Top-Down Networks

A coarse-to-fine reimagination of CNNs

Ioannis Lelekas

TU Delft

Top-Down Networks

A coarse-to-fine reimagination of CNNs

by

Ioannis Lelekas

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday March 13, 2020 at 3:00 PM.

Student number:	4742559	
Project duration:	May 1, 2019 – March 1, 2020	
Thesis committee:	Dr. J.C. van Gemert,	TU Delft, Supervisor
	Prof.Dr.Ir. M.J.T. Reinders,	TU Delft, Chair of the thesis committee
	Dr. F.M. Vos,	TU Delft, External thesis committee member

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

The current report "Top-Down Networks — A coarse-to-fine reimagination of CNNs" presents the work done for my master's graduation project. Research was conducted within the Computer Vision Lab of TU Delft, under the supervision of Dr. J.C. van Gemert; Dr. N. Tömen and Dr. S. Pintea have been my daily co-supervisors.

Having a strong inclination towards Deep Learning and Computer Vision, selecting my graduation project was one of the easiest decisions in my life. I enjoyed every moment of this nine-month journey and thus I want to express my sheer gratitude to Jan, for not only introducing me to the novel idea of Top-Down networks, but also giving me the opportunity of doing cutting-edge research under the supervision of leaders in the field. My hope is that our research paths will meet again some time in the future.

I would like to thank with all my heart, Nergis and Silvia for being always there for me, offering a critical eye against any problem I faced, no matter how simple or complex one. Their ideas gave me huge inspiration and vastly contributed to personal growth.

Furthermore, I would like to thank the thesis committee members Prof.Dr.Ir. Reinders and Dr. Vos, for the interest they have shown in my research, as well as for their evaluation of my work.

Moving on I want to express my sincere gratitude to my family for the endless emotional, moral and financial support throughout my life. To my brother and fellow engineer a big thanks, for always pushing each other forward. To my girl Marilena, there are actually no words to describe how thankful I am to her, for the constant support and sharing every moment with me. Last but not least, I owe a big thanks to my friends for always being there for me, but also a sorry for not having spent as much time with them, as they deserve and as I would want to have.

Ioannis Lelekas
Delft, March 2020

Top-Down Networks

Ioannis Lelekas
TU Delft
giannislelekas@gmail.com

Abstract

Biological vision adopts a coarse-to-fine information processing pathway, from initial visual detection and binding of salient features of a visual scene, to the enhanced and preferential processing given relevant stimuli. On the contrary, CNNs employ a fine-to-coarse processing, moving from local, edge-detecting filters to more global ones extracting abstract representations of the input. In the current paper we propose the extraction of top-down networks, by reversing the feature extraction part of the baseline, bottom-up architecture. This coarse-to-fine pathway, by blurring out higher frequency information and restoring it only at later stages, offers a line of defence against attacks introducing high frequency noise. High resolution of the final convolutional layer's feature map can contribute to the transparency of the network's decision making process, as well as favor more object-driven decisions over context driven ones and thus provide better localized class activation maps. The paper offers empirical evidence for the applicability of the method to various existing architectures, but also on multiple visual recognition tasks.

1. Introduction

It is well established that in human biological vision, perceptual grouping of visual features takes place based on Gestalt principles, where factors such as proximity, similarity or good continuation of features generate a salient percept [39]. Experimental studies in the domain of psychophysics have shown that figures salient in their global arrangement, are rapidly and robustly identified and segregated from the background in what is termed the "pop-out" effect [21]. This initial detection and binding of salient features into a coherent percept typically leads to enhanced and preferential processing by the visual system and is often described as stimulus-driven or bottom-up attention [49]. For relevant visual stimuli, the exogenously directed attention is sustained, and results in a more detailed visual evaluation of the object. This typical pipeline of perception and attention allocation in biological vision represents an efficient, coarse-to-fine processing of information [13].

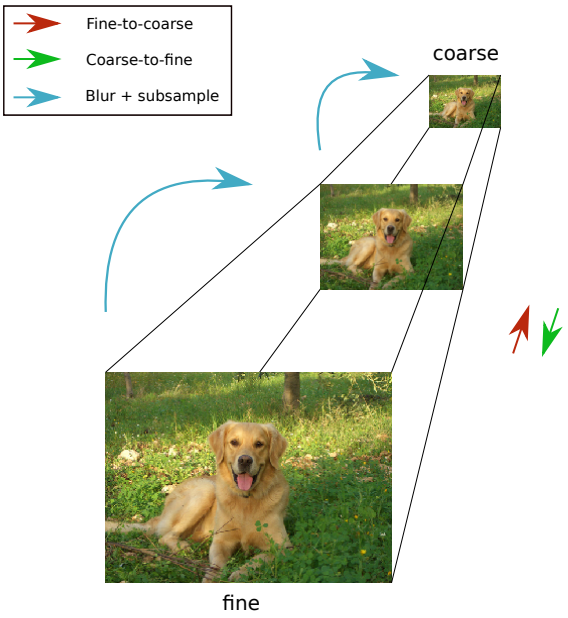


Figure 1. A coarse-to-fine versus fine-to-coarse processing pathway. The conventional fine-to-coarse pathway in a CNN, sacrifices localization for semantically richer information. Following the opposite path, the coarsest input is suitable for capturing the gist of the image; we can still see that this is clearly an image of a good boy/girl. Moving to finer representations of the input, higher frequency information is gradually restored, enabling now to carry out more fine-grained tasks such as scene segmentation, or classifying the input among different breeds of dogs. Original image is taken from [2].

In contrast, the organization of modern CNNs (Convolutional Neural Networks) do not incorporate this perspective [22, 35, 37, 11]. Conventional CNN architectures employ convolutional filters with a small spatial extent, which give rise to progressively increasing receptive field sizes through the network and thus perform hierarchical feature learning.

Starting from the high resolution input, information naturally propagates in a fine-to-coarse pathway, with early layers learning to extract local, shareable features whereas

deeper layers learn semantically richer and increasingly invariant representations. Notably, such architectures gradually diminish the spatial resolution of feature maps through pooling layers for computational efficiency.

In this paper, we propose the "reversal" of the conventional feature extraction hierarchy of CNNs. Our approach is partially motivated by biological vision and the fine-to-coarse versus coarse-to-fine processing, as demonstrated in figure 1. More specifically, we suggest the adoption of the latter, a coarse-to-fine processing of the input image, which can be interpreted as gradual focusing of visual attention. The reversed, or TD (Top-Down) hierarchy serves the purpose of first extracting the "gist" of a visual scene, emphasizing a more holistic initial representation, which is subsequently enhanced with higher frequency information useful for tasks requiring fine-grained, better localized information such as object detection and semantic segmentation.

We believe this paradigm shift may provide robustness against several different types of input noise. A growing body of literature since the seminal work of [38, 9] have shown that adversarial perturbations, which contain substantial high-frequency components, may cause well-generalizing models to misclassify. Based on the idea that suppressing higher frequencies in the input image can provide a first line of defence, we test our top-down models against multiple adversarial attacks.

Similarly, the lack of explainability of the decision making process of deep CNNs has recently emerged as an important concern and caveat of deep networks, with methods such as [51, 33] trying to shed some further light. We suggest that the coarse-to-fine processing scheme, which corresponds to feature maps with higher spatial resolution at deeper layers, will favor object-driven decisions over context-driven ones and provide better localized class activation maps. Consequently, we quantify the precision of Gradcam heatmaps of the top-down architecture in comparison to the baseline, or BU (Bottom-Up) models.

Considering the advantages of the coarse-to-fine pipeline, the current paper proposes the extraction and use of top-down networks, extracted from conventional bottom-up CNNs. The contributions of the current paper are: i) the proposal of a complete pipeline, instead of a fixed network model, for the extraction of TD networks corresponding to various diverse BU baseline models. This means that the proposed framework is versatile and directly applicable to existing architectures. ii) Extensive experimentation justifying the design choices made for the proposal of the pipeline. iii) Demonstration of enhanced robustness against certain types of adversarial attacks. Multiple attacks performed and evaluated, contributing to the transparency of the networks. iv) Increased transparency of the decision making process based on the fine output feature maps, as well as potent object localization capabilities. Trained models and scripts for recreating our experiments can be found at `https://github.com/giannislelekas/topdown`.

This paper principally aims to introduce a fresh perspective to the deep learning literature. We hope that the paper will lead to an attention shift and potentially rethinking of some design choices taken so far for granted. The remainder of this paper is organized as follows. Section 2 offers an overview of related current research. Section 3 describes in greater depth the proposed method, whereas valuable insight extracted from conducted experiments is presented in section 4. Finally, some discussion and concluding remarks are held in sections 5 and 6 respectively.

2. Related work

Coarse-to-fine processing. The coarse-to-fine processing is an integral part of efficient algorithms in Computer Vision. The iterative image registration by Kanade et al. [28] gradually refines registration from coarser variants of the original images, while in [15] a coarse-to-fine optical flow estimation method is proposed. Coarse-to-fine face detection is done by processing increasingly larger edge arrangements in [7], and coarse-to-fine face alignment using stacked auto-encoders is proposed in [47]. Efficient action recognition by using coarse and fine features coming from two LSTM (Long Short-Term Memory) modules is proposed in [41]. In [31] coarse-to-fine kernel networks are proposed, where a cascade of kernel networks are used with increasing complexity. Coarse-to-fine methods have been previously researched, both in terms of input resolution, as well as the manner of processing the input. Here, we also focus on coarse-to-fine image resolution, however we are the first to do this in a single deep neural network, trained end-to-end, rather than in an ensemble.

Bottom-up and top-down pathway. Many approaches exploit higher spatial resolution, and thus finer feature localization, which is crucial to the semantic segmentation task. The U-net [30] and FPN (Feature Pyramid Networks) [26] merge information extracted from bottom-up and top-down pathways, and thus combining semantically rich information of the bottom-up and the fine localization of the top-down. Similarly, combinations of a high-resolution branch with a low-resolution branch were proposed for efficient action recognition [5], for face hallucination [24], and depth map prediction [3]. Top-down signals are also used to model neural attention via a new backpropagation algorithm [46], and to extract informative localization maps for the classification task in Grad-CAM [33]. Similarly, we also focus on only top-down pathways where we slowly integrate higher levels of details, however our goal is improved prediction and not feature-map activation analysis.

Multi-scale networks. Merging and modulating information extracted from multiple scales is vastly popular [14, 20, 44, 45, 43]. In [45] the feature maps are resized by a consistent ratio to obtain a cascade of multiple resolutions. Incremental resolution changes during GAN (Generative Adversarial Network) training are proposed in [19]. Convolutional weight sharing over multiple scales is proposed in [1, 44]. Similarly [6] performs convolutions over multiple scales in combination with residual connections. In [20] convolutions are performed over a grid of scales, thus combining information from multiple scales in one response. Related, [36] combines responses over multiples scales, where filters are defined using 2D Hermite polynomials with a Gaussian envelope. A spatial pyramid pooling is proposed in [10] for aggregating information at multiple scales. In this work, we also extract multi-resolution feature maps, that restore high frequency information suppressed by the original network downscaling. However, we start from the lowest image scale and gradually add the high resolution details.

Beneficial effects of blurring. Blurring has been used to suppress high frequency information. A blurred input can lead to increased robustness as shown in [40, 50]. Models trained on blurred inputs exhibit increased robustness to distributional shift [18]. The work in [8] reveals the bias of CNNs towards texture, and analyzes the effect of blurring distortions on the proposed Stylized-ImageNet dataset. Blurring for anti-aliasing before downsampling, contributes to the preserving the shift invariance of CNNs [48]. By using Gaussian kernel with learnable standard deviation, [34] adapts the receptive field size. In this work we also rely on Gaussian blurring before downsampling the feature maps, to avoid aliasing effects, and as a consequence we observe improved robustness to adversarial attacks.

