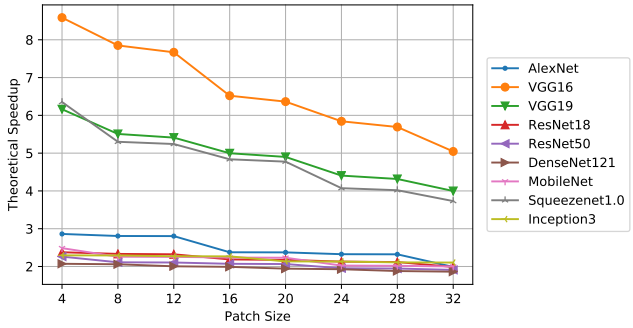


Figure 4: Theoretical speedup for popular CNN architectures with *incremental inference*.

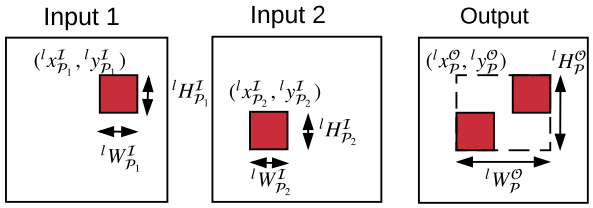


Figure 5: Input-output coordinate and dimension mapping for element-wise addition and depth-wise concatenation.

(1) Take in CNN f, image ${}^{img}\mathcal{I}$, predicted class label L, occlusion patch $\mathcal{P}$, stride ${}^{\mathcal{P}}S$ for the $\mathcal{P}$, and the set of occluding patch placement positions G as input.
(2) Pre-materialize output of all the transformations in f by feeding in ${}^{img}\mathcal{I}$.
(3) Prepare the occluded images (${}^{img}\mathcal{I}'_{x,y}$ s) corresponding to all positions in G.
(4) For batches of ${}^{img}\mathcal{I}'_{x,y}$ as the input invoke the transformations in f in a chained manner and calculate the corresponding values of heat map M.
   - For local context transformation invoke IncrementalInference.
   - For local context transformation that feeds in a global context transformation additionally materialize the full updated output.
   - For all others invoke the original transformation.
(5) Return M as the final output.

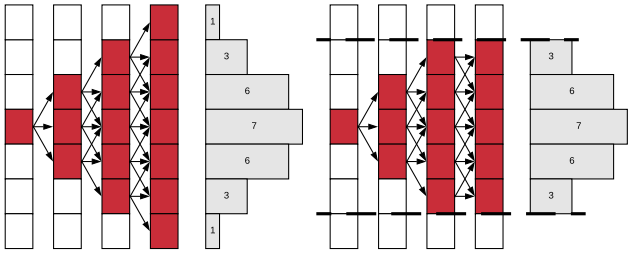
Theoretical Speedup with Incremental Inference. We analyze the theoretical speedup that can be achieved with the *incremental inference* approach when a square occlusion patch is placed on the center[1] of the input image. Figure 4 shows the results. VGG16 model results in the maximum theoretical speedup and DenseNet121 model has the lowest theoretical speedup. Most CNN architectures result in a theoretical speedup between 2-3. The theoretical speedup for a CNN with *incremental inference* is determined by the characteristics of its architecture such as the number of layers, the sizes of the filter kernels, and the filter stride values. For example, VGG16 which uses small Convolution filter kernels and strides incurs a very high computational cost (15 GFLOPs) for a single full inference. However, as the change propagation rate with small filter sizes and strides is small, significant computational savings and runtime speedups can be achieved with incremental inference.

TODO: MQO + IVM

CPU versus GPU implementation concerns. Through our experiments we found that even though a straightforward implementation of *incremental inference* approach as shown in Algorithm 1 produces expected speedups for the CPU environment it performs poorly on the GPU environment. The for loop on the line 7 of Algorithm 1 is essentially preparing the input for T by copying values from one part of the memory to another sequentially. This sequential operation becomes a bottleneck for the GPU implementation as it cannot exploit the available parallelism of the GPU efficiently. To overcome this we have extended PyTorch by adding a custom kernel written in CUDA language which performs the input preparation more efficiently by parallelly copying the memory regions for all items in the batch and then invoke the CNN transformation. TODO: Refer to hardware works, e.g. Li Tseng, Lin, Swanson, Yannis on hardware accelerators

Element-wise addition and depth-wise concatenation. Element-wise addition and depth-wise concatenation are two widely used linear algebra operators in CNN. Element-wise addition operator requires the two input volumes to have exact same dimensions and the depth-wise concatenation requires them to have same height and width dimensions. Consider a situation for these operators as shown in Figure 5 where the first input has incremental spatial update starting at ${}^{l}x^{\mathcal{I}}_{\mathcal{P}_1}, {}^{l}y^{\mathcal{I}}_{\mathcal{P}_1}$ coordinates with dimensions of ${}^{l}H^{\mathcal{I}}_{\mathcal{P}_1}$ and ${}^{l}W^{\mathcal{I}}_{\mathcal{P}_1}$ and for the second input starting at ${}^{l}x^{\mathcal{I}}_{\mathcal{P}_2}, {}^{l}y^{\mathcal{I}}_{\mathcal{P}_2}$ coordinates with dimensions of ${}^{l}H^{\mathcal{I}}_{\mathcal{P}_2}$ and ${}^{l}W^{\mathcal{I}}_{\mathcal{P}_2}$. In this case the coordinates and dimensions of both output and read-in patches will be the same and computing them is essentially finding the bounding box of the two input patches. For the horizontal axis (similarly to vertical axis) it can be expressed as follows:

[1]If the occlusion patch is placed towards a corner of the input image the theoretical speedup will be slightly higher. But placing the occlusion patch on the center gives us a worst-case estimate.