3. Top-down networks

Top-down (TD) networks mirror the baseline bottom-up (BU) networks, and reverse their feature extraction part. Information flows in the opposite direction, moving from lower to higher resolution feature maps. The initial input of the network corresponds to the minimum spatial resolution occurring in the BU baseline network. Downsampling operations are replaced by upsampling, leading to the coarse-to-fine information flow. By upscaling, the network can merely "hallucinate" higher resolution features, as no higher frequency information is available. To restore the higher frequency information, we use resolution merges, which combine the hallucinated features with higher frequency inputs, after each upscaling operation. Figure 2 depicts the difference between the BU architecture and our proposed TD architecture. The TD architecture relies solely on a top-down pathway.

3.1. Input and feature map resizing

To avoid hampering the performance of the network [48] and introducing artifacts, we perform the input downsampling by first blurring the inputs. For the upsampling operation we use interpolation followed by convolution. When upsampling, we have experimented with both nearest neighbour and bilinear interpolation, and have noticed improved robustness against adversarial attacks for the nearest neighbor interpolation. We have also considered the use of transpose convolutions, however we did not adopt these due to their leading to checkerboard artifacts in the output.

3.2. Merging low and high resolution

We considered three methods of merging the high resolution input with the low resolution information. The first method upsamples the low resolution input via a 1×1 convolution and uses an element-wise addition. The second considered method, concatenates the upsampled low resolution information with the high resolution information on the channel dimension. This is followed by a 3×3 convolution. We use 3×3 convolutions to expand the receptive field of the filters. Finally, the third merging option is a combination of the two aforementioned methods: we start by upscaling the low resolution input and then we add this to the high resolution input. This result is then concatenated with the high resolution input, and we add a 3×3 convolution at the end. Figure 3 shows these three options of merging the low and high resolution information. Options (2) and (3) introduce an equal number of additional paramaters, while method (1) is more memory efficient. Experimentally, we have observed that the combined method – option (3) – gives the best results and so adopted henceforth, however this comes at the cost of increased training parameters.

3.3. Filters arrangement

The feature extraction part of the TD network mirrors the BU: information propagates from lower to higher spatial dimensions in a TD network, while the number of filters shrinks with the increase in depth. The choice of expanding the number of filters at deeper layers in the BU network is efficiency-oriented. As the feature map resolution decreases, the number of channels increases, retaining the computational complexity roughly fixed per layer. Typically, in standard architectures the filters are doubled every time dimensions are halved [11, 35].

In our method we consider three options for deciding the number of filters per layer: the TD model where we apply the exact opposite process of BU by starting with many channels per layer and as the depth increases, we reduce the number of channels; $uniform$ (TD_{uni}) where the layers have a uniform number of filters; and $reversed$ (TD_{rev})

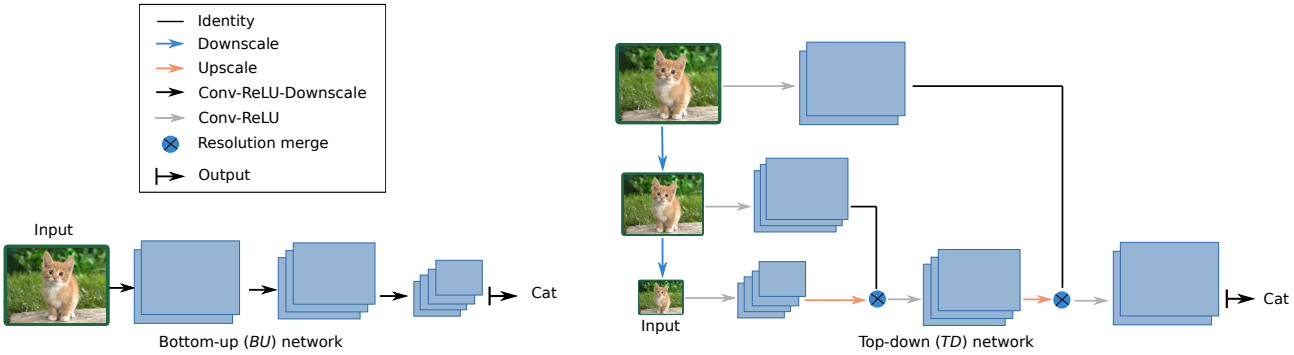


Figure 2. **Left:** The bottom-up (BU) baseline network. Feature maps decrease in spatial resolution with the network depth. **Right:** The proposed top-down (TD) network. The TD reverses the feature extraction part of the baseline network. It employs three inputs from highest to lowest scale, starts processing from the smallest resolutions and it progressively adds high resolution information.

which reverses the filter depth of TD and thus, follows the BU filter depth.

4. Experiments

We evaluate our top-down approach against the bottom-up baseline. In **Exp 1** we analyze the difference in performance on the MNIST, Fashion-MNIST and CIFAR10 classification tasks. We also evaluate the three different filters arrangement options proposed for the top-down model. In **Exp 2** we evaluate the robustness of our proposed models to a series of adversarial attacks applied on the same datasets. Finally, in **Exp 3** we apply the top-down model on a variant of Fashion-MNIST for segmentation.

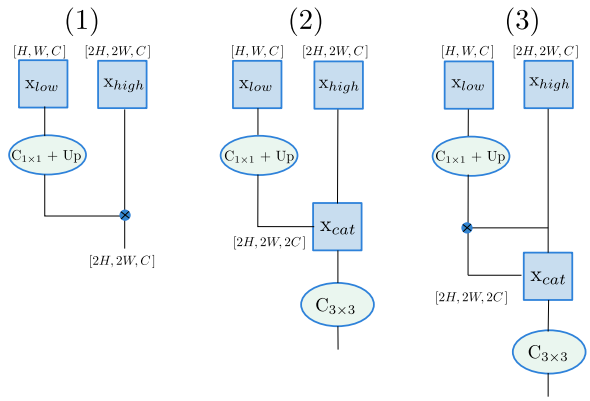


Figure 3. Merging low and high-frequency feature maps: (1) uses a 1×1 convolution followed by an element-wise addition; (2) uses a 1×1 convolution followed by a concatenation of the upscaled low-resolution information with the high resolution input; (3) combines the previous two options by using a 1×1 convolution, and an element-wise addition followed by concatenating this result with the high resolution input. For all methods the merged feature map is of size $[2H, 2W, C]$.

4.1. Exp. 1: Bottom-up versus top-down

Experimental setup. We compare our TD proposal with its respective baseline BU, with different configurations on MNIST, Fashion-MNIST and CIFAR10. For the simpler MNIST tasks we consider as baselines the "LeNetFC", a fully-convolutional variant of LeNet [23] and a lightweight version of the NIN (Network-In-Network) architecture [25] with reduced filters. The original architecture was used for the CIFAR10 task, along with the ResNet32 introduced in [11] incorporating the pre-activation unit of [12]. Batch Normalization [16] is used in all networks prior the non-linearities. The corresponding TD networks are defined based on the BU baselines. Table 1 depicts the number of parameters of different models. For TD we consider three variants: TD – which is mirroring the BU architecture also in terms of filter depth; TD_{uni} using uniform filter depth; and TD_{rev} where the filter depth of the TD is reversed, thus following the filter depth of BU. There is an increase in the number of parameters for the TD networks, because we need additional convolutional layers for merging the high and low resolution information.

Regarding training, we tried to abide by the setup found in the initial publications. We performed a linear search around these parameters[1] to tune the respective ones of TD networks. For all cases we train using SGD with momentum of 0.9 and a 3-stage learning rate decay scheme, dividing the learning rate by 10 at 50% and 80% of the total number of epochs. For the CIFAR10 dataset we test with and without augmentation — employing horizontal translations and flips. Runs are repeated four times, with dataset reshuffling and extracting new training and validation splits, for the extraction of mean values and standard deviations. Reported training times correspond to training on a single GTX1080Ti GPU and we have divided the mean total training time by the number of epochs.

[1] learning rate, batch size, weight decay.

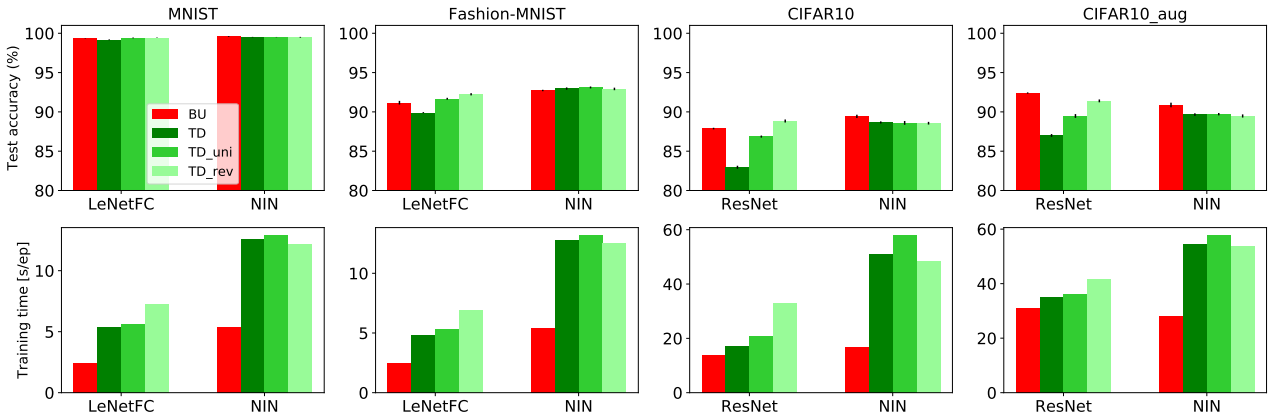


Figure 4. **Exp 1:** Relative comparison on MNIST, Fashion-MNIST, CIFAR10, and CIFAR10_aug (with augmentation) of mean test accuracies (**Top**) and training times (**Bottom**), between BU and the three different configurations of TD proposed in subsection 3.3. TD networks exhibit on par, some times even surpassing the baseline performance of respective BU. Regarding filter depth configurations, TD_{rev} report highest performance coming at the cost of both increased parameters and training time. Considering the small gap in performance and the increased cost for TD_{rev}, we adopt the TD configurations henceforth.

Model	#parameters			
	BU	TD	TD_{uni}	TD_{rev}
LeNetFC	8,790	14,452	23,486	57,846
NIN	62,094	215,118	217,182	214,206
NINext	969,822	3,405,918	3,434,910	3,388,446
ResNet	467,946	528,042	319,626	563,274

Table 1. **Exp 1:** Number of trainable parameters for different considered architectures. We denote by TD the method where we mirror the BU model, TD_{uni} the network with uniform number of filters per layer, and by TD_{rev} the network reversing the filter depth of TD and thus following the BU filter depth. There is an increase in the number of parameters for the TD networks, because they merge the high and low resolution information using additional convolutional layers.

Experimental analysis. Respective results are presented in figure 4. The TD network exhibit on par, even in some cases surpassing the corresponding baseline BU performance. As for the different filter depth configurations, TD_{rev} performs best at cost though of both increased complexity and training times. We adopt the TD variants henceforth, on account of the higher cost and the small gap in performance. The $TD - TD_{rev}$ accuracy gap rises for the ResNet architecture, where TD's test accuracy is roughly 83% (5.7% loss from respective BU), whereas the rotated version surpasses the baseline scoring higher than 89%. The decreased performance of TD is attributed to the network's architecture; the output of the final convolutional block is fed to a GAP layer, which then serves as input to a fully-connected layer which performs the final classification. The final block though has only 16 filters, thus leading to only 16 values used for the final classification, which is a not so potent input representation. By increasing the dimensionality of the final convolutional block to 32 we immediately got an increased performance of roughly 87%.

Apart from serving as a first BU vs TD crush test, **Exp 1** provides empirical evidence of the applicability of the proposed pipeline to different network architectures.

4.2. Exp. 2: Adversarial robustness

Experimental setup. The robustness of BU versus TD against various attacks is evaluated, where we attack the test set of the classifications tasks considered. We utilized Foolbox [29] for the attack generation. We emphasized on score-based and decision-based attacks, as they represent more realistic scenarios, where the attacker has no access to the gradients of the network. For all the attacks, the default parameters were used, as they led to successful attacks. Foolbox performs a linear search on parameters space, to find the minimal required perturbation to misclassify the input. To make this bound even tighter, we repeat each attack three times and keep the worst case for each sample[2], that is the minimum required perturbation for fooling the network.

Experimental analysis.

For the evaluation of adversarial robustness, figure 5 provides for each attack, plots of loss in test accuracy versus the $L2$ distance between the original and the perturbed input. TD networks are clearly more resilient against attacks introducing uncorrelated noise, due to the coarse-to-fine

[2]for attacks such as "SpatialAttack", "ShiftsAttack", "GaussianbBlurAttack" and "ContrastReductionAttack" involving no random number generation repetition leads to the same perturbations.

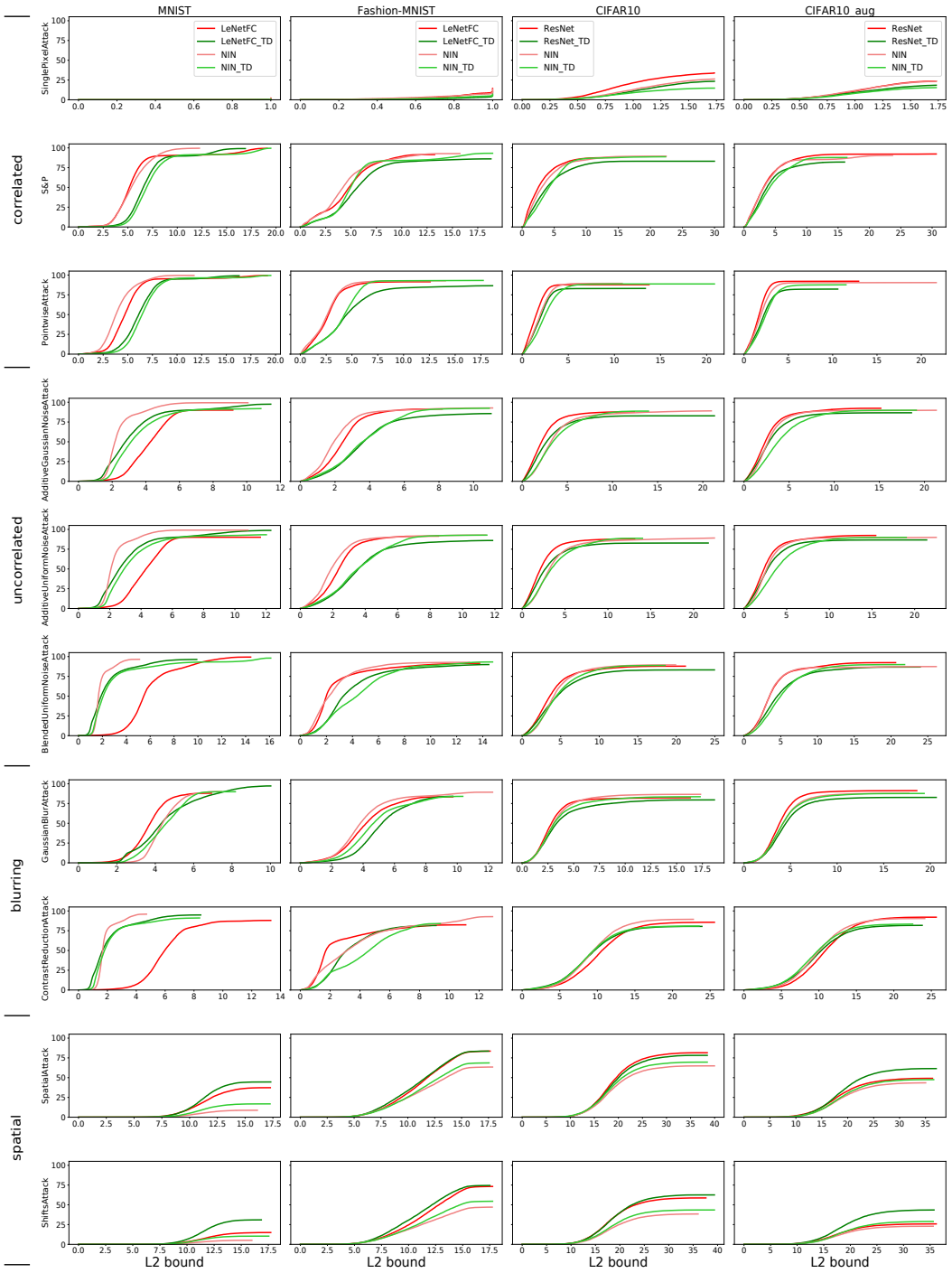


Figure 5. **Exp 2:** Comparison of adversarial robustness considering different datasets, models and attacks. The x-axis of each figure corresponds to the $L2$ distance between the original and the perturbed image and the y-axis to the introduced loss in test accuracy (original/unperturbed accuracy minus accuracy with the perturbed, within a given bound, images). A lower curve suggests increased robustness and green curves corresponding to TD are underneath the respective red curves of the BU networks in most cases. The TD networks appear to be more robust against both correlated and uncorrelated noise attacks due to the coarse-to-fine processing suppressing high frequency information on earlier stages. Additionally, the blurred downsampling baked in the network architecture offers enhanced robustness to the networks against the blurring attacks, as the network "sees" multiple scale of the input during training, thus being more resilient against resolution shifts. Finally, we do not measure enhanced robustness against spatial attacks.

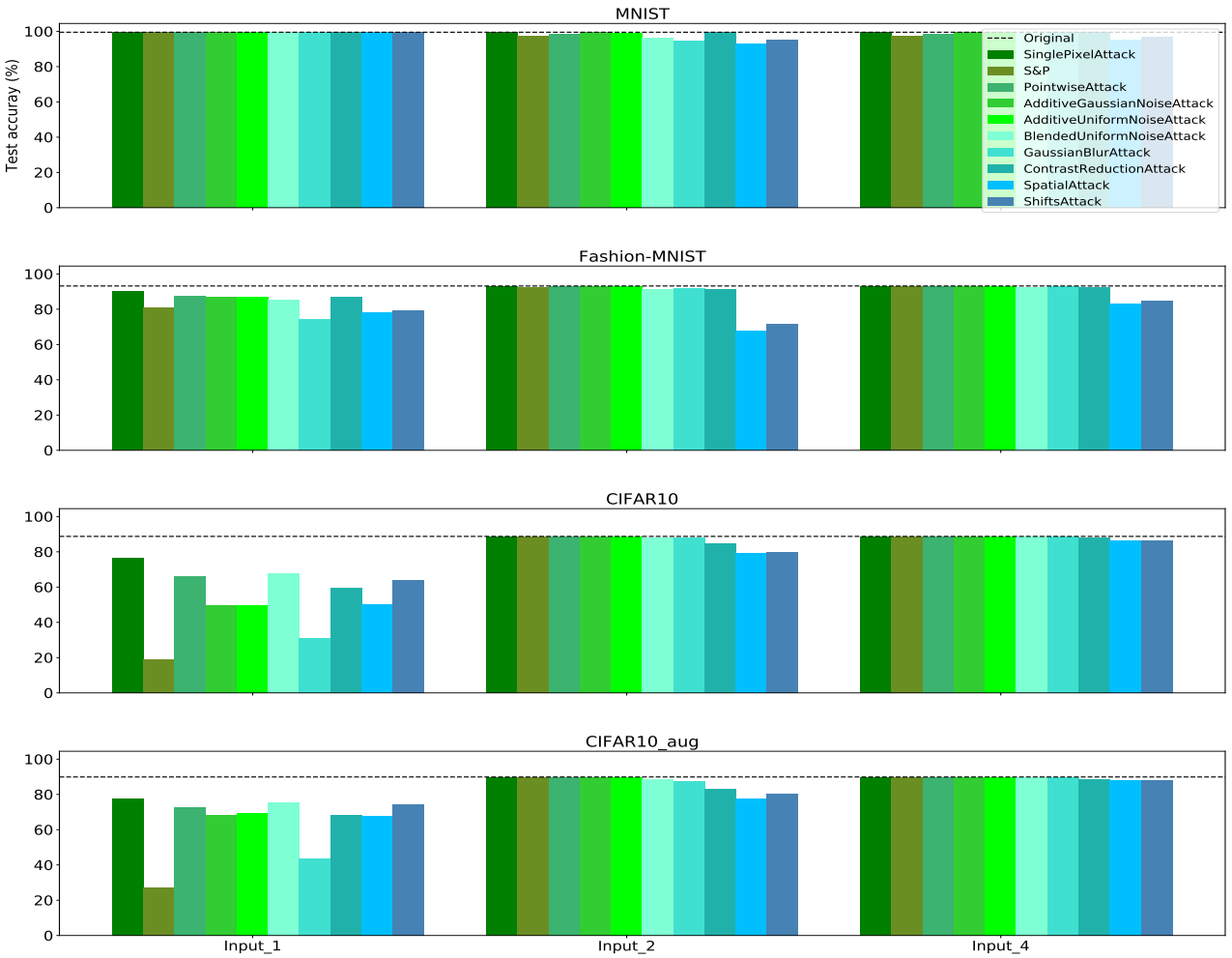


Figure 6. **Exp 2:** Testing the change in accuracy when feeding the perturbed images to a single input of TD NIN and re-evaluating the network; "Input_1", "Input_2", "Input_4" correspond to introducing perturbations to the highest, medium, lowest scale inputs respectively (refer to figure 2). Clearly, as more challenging datasets are considered highest vulnerability moves from the input of the medium to the one with the highest scale. This is attributed to the absence of information in the high frequency region for the simpler cases (i.e. MNIST).

processing adopted, with downscaled inputs diminishing or even eradicating the noise. Attacks introducing correlated noise are quite interesting; considering the Single-Pixel attack the perturbed pixel lies in smooth regions of the image. Thus a 0/1 in a region of 1s/0s is essentially a Dirac delta and based on the convolutional nature of CNNs this type of attack "pollutes" input with imprints of the learned filters[3], which gradually span a greater part of the input with more convolutions applied. Due to the highly correlated nature of the perturbation, the blurred downsampling can not eradicate completely the noise, but can still decrease the introduced pollution. On the contrary regarding BU networks, the noise gets propagated all the way down the network. The S&P (Salt&Pepper) noise and the Pointwise attack [32] are extensions to the Single-Pixel attack.

Regarding blurring attacks, the TD networks as expected are more robust, due to the blurred downsampling being baked in the architecture. Having "seen" multiple scales of the input during training, the network is more resilient against resolution changes. Anti-aliasing before downsampling is suggested to better preserve spatial invariance [48], hence we expected our networks to be more robust against Spatial [4] and Shifts attack, a variant performing only spatial shifts. However, no enhanced robustness is reported for TD networks. A substantial difference in robustness is reported for the ResNet architecture, which could be due to the performance gap measured between the TD variant and the baseline BU (please refer to section 4.1). Thus, we repeated the attacks on TD_{uni} and TD_{rev}

[3]in case of imperfect delta function, blurred versions of the filters are yielded.

versions of ResNet, which minimized the gap and surpassed baseline performance respectively. Results are presented in figure 11 in the appendix section A.a, where apart from enhanced robustness against spatial attacks we measured similar behaviour to the BU network regarding rest attacks, due to the increased number of filters at greater depth of the network, or equivalently higher scale maps.

For extra insight on TD robustness, we introduce the generated attacks to a single input of the TD networks, with results corresponding to the TD NIN architecture[4] presented in figure 6. For the simpler MNIST task the medium scale input of the network is the most vulnerable, which is mainly attributed to the absence of information in the high frequency region of the input's spectrum. Moving to more challenging Fashion-MNIST and CIFAR10 tasks, the high frequency input clearly becomes the easiest path to fooling the network. Perturbing multiple inputs naturally leads to increased degradation in performance. Respective results for perturbing two inputs are provided in section A.a and figure 12; as expected the higher and medium scale inputs are the most vulnerable ones.