(a) Projective Field (b) Projective field thresholding

Figure 6: (a) One dimensional Convolution demonstrating projective field growth (filter size = 2, stride = 1). (b) *Projective field thresholding* with $\tau = 5/7$.

$$
\begin{aligned}
{}^{l}x_{P}^{O} &= \texttt{min}({}^{l}x_{\mathcal{P}_1}^{I}, {}^{l}x_{\mathcal{P}_2}^{I}) \\
{}^{l}W_{\mathcal{P}}^{O} &= \texttt{max}({}^{l}x_{\mathcal{P}_1}^{I} + {}^{l}W_{\mathcal{P}_1}^{I}, {}^{l}x_{\mathcal{P}_2}^{I} + {}^{l}W_{\mathcal{P}_2}^{I}) - \texttt{min}({}^{l}x_{\mathcal{P}_1}^{I}, {}^{l}x_{\mathcal{P}_2}^{I})
\end{aligned} \tag{19}
$$

4.2 Approximate: Projective Field Thresholding

Projective field [20, 21] of a CNN neuron is the local region (including the depth) of the output volume which is connected to it. The term is borrowed from the Neuroscience field where it is used to describe the spatiotemporal effects exerted by a retinal cell on all of the outputs of the neuronal circuitry [22]. For our work, the notion of projective field is useful as it determines the change propagation path for incremental changes. The three types of CNN transformations affect the size of the projective field differently. Point transformations do not change the projective field size while global context transformations increase it to the maximum. Transformations that operate on a local spatial context increase it gradually. The amount of increase in a local context transformation is determined by the filter size and stride parameters. At every transformation, the size of the projective field will increase linearly by the filter size and multiplicatively by the stride value.

Because of the projective field growth, even though there will be many computational redundancies in the early layers towards the latter layers it will decrease or even have no redundancies. However, we empirically found that the projective field growth can be restrained up to a certain extent without significantly sacrificing the accuracy. For a more intuitive understanding of why this would work consider the simplified 1-D Convolution example shown in Figure 6 (a). In the example, a single neuron is modified (marked in red) and a filter of size three is applied with a stride of one repeatedly four times. Since the filter size is three, each updated neuron will propagate the change to three neurons in the next output layer causing the projective field to grow linearly. The histogram at the end of the fourth layer shows the number of unique paths that are available between each output neuron and the originally updated neuron in the first layer. It can be seen that this distribution resembles a Gaussian where many of the paths are connected to the central region. The amount of change in the output layer is determined by both the number of unique paths and also the individual weights of the connections. It can be shown that the distribution of change in the output layer will converge to a Gaussian distribution provided certain conditions are met for the weight values of the filter kernel (more details in Appendix X).

As most of the change will be concentrated on the center we introduce the notion of a projective field threshold τ ($0 < \tau \leq 1$) which will be used to restrict the growth of the projective field. It determines the maximum size of the projective field as a fraction of the size of the output. Figure 6 (b) demonstrates the application of projective field thresholding with a τ value of 5/7. From the histogram generated for the projective field thresholding approach, we can expect that much of the final output change is maintained by this approach.

In Krypton, *projective field thresholding* is implemented on top of *incremental inference* approach by applying set of additional constraints on input-output coordinate mappings. For the horizontal dimension (similarly to vertical dimension) the new set of calculations can be expressed as follows:

$$
{}^{l}W_{\mathcal{P}}^{O} = \texttt{min}\Big(\lceil({}^{l}W_{\mathcal{P}}^{I} + {}^{l}W_{\mathcal{K}} - 1)/{}^{l}S_x\rceil, {}^{l}W_{\mathcal{P}}^{O}\Big) \tag{20}
$$

$$
\text{If } {}^{l}W_{\mathcal{P}}^{O} > \texttt{round}(\tau \times {}^{l}W^{O}) : \tag{21}
$$

$$
{}^{l}W_{\mathcal{P}}^{O} = \texttt{round}(\tau \times {}^{l}W^{O}) \tag{22}
$$

$$
{}^{l}W_{\mathcal{P}_{new}}^{I} = {}^{l}W_{\mathcal{P}}^{O} \times {}^{l}S_x - {}^{l}W_{\mathcal{K}} + 1 \tag{23}
$$

$$
{}^{l}x_{\mathcal{P}}^{I} \mathrel{+}= ({}^{l}W_{\mathcal{P}}^{I} - {}^{l}W_{\mathcal{P}_{new}}^{I})/2 \tag{24}
$$

$$
{}^{l}W_{\mathcal{P}}^{I} = {}^{l}W_{\mathcal{P}_{new}}^{I} \tag{25}
$$

$$
{}^{l}x_{\mathcal{P}}^{O} = \texttt{max}\Big(\lceil({}^{l}P_x + {}^{l}x_{\mathcal{P}}^{I} - {}^{l}W_{\mathcal{K}} + 1)/{}^{l}S_x\rceil, 0\Big) \tag{26}
$$

Equation 20 calculates output width assuming no thresholding. But if the output width exceeds the threshold defined by τ output width is set to the threshold value as per Equation 22. Equation 23 calculates the input width that would produce an output of width $W_{\mathcal{P}}^{O}$ (think of this as making $W_{\mathcal{P}}^{I}$ the subject of equation 20). If the new input width is smaller than the original input width, the starting x coordinate should be updated as per Equation 24 such that the updated coordinates correspond to a center crop from the

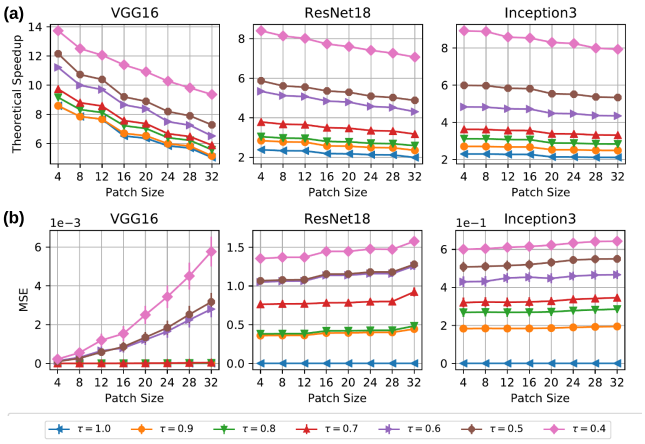


Figure 7: (a) Theoretical speedup ratio with projective field thresholding. (b) Mean Square Error between exact and approximate output of the final Conv. or Pool transformation.

original. Equation 25 set the input width to the newly calculated input width and Equation 26 calculates the x coordinate of the output patch from the updated values.