4.3. Exp 3: Top-down for segmentation

Experimental setup. To evaluate TD performance, on the more fine-grained task of semantic segmentation, we constructed a 12-class[5] toy segmentation dataset based on the Fashion-MNIST task, with the global label mapped to the image pixels corresponding to the item. For segmenting the objects a threshold was empirically set to $t = 0.2$, whereas any pixel corresponding to values in $(0, t]$ formed the ignore class and 0 pixels constituted the background class. To increase slightly the complexity of the task we extracted a 2×2 mesh from the original images leading to a total of 15000 and 2500 training and testing images respectively. To deal with class imbalance, weights for each class k were extracted as in (1). Finally, we trained for standard categorical cross-entropy loss, while extracting mean IoUs.

$$w_k = \frac{1/pixels_k}{1/pixels_{k=background}} \tag{1}$$

Regarding the network topologies utilized for this experiment a lighter FCN [27] and a U-net-like [30] network served as the BU baselines. A TD contestant was extracted based on the U-net-like BU, with corresponding layers and comparable number of training parameters. There is a substantial resemblance between the proposed TD networks and the U-net, with out network relying solely though on a top-down pathway.

Experimental analysis. Figure 7 plots the IoU per class;

[4] similar conclusions are drawn for the other networks.

[5] original 10 classes, plus a background and an ignore class; the ignore class is not considered in the loss and IoU computation.

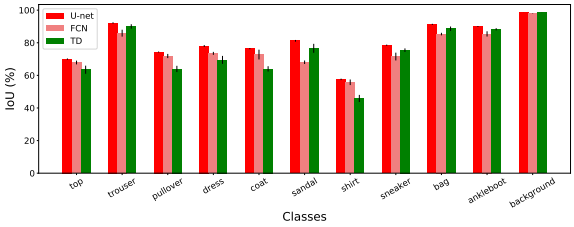


Figure 7. **Exp 3:** Segmentation results. TD is outperformed by the respective BU networks and mainly by the U-net architecture to which they have a substantial resemblance, but with small only margins. TD networks appear to be quite potent in the segmentation setting as well, thus solidifying the applicability of the proposed idea to more than one visual recognition tasks. The mean IoUs are roughly 81%, 76%, 75% for U-net, FCN and the TD network respectively.

the TD reaches comparable performance. Thus, we have shown that the proposed TD, are all-around networks that can be used to more than one visual recognition task.

5. Discussion

Gradcam heatmaps. Gradcam [33] provides class-discriminative localization maps, based on the feature maps of a convolutional layer, highlighting the most informative features for the classification task. The features of the ultimate convolutional layer are used, but using other layers is possible. The extracted heatmap is restored to the original image scale, thus producing a coarse map for the case of the BU. On the contrary, regarding a TD the corresponding feature maps' scale matches the scale of the input, hence Gradcam outputs a finer map.

The Gradcam heatmaps corresponding to a BU and TD network are provided in figure 8, corresponding to various layers of a "lighter" ResNet18 architecture [11] trained on the Imagenette dataset [17]; for further information about the setup please refer to appendix section A.b. TD's following an opposite, coarse-to-fine path is nicely demonstrated, starting from a coarser representation and gradually enriching it with higher frequency information. Hence, TD networks are not mirrors to the BU solely by an architectural point of view, but in their learning process as well.

More heatmaps, this time corresponding to the ultimate convolutional layer of the networks are visualized in figure 9. The coarse versus fine localization maps is the first point of the figure. Selection of the images with multiple objects corresponding to the global classes was intentional; the TD localized these objects more consistently than the BU.

Object localization. For a quantitative evaluation of the localization abilities of BU and TD, we used the MNIST and Fashion-MNIST datasets and the corresponding models trained from experiment 1. We then computed the IoU

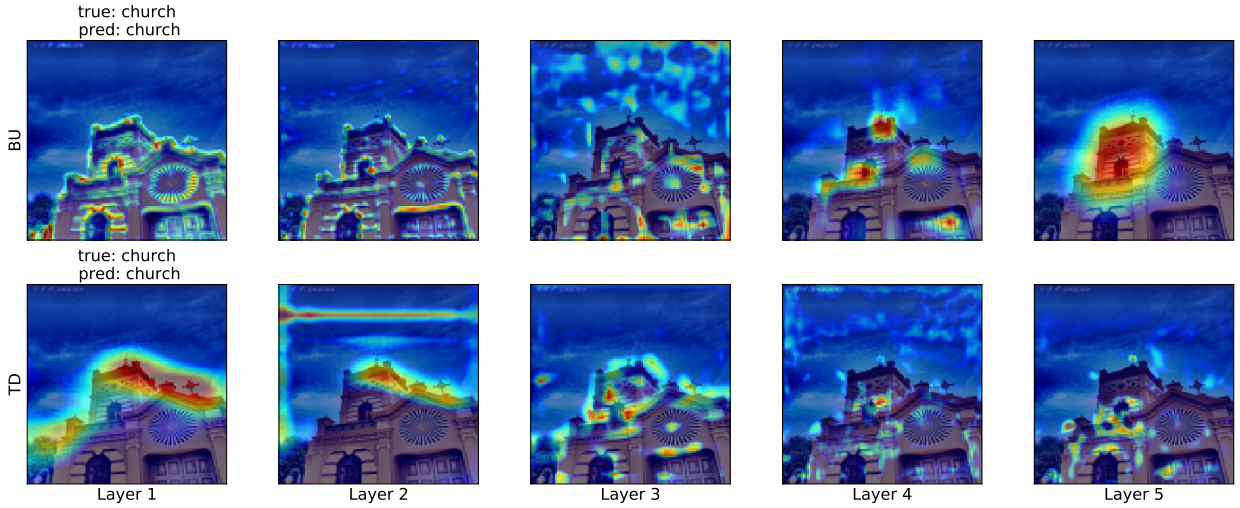


Figure 8. Fine-to-coarse versus coarse-to-fine processing; Gradcam heatmaps are visualized for BU ResNet18 versus its respective TD, trained on the Imagenette dataset [17]. Higher layer index means increased depth in the architecture; layer "1" corresponds to the input activation of the network's first group of convolutions, whereas layers "2-5" to the activations of each group's output. **Top**: The BU network, employing a fine-to-coarse processing. **Bottom**: the respective TD network following the opposite path, firstly looking into the image in a more holistic way and gradually adding higher frequency in deeper layers.

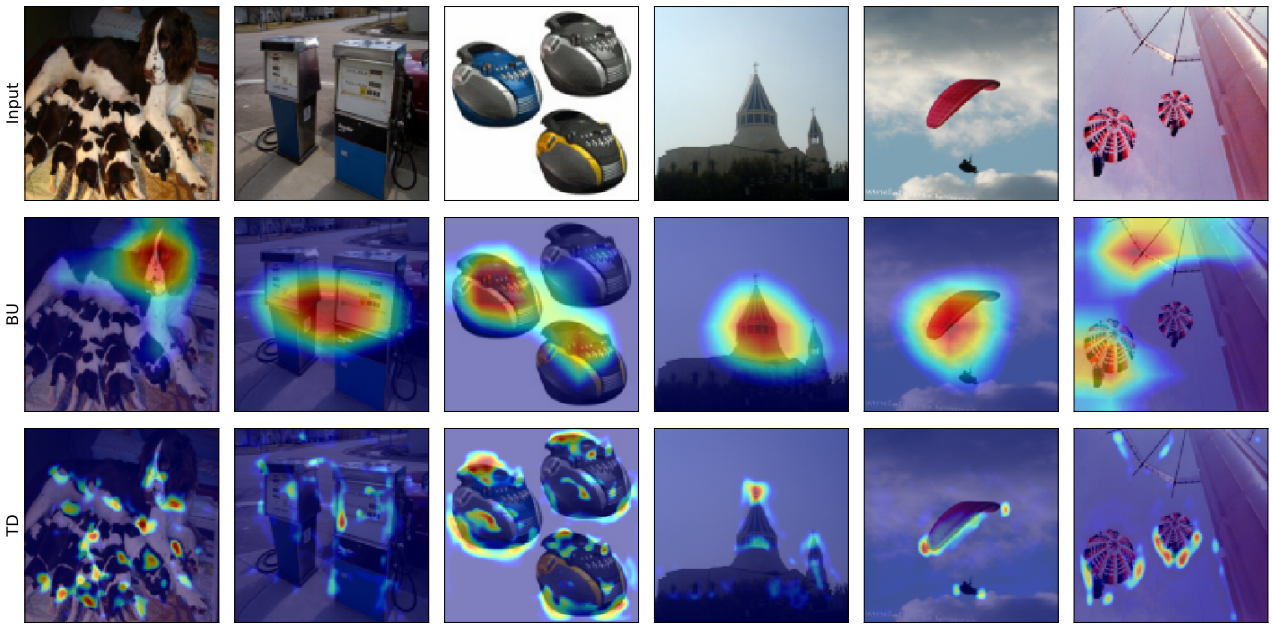


Figure 9. Gradcam heatmaps corresponding to the network's ultimate convolutional layer. **Top**: Gradcam heatmaps for the BU ResNet18. **Bottom**: Gradcam heatmaps for the TD ResNet18. Contrary to the coarse output of the BU, the TD network outputs high frequency feature maps, based on which the final classification is performed. The finer localization to the object of interest on behalf of TD, may proved to be invaluable to other settings as well (e.g. weekly-supervised setting).

between the object and the produced Gradcam heatmaps; for the object segmentation the threshold was empirically set to $t = 0.2$. For a fair comparison the samples correctly classified from both TD and BU are considered. Figure 10 provides the corresponding results; the TD variants consistently outperforms the BU. For the NIN architecture a 2.6%, 3.9% IoU difference is measured for MNIST and Fashion-MNIST respectively, resulting in a roughly 10% and 12% increase. The respective figures for the LeNetFC architecture are 0.38% and 0.64% meaning roughly 2% in-

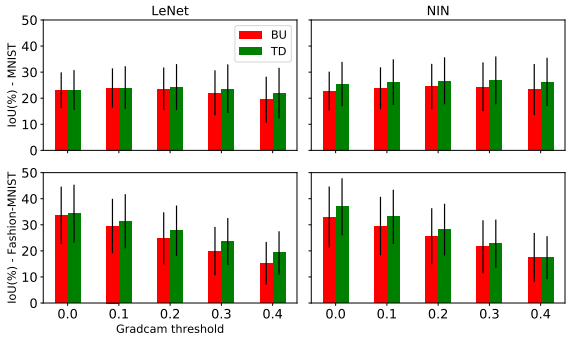


Figure 10. Mean IoUs reported for MNIST and Fashion-MNIST datasets and the LeNet, NIN architectures. Higher IoUs are reported for the TD networks suggesting the better localization of the object on their behalf. TD networks consistently outperform their BU baselines and in the case of the NIN architecture with a quite significant gap.

crease from the BU IoU.

Considering the fine Gradcam output and the potent object localization, the weakly supervised detection setting is an area where TDs could shine in the future. Before proceeding to an application though, some limitations should be firstly addressed.

Limitations. The current work mainly aimed at providing a fresh perspective to the CNNs' architecture which is mostly taken for granted. The coarse-to-fine pathway is biologically inspired, but also the core paradigm in efficient, iterative algorithms. However, expanding dimensions in increased depth leads to more convolutions thus both, higher computational complexity and memory footprint; despite there is no direct incompatibility, fully-connected layers can lead to the vast increase of parameters and magnify these limitations, hence convolutional layers should be preferred. To render TD networks directly applicable to large-scale datasets we should first deal with this issue. An easy workaround requiring no architectural adaptations, would be to employ mixed-precision training, which would decrease the memory requirements but computational complexity would still be an issue. Instead of increasing spatial resolution, we could use patches of the input of fixed dimensions; to select the most informative patch, Gradcam heatmaps could be utilized focusing in "hot" areas of the heatmap, or even self-attention [42].