Theoretical Speedup with Projective Field Thresholding. We analyze the theoretical speedup that can be achieved with *projective field thresholding* approach when a square occlusion patch is placed on the center of the input image. Figure 7 (a) presents the results. It can be seen that with increasing τ attainable theoretical speedup also increases. We also analyze the mean square error (MSE) between the exact and approximate output of the activation volume produced by the final Convolution or Pool transformation with a black occlusion patch placed on the center of the input image. The results are shown in Figure 7 (b). With increasing τ and increasing patch size we see that the MSE is also increasing.

4.3 Approximate: Adaptive Drill-Down

Adaptive drill-down approach, which is only applicable in the non-interactive mode, is based on the observation that in many occlusion based explainability workloads, such as in medical imaging, the regions of interest will occupy only a small fraction of the entire image. In such cases, it is unnecessary to inspect the entire image at a higher resolution with a small stride value for the occlusion patch. In *adaptive-drill-down* the final occlusion heat map will be generated using a two-stage process. At the first stage, a low-resolution heat map will be generated by using a larger stride which we call stage one stride S_1. From the heat map generated at stage one, a predefined drill-down fraction $r_{drill-down}$ of regions with highest probability drop for the predicted class is identified. At stage two a high-resolution occlusion map

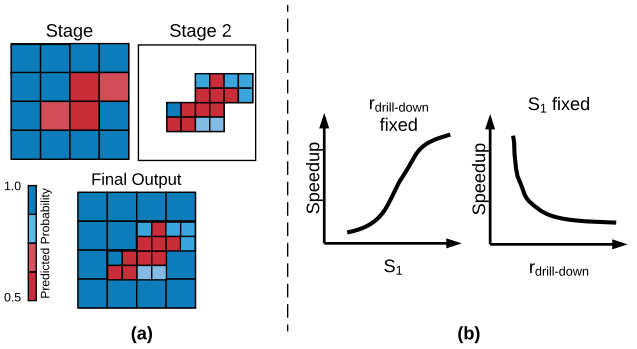


Figure 8: (a) Schematic representation of *adaptive drill-down*. (b) Conceptual diagram showing the effect of S_1 and $r_{drill-down}$ on speedup.

is generated using the original user provided stride value, also called stage two stride S_2, only for the selected region. A schematic representation of *adaptive drill-down* is shown in Figure 8 (a).

The amount of speedup that can be obtained from *adaptive drill-down* is determined by both $r_{drill-down}$ and S_1. If the $r_{drill-down}$ is low, only a small region will have to be examined at a higher resolution and thus it will be faster. However, this smaller region may not be sufficient to cover all the interesting regions on the image and hence can result in losing important information. Larger S_1 also reduces the overall runtime as it reduces the time taken for stage one. But it has the risk of misidentifying interesting regions especially when the granularity of those regions are smaller than the occlusion patch size. The speedup obtained by *adaptive drill-down* approach is equal to the ratio between the number of individual occlusion patch positions generated for the normal and *adaptive drill-down* approaches. Number of individual occlusion patch positions generated with a stride value of S is proportional to $1/S^2$ (total number of patch positions is equal to $\frac{H_{\mathcal{I}_{img}}}{S} \times \frac{W_{\mathcal{I}_{img}}}{S}$). Hence the speedup can be expressed as per Equation 27. Figure 8 (b) conceptually shows how the speedup would vary with S_1 when $r_{drill-down}$ is fixed and vice versa.

$$\text{speedup} = \frac{S_1^2}{S_2^2 + r_{drill-down} \times S_1^2} \tag{27}$$

4.4 System Tuning

In this section we explain how KRYPTON sets its internal configuration parameters for *approximate inference* optimizations.

Tuning projective field threshold. The inaccuracies incurred when applying *projective field thresholding* can cause

quality degradation in the generate approximate heat map all the way from indistinguishable changes major structural changes. To measure this quality degradation we use Structural Similarity (SSIM) Index [23] which is one of the widely used approaches to measuring the *human perceived difference* between two similar images. When applying SSIM index we treat the original heat map as the reference image with no distortions and the perceived image similarity of the approximate heat map is calculated with reference to it. The generated SSIM index is a value between -1 and 1, where 1 corresponds to perfect similarity. Typically SSIM index values in the range of $0.90 - 0.95$ are used in practical applications such as image compression and video encoding as at the human perception level they produce indistinguishable distortions. For more details on SSIM Index method, we refer the reader to the original SSIM Index paper [23].

Tuning *projective field threshold* τ is done during a special initial tuning phase. During this tuning phase Krypton takes in a sample of images (default 30) from the operational workload and evaluates SSIM value of the approximate heat map (compared to the exact heat map) for different τ values (default values are 1.0, 0.9, 0.8, ..., 0.4). These τ versus SSIM data points are then used to fit a second-degree curve. At the operational time, Krypton requires the user to provide the expected level of quality for the heat maps in terms of a SSIM value. τ is then selected from the curve fit to match this target SSIM value. Figure 9 (a) shows the SSIM variation and degree two curve fit for different τ values and three different CNN models for a tuning set (n=30) from OCT dataset. From the plots, it can be seen that the distribution of SSIM versus τ lies in a lower dimensional manifold and with increasing τ SSIM also increases. Figure 9 (b) shows the cumulative percentage plots for SSIM deviation for the tune and test sets (n=30) when the system is tuned for a target SSIM of 0.9. For a target SSIM of 0.9 system picks τ values of 0.5, 0.7, and 0.9 for VGG16, ResNet18, and Inception3 models respectively. It can be seen that approximately more than 50% of test cases will result in an SSIM value of 0.9 or greater. Even in cases where it performs worse than 0.9 SSIM, significant $(95\% - 100\%)$ portion of them are within +0.1 deviation.

Tuning adaptive drill-down. As explained in section 4.3 the speedup obtained by *adaptive drill-down* approach is determined by two factors; stage one stride value S_1 and drill-down fraction $r_{drill-down}$. For configuring *adaptive drill-down* Krypton requires the user to provide $r_{drill-down}$ and a target speedup value. $r_{drill-down}$ should be selected based on the user's experience and understanding on the relative size of interesting regions compared to the full image. This is a fair assumption and in most cases, such as in

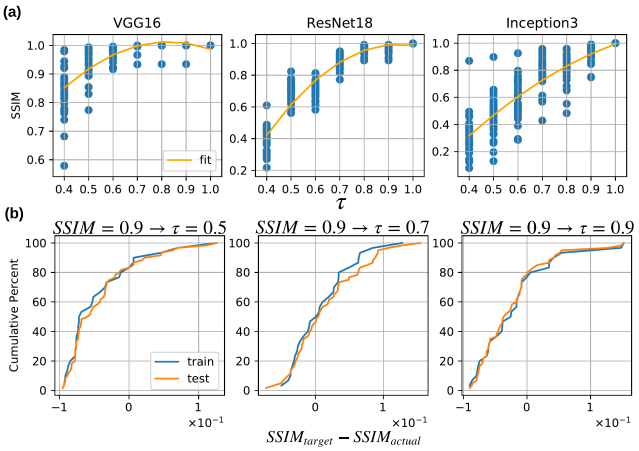


Figure 9: (a) SSIM variation and degree two curve fit for a sample of OCT dataset. (b) CDF plot for the SSIM deviation for the τ values picked from the curve fit for a target SSIM of 0.9.

medical imaging, users will have a fairly good understanding on the relative size of the interesting regions. However, if the user is unable to provide this value Krypton will use a default value of 0.25 as $r_{drill-down}$. The speedup value basically captures user's input on how much faster the occlusion experiment should run. Higher speedup values will sacrifice the quality of non-interesting (1-$r_{drill-down}$) regions for faster execution. The default value for speedup value is three. The way how Krypton configures *adaptive drill-down* is different to how it configures *projective field thresholding*. The reason for this is unlike in *projective field thresholding* in *adaptive drill-down* users have more intuition on the outcomes of $r_{drill-down}$ and target speedup parameters compared to the SSIM quality value of the final output. Given $r_{drill-down}$, target speedup value, and original occlusion patch stride value S_2 (also called stage two stride) Krypton then calculates the stage one stride value S_1 as per equation 28. As S_1 cannot be greater than the width W_{img} (similarly height H_{img}) of the image it can be seen that possible values for the speedup value are upper-bounded as per equation 29.

$$S_1 = \sqrt{\frac{\text{speedup}}{1 - r_{drill-down} \times \text{speedup}}} \times S_2 \tag{28}$$

$$S_1 = \sqrt{\frac{\text{speedup}}{1 - r_{drill-down} \times \text{speedup}}} \times S_2 < W_{img}$$

$$\text{speedup} < \min\left(\frac{W_{img}^2}{S_2^2 + r_{drill-down} \times W_{img}^2}, \frac{1}{r_{drill-down}}\right) \tag{29}$$

5 EXPERIMENTAL EVALUATION

We empirically validate if Krypton is able reduce the runtime taken for occlusion based deep CNN explainability workloads. We then conduct controlled experiments to show the individual contribution of each optimization in Krypton for the overall system efficiency.

Datasets. We use three real-world datasets: *OCT*, *Chest X-Ray*, and a sample from *ImageNet*. *OCT* has about 84,000 optical coherence tomography retinal images categorized into four categories: CNV, DME, DRUSEN, and NORMAL. CNV (choroidal neovascularization), DME (diabetic macular edema), and DRUSEN are three different varieties of Diabetic Retinopathy. NORMAL corresponds to healthy retinal images. *Chest X-Ray* has about 6,000 X-ray images categorized into three categories: VIRAL, BACTERIAL, and NORMAL. VIRAL and BACTERIAL categories correspond to two varieties of Pneumonia. NORMAL corresponds to chest X-Rays of healthy people. Both *OCT* and *Chest X-Ray* datasets are obtained from an original scientific study [6] which uses CNNs for predicting Diabetic Retinopathy and Pneumonia from radiological images. *ImageNet* sample dataset contains 1,000 images corresponding to two hundred categories selected from the original thousand categorical dataset [24].

Workloads. We use three popular ImageNet-trained deep CNNs: VGG16 [2], ResNet18 [3], and Inception3 [4], obtained from [25]. They complement each other in terms of model size, computational cost, amount of theoretical redundancy that exist for occlusion experiments, and the level of architectural complexity of the CNN model. For *OCT* and *Chest X-Ray* datasets the three CNN models are fine-tuned by retraining the final fully-connected layer with hyper-parameter tuning as per standard practice. More details on the fine-tuning process are included in the Appendix. Heat map for the predicted probabilities is generated using Python Matplotlib library's `imshow` method using the `jet_r` color scheme. For the heat map, maximum threshold value is set to $\texttt{min}(1, 1.25 \times p)$ and minimum threshold value is set to $0.75 \times p$ where p is predicted class probability for the unmodified image. Original images were resized to the size required by the CNNs (224×224 for VGG16 and ResNet18 and 299×299 for Inception3) and no additional pre-processing is done. For GPU experiments a batch size of 128 and for CPU experiments a batch size 16 is used. CPU experiments are executed with a thread parallelism of 8. All of our datasets, fine-tuning, experiment, and system code will be made available on our project web page.