Understandability of TD's processing pathway and a relative comparison with the respective BUs were the top concerns for all the conducted experiments. After all, increased transparency can eventually facilitate reaching State-of-the-art performance, by further understanding strong and weak points of the CNNs. After addressing the aforementioned limitations, we will try to enhance performance of the TD networks also aiming at some bold numbers, with the weakly supervised setting being a promising area. Having established the applicability of TD networks, it will be then time to show their muscles.

6. Conclusion

In the current work we revisit the architecture of conventional CNNs, aiming at shaking the waters in the field. We empirically demonstrate the applicability of the proposed TD networks adopting a coarse-to-fine information processing pathway, to various existing architectures, as well as multiple visual recognition tasks. TD networks exhibit enhanced robustness against certain types of attacks, attributed to the design choices made for their architecture. Additionally the high spatial dimensions of the ultimate feature map significantly contributes to the transparency of the network's decision making process, as well as to potent object localization skills. Limitations do exist, providing thus excellent field for future work. The sure thing is that TD networks are here to stay, so start holistic and welcome to the upside down!

References

[1] Shubhra Aich, Masaki Yamazaki, Yasuhiro Taniguchi, and Ian Stavness. Multi-scale weight sharing network for image recognition. *Pattern Recognition Letters*, 131:348–354, 2020.

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

[3] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[4] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. *CoRR*, 2017.

[5] Quanfu Fan, Chun-Fu Richard Chen, Hilde Kuehne, Marco Pistoia, and David Cox. More is less: Learning efficient video representations by big-little network and depthwise temporal aggregation. In *Advances in Neural Information Processing Systems*, pages 2261–2270, 2019.

[6] Yuchen Fan, Jiahui Yu, Ding Liu, and Thomas S Huang. Scale-wise convolution for image restoration. *Association for the Advancement of Artificial Intelligence (AAAI*, 2020.

[7] Francois Fleuret and Donald Geman. Coarse-to-fine face detection. *International Journal of Computer Vision*, 41(1-2):85–107, 2001.

[8] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *International Conference on Learning Representations*, 2019.

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations*, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916, 2015.

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

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on Computer Vision*, pages 630–645, 2016.

[13] Jay Hegdé. Time course of visual perception: coarse-to-fine processing and beyond. *Progress in neurobiology*, 2008.

[14] Sina Honari, Jason Yosinski, Pascal Vincent, and Christopher Pal. Recombinator networks: Learning coarse-to-fine feature aggregation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5743–5752, 2016.

[15] Yinlin Hu, Rui Song, and Yunsong Li. Efficient coarse-to-fine patchmatch for large displacement optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5704–5712, 2016.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, 2015.

[17] FastAI Jeremy Howard. The imagenette dataset. `https://github.com/fastai/imagenette`.

[18] Jason Jo and Yoshua Bengio. Measuring the tendency of cnns to learn surface statistical regularities. *CoRR*, 2017.

[19] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *International Conference on Learning Representations*, 2018.

[20] Tsung-Wei Ke, Michael Maire, and Stella X Yu. Multigrid neural architectures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6665–6673, 2017.

[21] I Kovács and B Julesz. A closed curve is much more than an incomplete one: effect of closure in figure-ground segmentation. *PNAS*, 1993.

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

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[24] Mengyan Li, Yuechuan Sun, Zhaoyu Zhang, and Jun Yu. A coarse-to-fine face hallucination method by exploiting facial prior knowledge. In *International Conference on Image Processing (ICIP)*, pages 61–65, 2018.

[25] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *International Conference on Learning Representations*, 2014.

[26] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[27] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[28] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[29] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *CoRR*, 2017.

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241, 2015.

[31] Hichem Sahbi. Coarse-to-fine deep kernel networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1131–1139, 2017.

[32] Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on mnist. *CoRR*, 2018.

[33] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International conference on Computer Iision*, 2017.

[34] Evan Shelhamer, Dequan Wang, and Trevor Darrell. Blurring the line between structure and learning to optimize and adapt receptive fields. *CoRR*, 2019.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

[36] Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. Scale-equivariant steerable networks. *International Conference on Learning Representations*, 2020.

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

[38] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *International Conference on Learning Representations*, 2014.

[39] Johan Wagemans, James H Elder, Michael Kubovy, Stephen E Palmer, Mary A Peterson, Manish Singh, and Rüdiger von der Heydt. A century of gestalt psychology in visual perception: I. perceptual grouping and figure-ground organization. *Psychological bulletin*, 2012.

[40] Haohan Wang, Xindi Wu, Pengcheng Yin, and Eric P Xing. High frequency component helps explain the generalization of convolutional neural networks. *CoRR*, 2019.

[41] Zuxuan Wu, Caiming Xiong, Yu-Gang Jiang, and Larry S Davis. Liteeval: A coarse-to-fine framework for resource efficient video recognition. In *Advances in Neural Information Processing Systems*, pages 7778–7787, 2019.

[42] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, 2015.

[43] Yichong Xu, Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, and Zheng Zhang. Scale-invariant convolutional neural networks. *CoRR*, 2014.

[44] Taojiannan Yang, Sijie Zhu, Shen Yan, Mi Zhang, Andrew Willis, and Chen Chen. A closer look at network resolution for efficient network design. *CoRR*, 2019.

[45] Chengxi Ye, Chinmaya Devaraj, Michael Maynord, Cornelia Fermüller, and Yiannis Aloimonos. Evenly cascaded convolutional networks. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4640–4647, 2018.

[46] Jianming Zhang, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. In *European Conference on Computer Vision*, 2016.

[47] Jie Zhang, Shiguang Shan, Meina Kan, and Xilin Chen. Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment. In *European conference on computer vision*, pages 1–16, 2014.

[48] Richard Zhang. Making convolutional networks shift-invariant again. *International Conference on Machine Learning*, 2019.

[49] Xilin Zhang, Li Zhaoping, Tiangang Zhou, and Fang Fang. Neural Activities in V1 Create a Bottom-Up Saliency Map. *Neuron*, 2012.

[50] Zhendong Zhang, Cheolkon Jung, and Xiaolong Liang. Adversarial defense by suppressing high-frequency components. *CoRR*, 2019.

[51] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

A. Appendix

A.a. Adversarial robustness

Figure 11 presents the robustness results for the CIFAR10-augmented and the ResNet architecture variants. Clearly, TD_{uni} and TD_{rev} variants exhibit enhanced robustness against spatial attacks, however, they also have similar to the BU behaviour against other attacks. This can be attributed to the increased number of filters at greater depth of the network, or equivalently increased scale of feature maps, thus greater contribution of the finer scales to the final output. However, finer scales are much more vulnerable against attacks. All in all, the reversal of the BU network for the extraction of the TD variant is not solely efficiency driven, keeping a roughly fixed computational complexity across layers, but also contributes to the network's robustness as well. Finally, we need to mention that the respective figure for the non-augmented CIFAR10 case tels the same story.

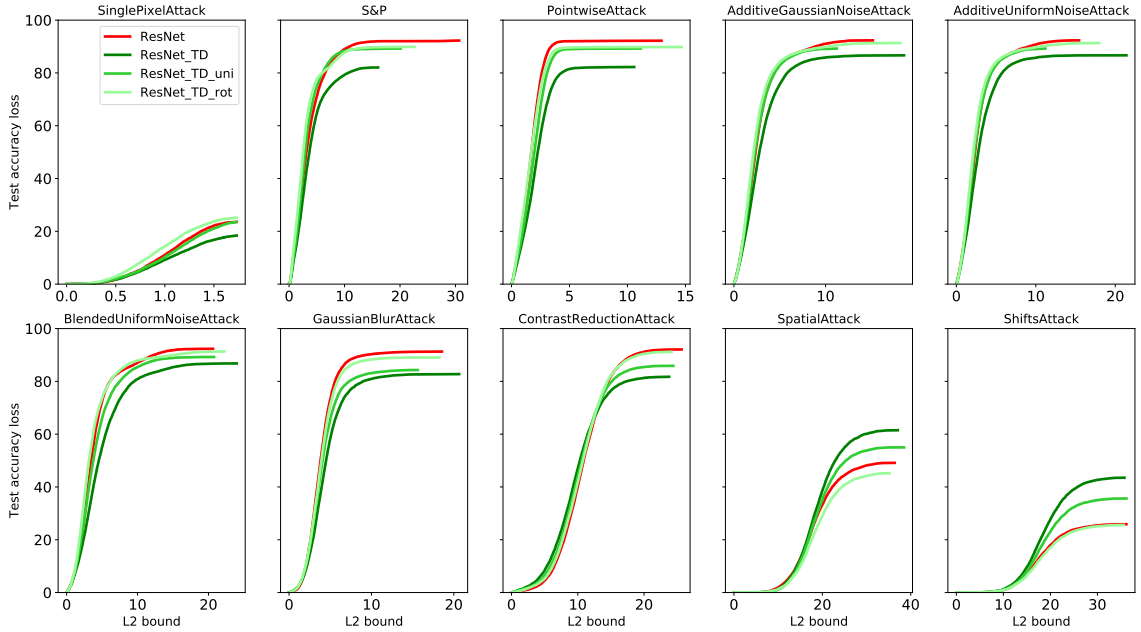


Figure 11. Test accuracy loss versus the $L2$ distance between original and perturbed input, for the CIFAR10-augmented and the ResNet architectures. Robustness is enhanced for the spatial attacks, but in general TD_{uni} and TD_{rev} variants exhibit similar behaviour to the BU baseline, which can be attributed to the increased filters at deeper layers.

Next, figure 12 presents the respective results for reintroducing the perturbation to two of the inputs of the network. Clearly, the highest and medium scale input are the most vulnerable one, except for the simpler case of the MNIST dataset. The absence or scarce information in the high frequency region, yields the medium and smallest scale inputs as the one with the highest impact.

A.b. Imagenette training

Imagenette [17] is a 10-class sub-problem of Imagenet [2], allowing experimentation with a more realistic task, without the high training times and computational costs required for training on a large scale dataset. A set of examples, along with their corresponding labels are provided in figure 13. The datasets contains a total of 9469, 3925 training and validation samples respectively. Training samples were resized to 156×156, from where random 128×128 crops were extracted; validation samples were resized to 128×128.

We utilized a lighter version[6] of the ResNet18 architecture introduced in [11] for Imagenet training, as this is a 10-class sub-problem, incorporating the pre-activation unit of [12]. Additionally, the stride s and the kernel extent k of the first convolution for depth initialization were set to $s = 1$ and $k = 3$ respectively. Regarding training, a 128×128 crop is extracted from the original image, or its horizontal flip, while subtracting the per-pixel mean [22]; the color augmentation of

[6] dividing the filters of the original architecture by 2.