Experimental Setup. We use a workstation which has 32 GB RAM, Intel i7-6700 @ 3.40GHz CPU, 1 TB Seagate ST1000DM010-2EP1, and Nvidia Titan X (Pascal) 12 GB memory GPU. The system runs Ubuntu 16.04 operating system with PyTorch version of 0.4.0, CUDA version of 9.0, and cuDNN version of 7.1.2. Each runtime reported is the average of three runs with 95% confidence intervals shown.

5.1 End-to-End Evaluation

For the GPU based environment we compare two variations Krypton, Krypton-Exact which only applies the *incremental inference* optimization and Krypton-Approximate which applies both *incremental inference* and *approximate inference* optimizations, against two baselines. *Naive* is the current dominant practice of performing full inference for multiple images with each corresponding to individual occlusion patch position in batched manner. *Naive Incremental Inference-Exact* is a pure PyTorch based implementation of Algorithm 1 which does not use any GPU optimized kernels for memory copying where as Krypton does. For CPU based environments we only compare Krypton-Exact and Krypton-Approximate against *Naive* as no customization is needed for the pure PyTorch based implementation. For different datasets we set *adaptive drill-down* system tuning parameters differently. For *OCT* images the region of interest is relative small and hence a $r_{drill-down}$ value of 0.1 and a target speedup of 5 is used. For *Chest X-Ray* images the region of interest can be large and hence a $r_{drill-down}$ value of 0.4 and a target speedup of 2 is used. For *ImageNet* experiments we use a $r_{drill-down}$ value of 0.25 and a target speedup value of 3 which are also the Krypton default values. For all experiments τ is configured using a separate tuning image dataset ($n = 30$) for a target SSIM of 0.9. Visual examples for each dataset is shown in Appendix. Figure 10 presents the final results.

We see that Krypton improves the efficiency of the occlusion based explainability workload across the board. Krypton-Approximate for *OCT* results in the highest speedup with VGG16 on both CPU and GPU environments (16X for CPU and 34.5X for GPU). Speedups obtained by Krypton-Exact for all the datasets are same for all three CNN models. However, with Krypton-Approximate they result in different speedup values. This is because with *approximate inference* each dataset uses different system configuration parameters. *OCT* which is configured with a low $r_{drill-down}$ of 0.1, high target speedup of 5, and a *projective field threshold* value of 0.5 results in the highest speedup. Speedup obtained by Krypton-Exact on GPU with Inception3 model (0.7X) is slightly lower than one. However ResNet18 which has roughly the same theoretical speedup (see Figure 4) results in a higher speedup value (1.6X). The reason for this is Inception3's internal architecture is more complex compared ResNet18 with more branches and depth-wise stacking operations. Thus Inception3 requires more memory copying operations whose overheads

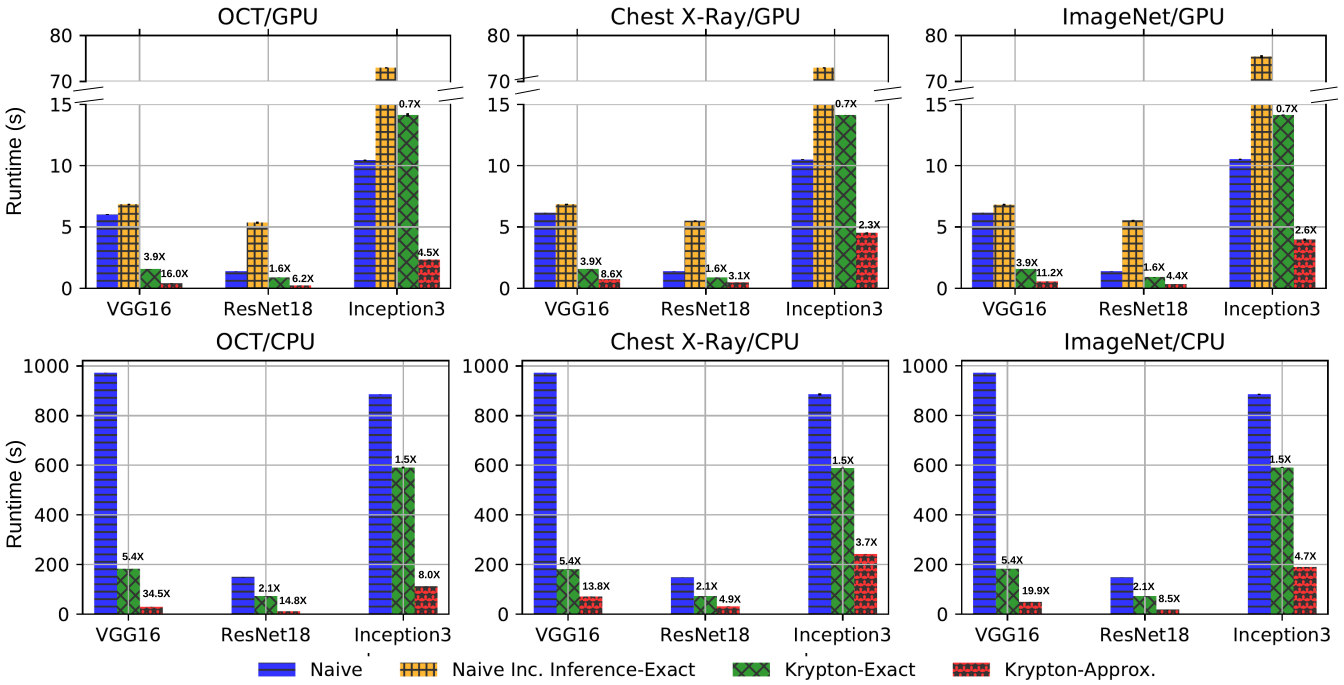


Figure 10: End-to-end efficiency achieved by Krypton over naive approaches.

are not captured by our theoretical speedup calculation. Overall compared to GPU environment Krypton results in higher speedups on the CPU environment though the actual runtimes are much slower. GPUs enable higher parallelism with thousands of processing cores compared to CPUs with several cores. Hence computations are much cheaper on GPU. Memory operations required by Krypton throttles the overall performance on GPU and hinders it from achieving higher speedups. On CPU environment as computational cost dominates the overall runtime, the additional overhead introduced by the memory operations does not matter much. Therefore on CPU Krypton achieves higher speedups which are closer to the theoretical speedup value. Overall Krypton offers the best efficiency on these workloads. This confirms the benefits of different optimizations performed by Krypton for improving the efficiency of the workload and thereby to reduce the computational and runtime costs. Bringing down the runtimes also make occlusion experiments more amenable for interactive diagnosis of CNN predictions.

5.2 Lesion Study

We now present the results of controlled experiments that are conducted to identify the contribution of various optimizations discussed in Section 4. The speedup values are calculated compared to the runtime taken by the current dominant practice of performing full inference for batches of modified images.

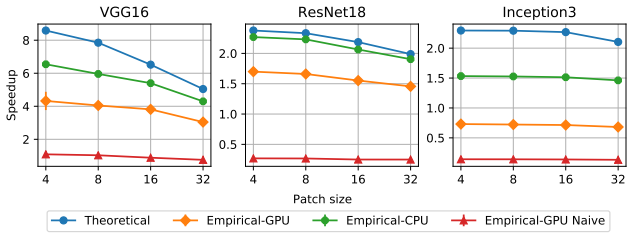


Figure 11: Theoretical versus empirical speedup for *incremental inference* (Occlusion patch stride $S = 4$).

Speedups from Incremental Inference.

We compare theoretical speedup and empirical speedups obtained by *incremental inference* implementations for both CPU and GPU environments. The patch sizes that we have selected cover the range of sizes used in most practical applications. Occlusion patch stride is set to 4. Figure 11 shows the results. Empirical-GPU Naive results in the worst performance for all three CNN models. Empirical-GPU and Empirical-CPU implementations result in higher speedups with Empirical-CPU being closer to the theoretical speedup value. As the occlusion patch size increases the speedups decrease.

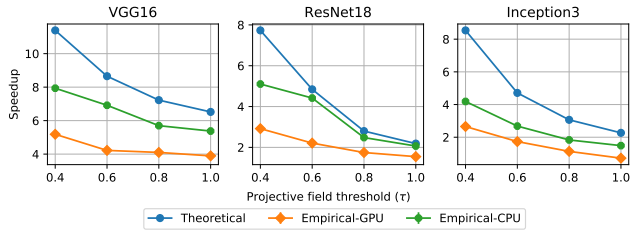


Figure 12: Theoretical versus empirical speedup for *incremental inference* with *projective field thresholding* (Occlusion patch size = 16×16, stride $S = 4$).

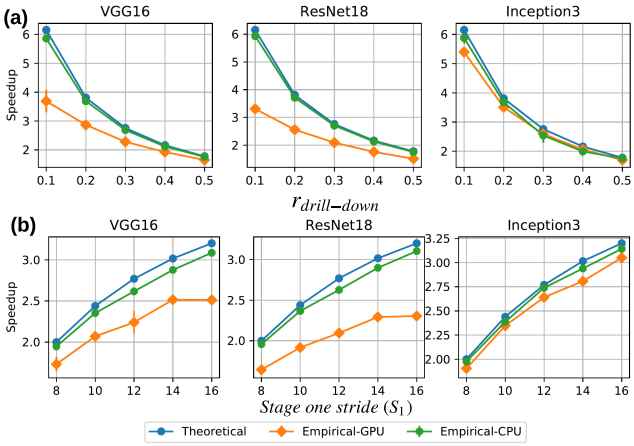


Figure 13: Theoretical versus empirical speedup for *adaptive drill-down* (Occlusion patch size = 16×16, stage two stride $S_2 = 4$, projective field threshold $\tau = 1.0$. For (a) S_1=16 and for (b) r_{drill_down}=0.25).

Speedups from Projective Field Thresholding.

We vary *projective field threshold* τ from 1.0 (no thresholding) to 0.4 and evaluate the speedups. The occlusion patch size used is 16 and the stride is 4. The results are shown in Figure 12. Empirical-CPU and Empirical-GPU both results in higher speedups with Empirical-CPU being closer to the theoretical speedup value. When τ decreases the speedups increase as the amount of computational savings increase.

Speedups from Adaptive Drill-Down.

Finally we evaluate the effect of *adaptive drill-down* on overall Krypton efficiency. The experiments are run on top of the *incremental inference* approach with no *projective field thresholding* (τ=1.0). $r_{drill-down}$ is varied between 0.1 to 0.5 fixing the stage one stride value S_1 to 16. Occlusion patch size is set to 16 and the stage two stride S_2 is set to 4. Figure 13 (a) shows the results. We also vary S_1 fixing $r_{drill-down}$ to 0.25. Occlusion patch size and the S_2 are set same as in the previous case. Figure 13 (b) presents the results. In both cases we see Empirical-GPU and Empirical-CPU achieve higher speedups with Empirical-CPU being very close to the theoretical speedup. On the CPU environment, the relative cost of other overheads is much smaller than the CNN computational cost. Hence on the CPU environment Krypton achieves near theoretical speedups for *adaptive drill-down*. Speedups decrease as we increase $r_{drill-down}$ and decrease S_1.

Summary of Experimental Results. Overall Krypton increases the efficiency of the occlusion based CNN explainability workload by up to 16X on GPU and 34.5X on CPU. Speedup obtained by *approximate inference* optimization (Krypton-Approximate) depends on the characteristics of the CNN model such as the effective growth of the projective field and the characteristics of the occlusion use case such as the relative size of the interesting regions on the image. Furthermore Krypton results in higher speedups on CPU environment compared to GPU environment. Increasing the occlusion patch size and τ decrease the speedup. Increasing $r_{drill-down}$ and decreasing S_1 also decrease the speedup.