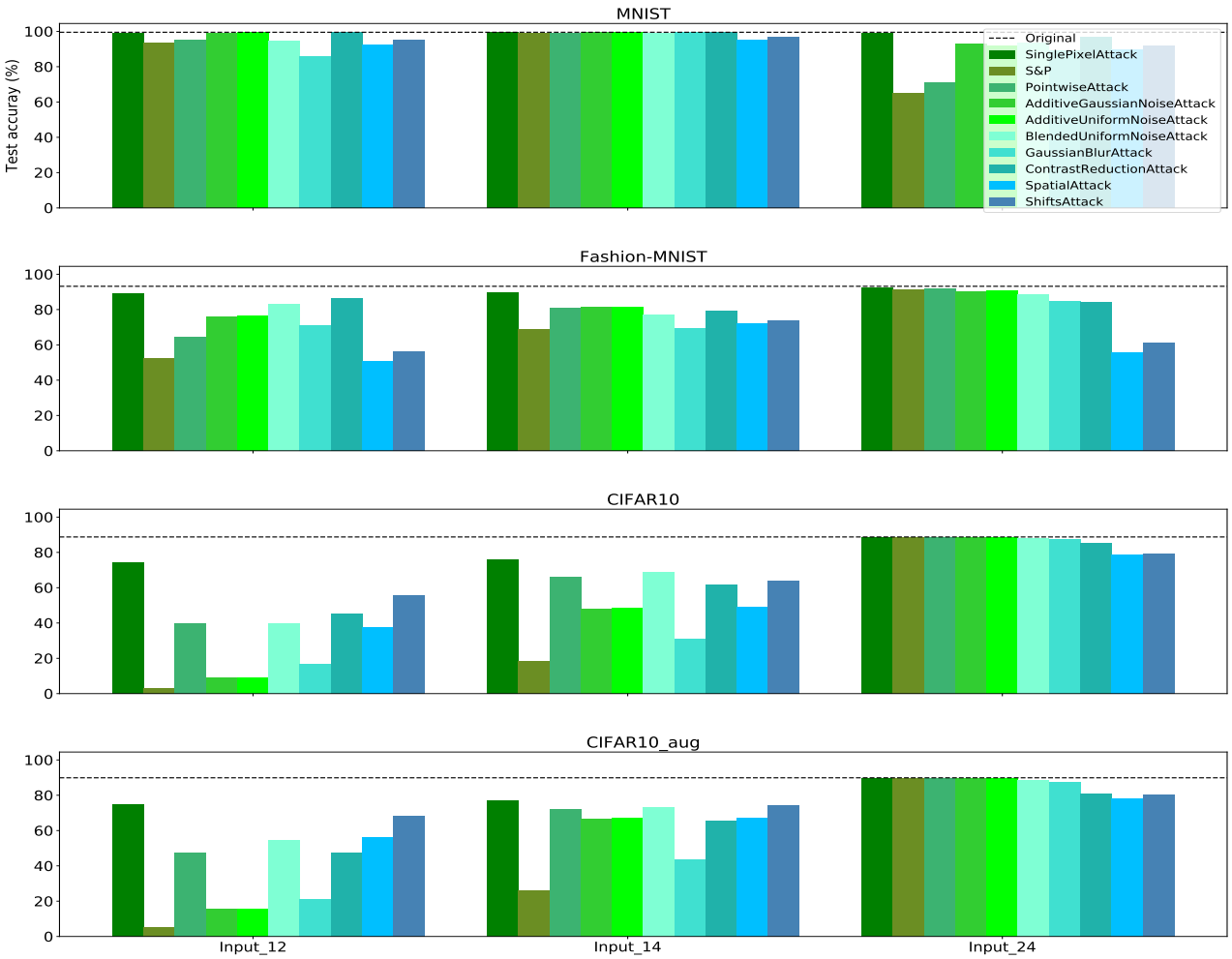


Figure 12. Reintroducing perturbations to two of the inputs of TD NIN. Clearly, perturbing the two highest scale inputs, "Input_12" has the highest impact. Regarding the case of the simpler MNIST and the information gathered in the low to mid frequency region, the medium and the smallest scale input have the highest impact instead.

[22] is also used. For the BU network a batch size of 128 is used and the network is trained for a total of 50 epochs with a starting learning rate of 0.1. As for the TD, increased memory footprint led to the reduction of the batch size to 64 and the adaptation of the starting learning rate and the total epochs to 0.05 and 80. We trained with SGD with momentum of 0.9 and a weight decay of 0.001; we also adopted a 3-stage learning rate decay scheme, where the learning rate is divided by 10 at 50% and 80% of the total epochs. Regarding performance, BU outperformed the TD variant as in the first experiment in section 4.1, by 5%. Gradcam is finally utilized for generating class-discriminate localization maps of the most informative features.

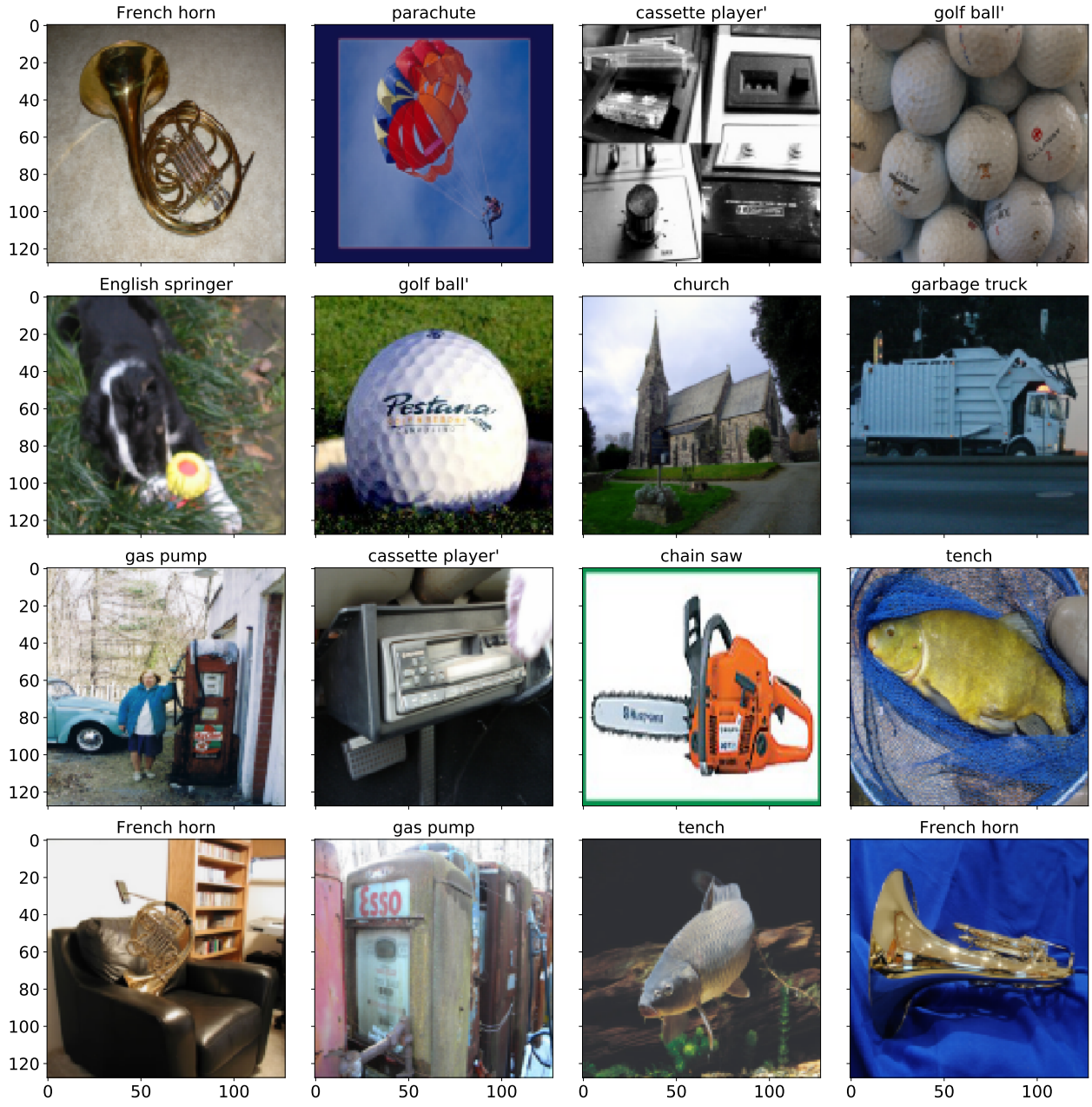


Figure 13. Validation samples from the Imagenette dataset [17], along with their corresponding labels. Samples are resized to 128×128.

Contents

1 Introduction

The pipeline of perception and attention allocation in biological vision represents an efficient coarse-to-fine information processing pathway [8]. Starting from the initial detection and binding of salient features of a visual scene, to the enhanced and preferential processing given relevant stimuli, driven by the bottom-up attention [27] and leading to enhanced evaluation of the scene. In simple words, a more holistic representation of the scene is first considered in order to extract the gist, which is then enriched with higher frequency information for carrying out more fine-grained tasks, such as scene segmentation and object detection.

However, CNNs (Convolutional Neural Networks) having reformulated State-of-the-Art in many visual recognition tasks [7, 12, 20] employ an opposite fine-to-coarse processing pathway. Filters of earlier layers operate as edge detectors to high resolution input. Successive pooling operations applied to the extracted features, decrease spatial dimensions while removing redundancy and increasing the receptive field of the filters, thus leading to global filters in deeper layers, yielding more abstract representations of the input. Moreover, a trade-off exists between better localization of features from earlier layers, versus the semantically richer features of deeper layers.

Inspired by the biological vision, in the current work we propose the reversal of the feature extraction part of CNNs for the extraction of TD (Top-Down) Networks; more specifically, we propose the adoption of a coarse-to-fine processing pathway commencing from evaluation of the gist and proceeding to its gradual refinement, by incorporating higher frequency information. The proposed paradigm-shift is a novel idea, which can test whether CNNs can effectively follow the opposite from the conventional learning path, potentially leading to more beneficial effects upon success. Coarse-to-fine processing can serve as an effective defence against adversarial attacks [6, 23] introducing different types of noise, simply by suppressing and only later adding back higher frequency information; removal of high frequency can effectively eradicate/decrease uncorrelated/correlated noise. Additionally, the high spatial dimensions of output feature map of the proposed TD networks can shed serious light to the decision process of CNNs [19, 28], while potentially favoring object-driven decisions over context-driven ones and provide better localized class activation maps.

The research question we will try to answer is *"Can we effectively switch to a coarse-to-fine information processing pathway in a conventional CNN's architecture, thus enforcing it to learn starting from more holistic and moving to finer representations of the input?"*. Furthermore and upon answering our main research question, we will try to answer the following questions; *"Do the TD networks lead to enhanced robustness against certain types of attacks?"*, *"Can this coarse-to-fine processing pathway adopted by the TD networks lead to increased transparency in the decision-making process, as well as to potent object localization skills?"*.

Stranger things are happening in the Computer Vision field, with the upside down having infiltrated the universe of CNNs as well. Making an analogy based on the popular series of Netflix "Stranger things"[1], a Top-Down architecture resembles an upside-down version of the Bottom-Up baseline. So with no further ado let's dive into the world of Top-Down Networks; *"Welcome to the Upside Down"*.

[1] the cover image is shared from the series' content.

2

Deep Learning

From hand-written digits recognition, to semantic segmentation and object detection, DNNs (Deep Neural Networks) have reformulated State-of-the-Art in many visual recognition tasks [7, 12, 13, 20, 24]. Technological advances in modern-hardware, as well as the abundance of excellent frameworks and readily available datasets, have turned training of extremely deep networks to a plug-and-play task.

The central aspect of DNNs, separating them from pre-existing ML (Machine Learning) models, is the automated feature extraction. There is no longer need for feature engineering, one simply needs to feed the dataset of interest in a DNN and the network will automatically extract a representation of the input to use for the visual recognition task. Therefore, a DNN may be used to extract features, on top of which a conventional ML model can be used for classification purposes. Does this mean that hand-crafted features and subsequently conventional machine-learning models are rendered obsolete? Absolutely not, the area where Deep Learning models can shine though is when it is not clear how to extract informative features for a task.

All these sounds perfect, but have the DNNs solved visual recognition? Despite having revolutionised the field, the task of visual recognition is far from being solved. Great depth and non-linearity severely degrade understandability, in many cases DNN's being considered black boxes. Thus, the scientific community has shifted its attention from State-of-the-Art and bold numbers, to enhancing transparency [19, 23, 25, 28]; after all, a better understanding can lead to better performance, but also devise effective defences against adversarial attacks[1] [6, 23]. Before going any deeper, let's try to break down these deep networks and shed some light in their operation. Content and visual aids for the current chapter, unless stated otherwise, are based on [5].

2.1. Convolutional Neural Networks

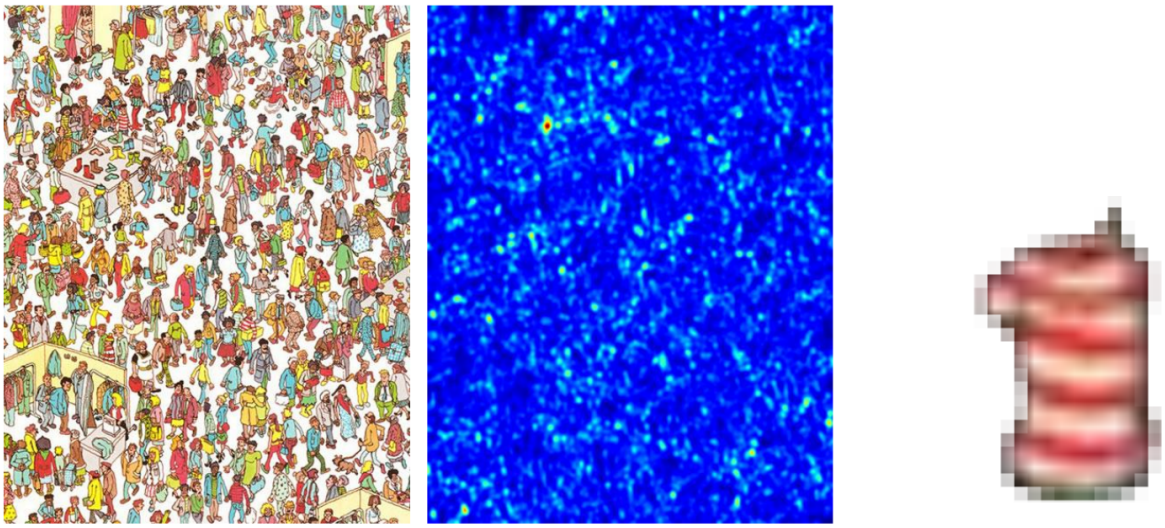
CNNs (Convolutional Neural Networks) are the culmination of the Deep Learning domain. As suggested by the name, the operation of convolution lies at their very core. CNNs are cascades of convolutions, pooling operations and non-linearities with the output of each layer serving as input to the subsequent one. Convolutional filters sliding over an image extract features. Next, pooling layers decrease spatial resolution, while enhancing the robustness of extracted features by discarding redundant information. Finally, activation functions render CNNs potent function approximators. Let us now get a closer look to the building components of a CNN.

2.1.1. Convolutions

Convolutional kernels are in essence the eyes of the network. A suitable kernel, operating on the input image in a sliding-window fashion, can render "demanding" detection tasks to a piece of cake as shown in figure 2.1. Thus, a neural network is trained for learning the most informative features for detecting the "Waldos" we are interested in. What if Waldo was now moved to a different location in the image of figure 2.1, could we still use the same kernel? Based on the shift invariance property of convolution any translations of the input has no effect to the operation's output, hence there is no need for another kernel. Convolutions as a design choice,

[1] please refer to chapter 4.

equip the network with translation invariance, in simple words meaning that the same filters can be used to detect an object regardless of its relative position in the input image. In any different case, there would be an explosion in the number of network's parameters, basically rendering its training practically infeasible[2].



(a) Convolution (b) Kernel

Figure 2.1: From left to right; input image, output of convolution, convolutional kernel. The "hot" circle in the north-west region of the output reveals the location of Waldo.

To avoid any confusion between the terms convolutional kernels and filters let us give a closer look to the later. A set of n convolutional kernels of size $k \times k$ comprises a 3D filter. The parameters of the filter are the size k and stride s of the sliding window. Applied now to an input of n channels, the filter yields a single output by performing element-wise multiplication and the addition. Clearly, this is a 3D operation, but due to the filter's sliding over 2D (height, width) the operation is known as 2D convolution. Now for the more general case of transition to layers with different depth, with the input and output layer having D_{in} and D_{out} channels respectively. We need to apply D_{out} filters, each of them comprised by a set of D_{in} kernels. Each of the filters yield a single output channel, thus leading a total of D_{out} channels [2].

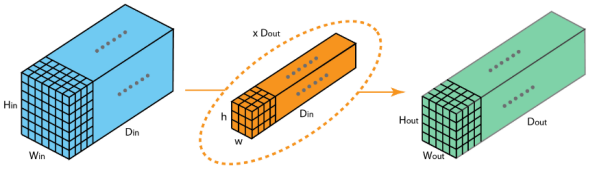


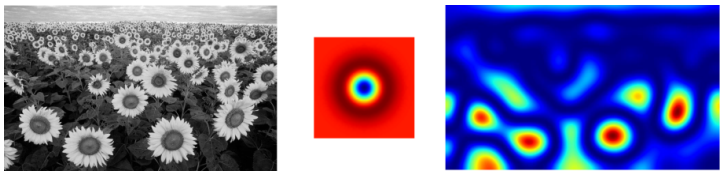
Figure 2.2: Transition between input and output layer having D_{in} and D_{out} channels respectively by applying D_{out} filters. Each filter has D_{in} kernels, yielding a single output channel, thus leading to a total of D_{out} channels [2].

There are other types of convolutions incorporated in many network architectures. Dilated convolution is a nice example, introduced to model longer-range dependencies of the input. Transpose convolution or deconvolution is another, where it does the opposite from a standard convolution, offering a one-to-many relationship from the input. Finally, a strided convolution offers an alternative from the max-pooling operation, the dominant option for feature maps downscaling. In the strided convolution, a stride of $s > 1$ is applied, which apart from the extraction of features leads to the down-scaling of the input. For a very thorough overview of the different convolutions please refer to [2].

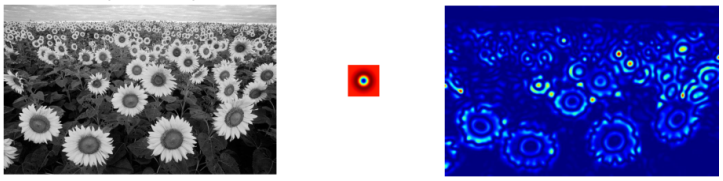
Filters across the layers of a trained CNN learn to detect different objects. Starting from the earlier layers and high spatial resolutions, the filters basically operate as edge detectors, looking for fine details in the image. In deeper layers where resolution has shrinked, fusing these fine details and the filters now spanning a greater part of the image, the network detects more abstract patterns (classes). It has to be noted at this point

[2] as any ML model, more trainable parameters mean more data needed for training or equivalently more prone to overfitting; for further details of overfitting and methods against it please refer to section 2.3.

that images are resized when they are propagated down the network, but the kernels' sizes remain fixed $k \times k$ based on the equivalence demonstrated in figure 2.3. Resizing the image instead of the kernels, prevent from the rise in the number of the trainable parameters.



(a) convolving with a larger kernel



(b) convolving with a smaller kernel

Figure 2.3: For both a) and b), on the left lies the input image, in the middle the convolutional kernel and on the right the output. Convolving with a smaller image is equivalent to convolving with a larger kernel. Based on that, instead of resizing the kernel, CNNs resize the input image.

2.1.2. Activation function

Neural Networks (NNs) are considered universal function approximators, a property which may be attributed on great extent to the application of activation functions, or equivalently the non-linearities. With linear activation functions, simple linear mappings could only be learned with the NNs being merely a Linear regression model. Non-linear activation functions allow learning of much more complicated mappings from input to output data.

Why are activation functions so crucial though to the Neural Networks? For the sake of simplicity let us consider a simple feed-forward network. Any conclusions drawn also apply to CNNs as well, after all CNNs are just limited versions of feed-forward networks based on the equivalence between matrix multiplication and convolution, along with the sparsity of the Toeplitz matrix. Thus, considering a feed-forward network, each neuron computes a weighted sum of its input x as in (2.1), given learned weights W and bias b. However, the output y can have any value in range $(-\infty, \infty)$ and it is difficult to assess whether this neuron is in activate state (should fire). So, in simple words the activation functions squash y in a given range in order to assess the state of the neuron and what should be propagated to the children of that neuron. Some of the most commonly used types of activation functions are presented in figure 2.4.

$$y = Wx + b \tag{2.1}$$

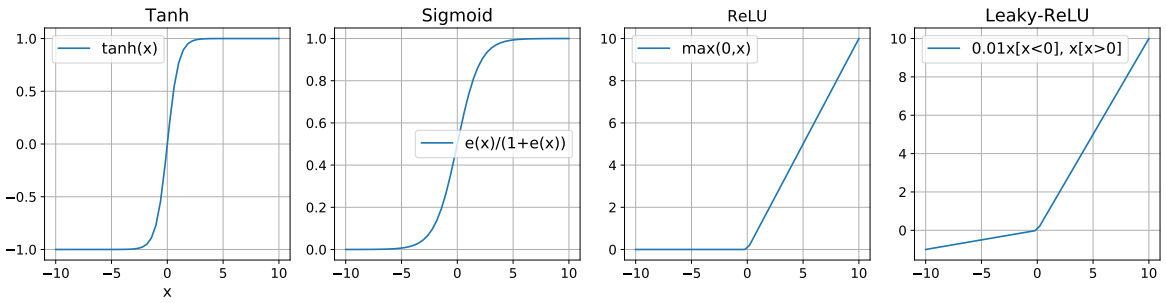


Figure 2.4: Activation functions

Compatibility with the Backpropagation algorithm[3] being vastly used for the end-to-end training of CNNs, dictates differentiability of the activation functions. However, ReLU and its variant Leaky-ReLU are differentiable everywhere apart from $x = 0$; does this mean that they are not eligible for gradient-based optimization?

[3] which we examine in subsection 2.2.3.

The answer is obvious, since both these non-linearities are commonly used in various network architectures and the reason they are still eligible is that numerical methods used for the training process are subject to numerical error, hence the absolute zero is practically never reached.

2.1.3. Pooling

The next step in each operational stack of CNNs is the pooling operation. Pooling is employed for decreasing spatial resolution and it is effectively capturing the "gist" over a region of the image. Localization is sacrificed for the extraction of more robust features.

Pooling like convolution works in a sliding window fashion, where an operator T is applied over a pooling region of size $k \times k$. The stride s of the pooling window determines the down-sampling ratio applied to an image, e.g. for $s = 2$ the spatial resolution of an image will be reduced to half. The most common options are max and average pooling, where $T = Max()$, $T = Average()$ respectively.

To sum up the current section with the building blocks of a CNN, the entire pipeline of a layer is visualized in figure 2.5. The output of the layer will serve as input to the subsequent one.

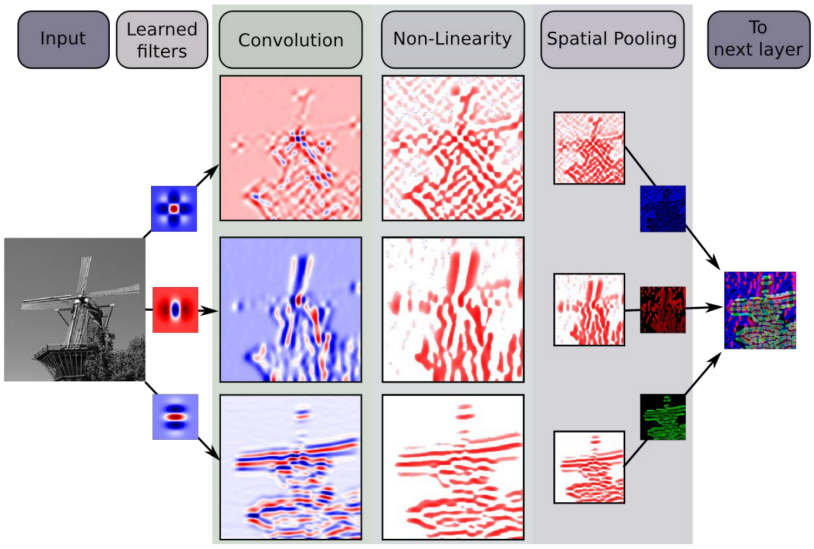


Figure 2.5: Pipeline of a CNN's layer

2.1.4. Fully-convolutional Neural Networks

CNNs have rendered the need for hand-crafted features obsolete. The representation of the input learned by a CNN, may then be used by any conventional ML model. Typically, fully-connected layers were used for the final classification at the cost though of vast increase in parameters, consequently affecting the generalization abilities of the network as well[4]. Fully-convolutional variants have yielded competitive results in object recognition tasks [15, 21], simply by replacing the fully-connected layers with convolutional ones. Apart from significant decrease in the number of trainable parameters, fully-convolutional networks generate feature maps instead of a set of values, which can also be used in the more fine-grained image segmentation setting [15].