6 OTHER RELATED WORK

7 CONCLUSIONS AND FUTURE WORK

REFERENCES

[1] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[3] Kaiming He et al. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Christian Szegedy et al. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[5] Olga Russakovsky et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[6] Daniel S Kermany et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5):1122–1131, 2018.

[7] Mohammad Tariqul Islam et al. Abnormality detection and localization in chest x-rays using deep convolutional neural networks. *arXiv preprint arXiv:1705.09850*, 2017.

[8] Sharada P Mohanty et al. Using deep learning for image-based plant disease detection. *Frontiers in plant science*, 7:1419, 2016.

[9] Farhad Arbabzadah et al. Identifying individual facial expressions by deconstructing a neural network. In *German Conference on Pattern Recognition*, pages 344–354. Springer, 2016.

[10] Yilun Wang and Michal Kosinski. Deep neural networks are more accurate than humans at detecting sexual orientation from facial images. 2017.

[11] Ai device for detecting diabetic retinopathy earns swift fda approval. https://www.aao.org/headline/first-ai-screen-diabetic-retinopathy-approved-by-f. Accessed September 31, 2018.

[12] Radiologists are often in short supply and overworked deep learning to the rescue. https://government.diginomica.com/2017/12/20/radiologists-often-short-supply-overworked-deep-learning-rescue. Accessed September 31, 2018.
[13] Kyu-Hwan Jung et al. Deep learning for medical image analysis: Applications to computed tomography and magnetic resonance imaging. *Hanyang Medical Reviews*, 37(2):61–70, 2017.
[14] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
[15] Forrest N Iandola et al. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.
[16] Andrew G Howard et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
[17] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *arXiv preprint arXiv:1706.07269*, 2017.
[18] Cafee model zoo. https://github.com/BVLC/caffe/wiki/Model-Zoo. Accessed September 31, 2018.
[19] Models and examples built with tensorflow. https://github.com/tensorflow/models. Accessed September 31, 2018.
[20] Hung Le and Ali Borji. What are the receptive, effective receptive, and projective fields of neurons in convolutional neural networks? *arXiv preprint arXiv:1705.07049*, 2017.
[21] Basic operations in a convolutional neural network - cse@iit delhi. http://www.cse.iitd.ernet.in/~rijurekha/lectures/lecture-2.pptx. Accessed September 31, 2018.
[22] Saskia EJ de Vries et al. The projective field of a retinal amacrine cell. *Journal of Neuroscience*, 31(23):8595–8604, 2011.
[23] Zhou Wang et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.
[24] Jia Deng, Wei Dong, et al. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.
[25] torch vison models. https://github.com/pytorch/vision/tree/master/torchvision/models. Accessed September 31, 2018.
[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A SPECIAL SITUATIONS WITH INCREMENTAL INFERENCE

It is important to note that there are special situations under which the actual output patch size can be smaller than the values calculated in Section 4.1. Consider the simplified one dimensional situation shown in Figure 14 (a), where the stride value[2] (3) is same as the filter size (3). In this situation, the size of the output patch is one less than the value calculated by Equation 16. However, it is not the case in Figure 14 (b) which has the same input patch size but is placed at a different location. Another situation arises when the input patch is placed at the edge of the input as shown in Figure 14 (c). In this situation, it is not possible for the filter to move freely through all filter positions as it hits the input

[2]Note that the stride value is generally less than or equal to the filter size.

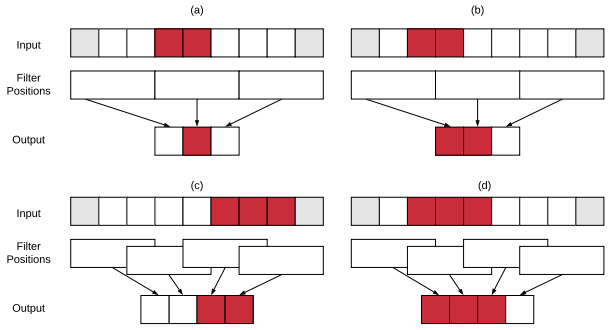


Figure 14: One dimensional representation showing special situations under which actual output size will be smaller than the values calculated by Equations 15 and ??. (a) and (b) shows a situation with filter stride being equal to the filter size. (c) and (d) shows a situation with input patch being placed at the edge of the input.

boundary compared to having the input patch on the middle of the input as shown in Figure 14 (c). In Krypton we do not treat theses differences separately and use the values calculated by Equation 16 and **??** as they act as an upper bound. In case of a smaller output patch, Krypton simply reads off and updates slightly bigger patches to preserve uniformity. This also requires updating the starting coordinates of the patches as shown in Equations 30 and 31. Such uniform treatment is required for performing batched inference operations which out of the box gives significant speedups compared to per image inference.

$$\text{If } x^O_{\mathcal{P}} + W^O_{\mathcal{P}} > W_O:$$
$$\begin{aligned} x^O_{\mathcal{P}} &= W_O - W^O_{\mathcal{P}} \\ x^I_{\mathcal{P}} &= W_I - W^I_{\mathcal{P}} \\ x^{\mathcal{R}}_{\mathcal{P}} &= W_I - W^{\mathcal{R}}_{\mathcal{P}} \end{aligned} \tag{30}$$
$$\text{If } y^O_{\mathcal{P}} + H^O_{\mathcal{P}} > H_O:$$
$$\begin{aligned} y^O_{\mathcal{P}} &= H_O - H^O_{\mathcal{P}} \\ y^I_{\mathcal{P}} &= H_I - H^I_{\mathcal{P}} \\ y^{\mathcal{R}}_{\mathcal{P}} &= H_I - H^{\mathcal{R}}_{\mathcal{P}} \end{aligned} \tag{31}$$

B FINE-TUNING CNNS

For *OCT* and *Chest X-Ray* datasets the three ImageNet pretrained CNN models are fine-tuned by retraining the final layer. We use a train-validation-test split of 60-20-20 and the exact numbers for each dataset are shown in Table 3. Cross-entropy loss with L2 regularization is used as the loss function and Adam [26] is used as the optimizer. We tune learning rate $\eta \in [10^{-2}, 10^{-4}, 10^{-6}]$ and regularization parameter $\lambda \in [10^{-2}, 10^{-4}, 10^{-6}]$ using the validation set and

train for 25 epochs. Table 4 shows the final train and test accuracies.

	Train	Validation	Test
OCT	50,382	16,853	16, 857
Chest X-Ray	3,463	1,237	1,156

Table 3: Train-validation-test split size for each dataset.

	Model	Accuracy(%)		Hyperparams.	
		Train	Test	η	λ
OCT	VGG16	79	82	10^{-4}	10^{-4}
	ResNet18	79	82	10^{-2}	10^{-4}
	Inception3	71	81	10^{-2}	10^{-6}
Chest X-Ray	VGG16	75	76	10^{-4}	10^{-4}
	ResNet18	78	76	10^{-4}	10^{-6}
	Inception3	74	76	10^{-4}	10^{-2}

Table 4: Train and test accuracies after fine-tuning.

C VISUAL EXAMPLES

Figure 15 presents occlusion heat maps for a sample image from each dataset with (a) *incremental inference* and (b) *incremental inference* with *adaptive drill-down* for different *projective field threshold* values. The predicted class label for *OCT*, *Chest X-Ray*, and *ImageNet* are DME, VIRAL, and OBOE respectively.

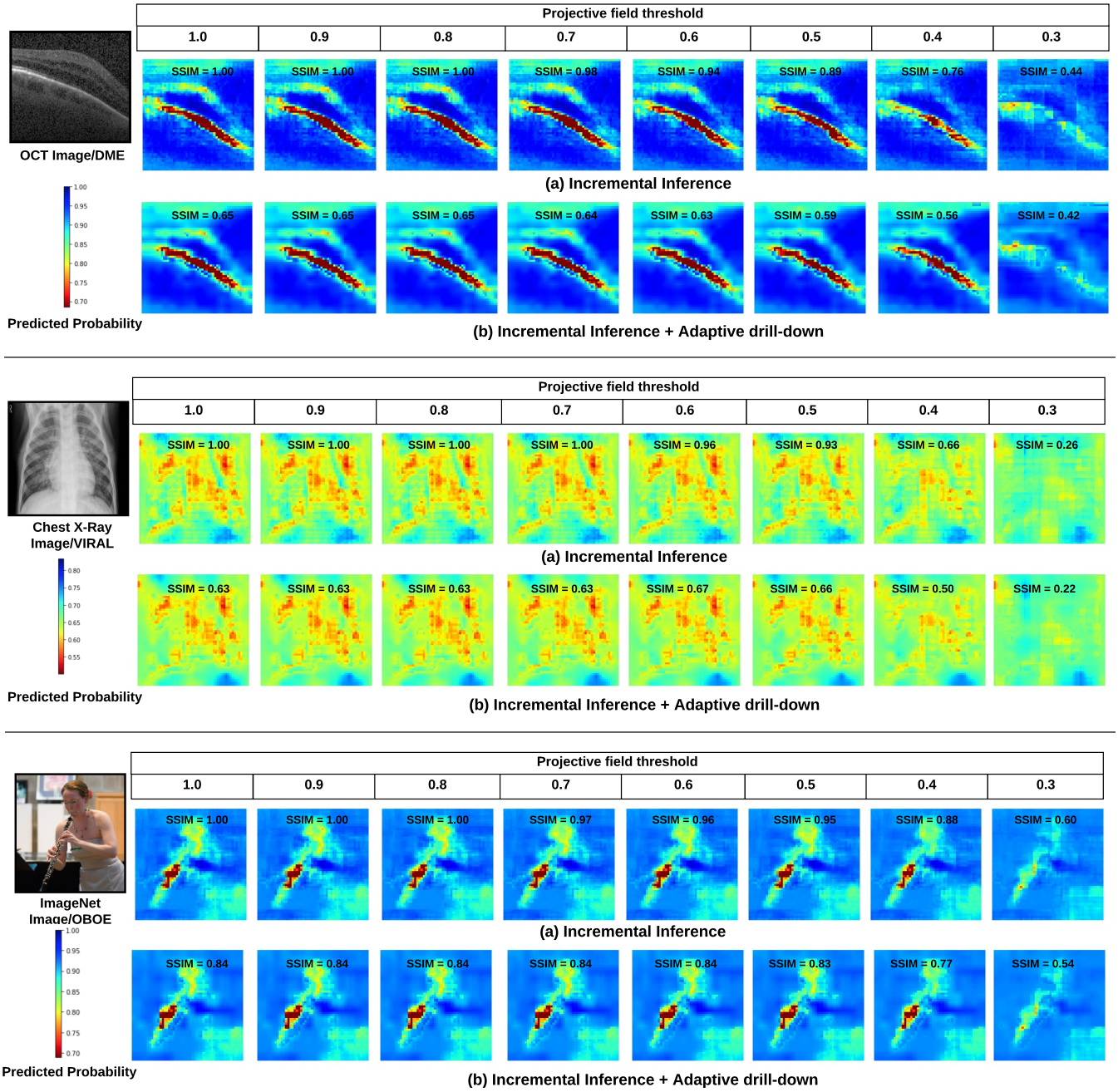


Figure 15: Occlusion heat maps for sample images (CNN model = VGG16, occlusion patch size = 16, patch color = black, occlusion patch stride (S or S_2) = 4. For *OCT* $r_{drill\_down} = 0.1$ and target speedup=5. For *Chest X-Ray* $r_{drill\_down} = 0.4$ and target speedup=2. For *ImageNet* $r_{drill\_down} = 0.25$ and target speedup=3).