2.1.5. Bottom-Up hierarchy

Learned filters operate in a different way across the layers of a trained CNN. Starting from local, edge detecting filters with their RFs (Receptive Fields) spanning only a small region of the input image, thus looking for fine details. Based on the equivalence demonstrated in figure 2.3, extracted feature maps propagated down the network are downscaled through successive series of pooling operations, leading to the decrease of spatial resolution and consequently the increase of the receptive fields of the filters. Looking now at a greater part of the input while incorporating features extracted on earlier layers, the filters are more global, extracting abstract representations of the input. Moreover, there is a trade-off between spatial resolution and semantic richness of the extracted features, with filters of earlier layers offering better localization due to high spatial

[4] for further information on overfitting please refer to section 2.3.

resolution, while features extracted in deeper layers are of higher semantic information. All in all, this fine-to-coarse pathway constituting the Bottom-Up hierarchy of any conventional CNN, renders feature extraction a productive process, starting from fine details and gradually merging them into more abstract representations.

This hierarchy is visualized in figure 2.6, shared from Zeiler et. al. [25]. For each layer of a trained network, the top nine activations corresponding to a random subset of the validation set feature maps are projected to pixel space and visualized, along with the corresponding image patches. Regarding their method, deconvolution, for projection to pixel space please refer to [25]. The visualization clearly demonstrates the gradual transition of learned filters, from local, shared edge detectors, to more global and abstract patterns representers.

2.2. Training a CNN - Loss functions

Training a neural network is an intricate, iterative process which can be mainly broken down to three major parts; the forward pass, the extraction of the loss and finally the backward pass with the parameters update.

2.2.1. Forward pass

Nothing special here, the input is modulated in a similar manner to an MLP (Multi-Layered Perceptron). Assuming weights W and bias b, the output y of a node with input x is given as in (2.1). The output of the node will serve as input to its children nodes and so forth till the output of the network.

2.2.2. Loss function

Training a neural network aims at learning the trainable parameters, namely weights and biases of the filters, essentially by minimizing a loss function. A forward pass is executed and a loss is measured based on actual and predicted labels and the current network's parameters. A typical choice for a loss function is the CE (Cross-Entropy) loss[5] L_{CE} (2.2), where y_{true} the actual label of a sample in one-hot format for a K-class classification problem. Next, y_{pred} is the predicted label, coming from the output of the softmax function applied to the logits z (2.3), the output of the network, in order to turn these scores into a proper probability distribution. Equations (2.2) and (2.3) are an immediate extension from the respective equations of a binary classification problem.

$$L_{CE}(y_{true}, y_{pred}) = -\sum_{k=1}^{K} y_{true,k} log\, y_{pred,k} = -y_{true}^{T}(log\, y_{pred}) \tag{2.2}$$

$$y_{pred,n} = softmax(z_1, z_2, \cdots, z_K) = \frac{e^{z_k}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.3}$$

Given a set of m training examples $X = \{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$, drawn i.i.d[6] from the true yet unknown distribution $p_{data}(x)$ (using the empirical distribution $\hat{p}_{data}$ instead, drawn from training samples) and a parametric family of probability distributions $p_{model}(x;\theta)$ corresponding to a model with parameters θ, minimizing the cross-entropy H is equivalent to minimizing negative log-likelihood (maximize maximum likelihood) (2.4), or the KL divergence D between the two distributions (2.5).

$$argmin_{\theta} \mathbb{E}_{x\sim\hat{p}_{data}}[log\, p_{model}(x)] \tag{2.4}$$

$$\begin{aligned} &D(p_{data}, p_{model}) = [log \frac{p_{data}}{p_{model}}] \\ &H(p_{data}, p_{model}) = H(p_{data}) + D(p_{data} || p_{model}) \\ &argmin_{\theta} \mathbb{E}_{x\sim\hat{p}_{data}}[log\, p_{data}(x) - log\, p_{model}(x)]. \end{aligned} \tag{2.5}$$

2.2.3. Backpropagation

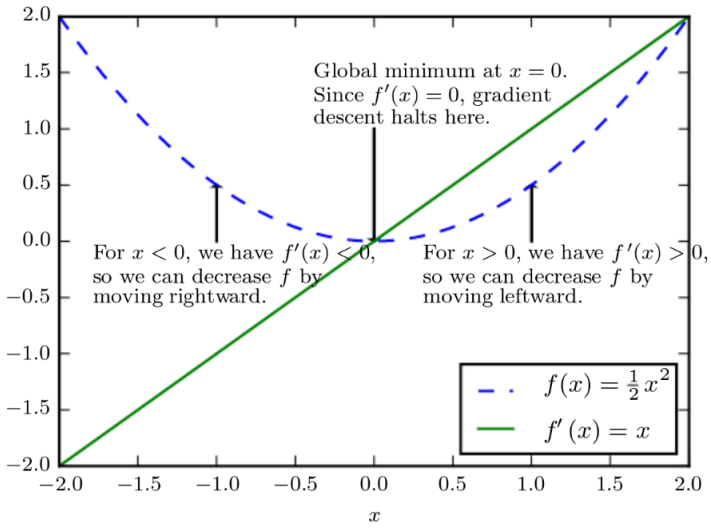
Having now completed a forward pass and computed the respective loss, it is now time to minimize that loss. The backpropagation is an elegant algorithm based on the chain rule of calculus, allowing end-to-end optimization of CNNs.

[5] besides the aforementioned L_{CE}, there is an abundance of loss functions depending on the application itself.
[6] independent and identically distributed.

Figure 2.6: Bottom-up hierarchy of a trained network [25]; for each layer the top nine activations, projected to pixel space, for a random subset of feature maps in the validation set are visualized, along with the the corresponding image patches.

The parameters of our network θ correspond to weights and biases. Adopting a simpler notation and appending the bias to the weights, the parameters are now w. To this goal, gradient descent is applied taking smaller or larger steps along the negative gradient $-\frac{\partial L}{\partial w}$; a trivial example in an 1D scenario is demonstrated in figure 2.7. The elegance of the backpropagation algorithm lies at refraining from recomputing gradients

based on the chain rule, demonstrated with a simple example in (2.6) where the "↦" notation symbolises already computed gradients.



1-d gradient descent

Figure 2.7: 1D gradient descent

$$
\begin{aligned}
&z = wx + b, \qquad y = \sigma(z), \qquad L = \frac{1}{2}(y-t)^2 \\
&\frac{\partial L}{\partial y} = y - t, \quad \frac{\partial L}{\partial z} = \frac{\partial L}{\partial y}\sigma'(z), \quad \frac{\partial L}{\partial w} = \frac{\partial L}{\partial z}x, \quad \frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \\
&\bar{y} = y - t, \quad \bar{z} = \bar{y}\sigma'(z), \quad \bar{w} = \bar{z}x, \quad \bar{b} = \bar{z}
\end{aligned}
\tag{2.6}
$$

This way, starting from the final output of the network, the extracted loss is backpropagated to the entire network through the corresponding gradients. The parameters of the network are updated during training and the computational complexity is restrained by avoiding redundant computations.

2.2.4. Optimization

During the gradient descent, smaller or larger steps are taken along the direction of $-\frac{\partial L}{\partial w}$. The magnitude of these update steps are determined by the learning rate of the gradient descent and the actual update of the parameters by the selected optimizer.

Learning rate

The learning rate controls the magnitude of steps, in other words how drastic we want the changes to be in the network's parameters. Naturally, at an early phase of training (early epochs) we want the network to be able to explore the $\mathbb{R}^d$ loss space, where d the dimensionality of the feature space. As the training progresses we want the gradual reduction of the learning rate with the training optimally converging at the global optimum. Premature and aggressive reduction will possibly get the training stuck at a local minimum.

Tuning the learning rate is an intricate process, considering the computational complexity and the time required for training a DNN. In fact, the burden of hyperparameters tuning is a probable reason to turn against deep learning models.

Optimizers

Given a model with parameters θ and a set of m training examples x_i with labels y_i the loss is computed as in (2.7). $J(\theta)$ is the average loss from all training samples and $\nabla_\theta J(\theta)$ the respective gradients flowing back for the model's parameters update. Given the learning rate η the parameters are updated based on (2.8). Vanilla aka batch gradient descent comes with great computational cost, since it requires the computation of loss and respective gradients corresponding to the entire set of training samples for a single parameters update.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(x_i, y_i, \theta)$$
$$\nabla_\theta J(\theta) = \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} L(x_i, y_i, \theta) \tag{2.7}$$

$$\theta' = \theta - \eta \nabla_\theta J(\theta) \tag{2.8}$$

To cut down computational cost the SGD (Stochastic Gradient Descent), also known as mini-batch gradient descent approximates gradients and performs updates for mini-batches of n training samples. Convergence to the global optimum is certain regarding the batch gradient descent and convex surfaces, while SGD combined with learning rate decay has similar convergence behaviour[7] [17]. The batch size is essentially a trade-off, with a bigger size rendering a better approximation for gradients thus preventing the SGD overshooting. On the other hand, the bigger the batch size the higher the computational cost for the extraction of gradients. Additionally, the batch size affects also the learning rate, with higher batch sizes enabling generally the utilization of higher learning rates and thus the faster convergence.

SGD is undoubtedly the most common optimization algorithm used for training DNNs. There is an abundance of other optimization algorithms, exploiting the first central moments of updates to improve over SGD (reduce oscillations, align updates to the direction of the local/global minima). Momentum and RMSprop are two examples, utilizing first and second central moments respectively, while Adam brings the best of both worlds. We won't go into further details, but instead refer to [17] for a very nice overview. We need to note though, that it is a field of active research with many new algorithms being developed.

Normalization

Normalization of input features is crucial, as in any conventional ML model. Unnormalized features would lead to unequal step lengths in the $\mathbb{R}^d$ loss function space, severely hampering convergence to global or even local minima. For the sake of visualization, consider a 2D feature space and a corresponding $\mathbb{R}^2$ loss space as shown in figure 2.8. Unnormalized features corresponding to the left loss space, lead to unequal contributions of features to the final loss. To prevent this, given a set of m samples each feature x of a d-dimensional feature space, is normalized to mean $\mu = 0$ and variance $\sigma^2 = 1$ as in (2.9). In a complete similar fashion, intermediate features maps are also normalized to prevent the phenomenon of internal shift, with the method of Batch Normalization of Szegedy et. al [9], which we will cover in subsection 2.3.5.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$$
$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i)^2 \tag{2.9}$$
$$X = \frac{X - \mu}{\sqrt{\sigma^2}}$$

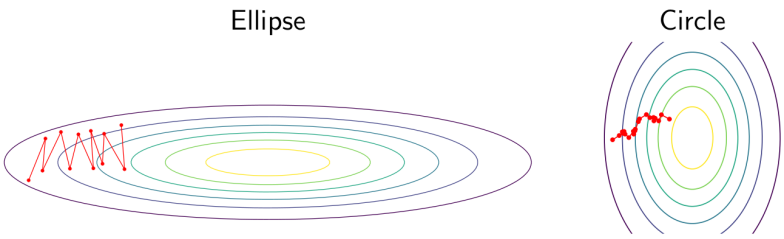


Figure 2.8: A 2D loss space and updates performed by simple SGD, for unnormalized (left) and normalized input (right). Clearly for the unnormalized case, the 2 features have unequal variances, thus leading to unequal contributions to the final loss

[7] local minimum is guaranteed for non-convex surfaces.

2.3. Regularization

Likewise to the context of ML the goal for a Deep Learning model is to generalize well on unseen data. A model doing quite well on the training data, but on the other hand reaching poor classification performance on test data is suffering from overfitting. Considering a learning curve as in figure 2.9, where accuracy is plotted against the size of the training set, the gap between the training and test accuracy is the overfit.

Clearly, the most straight-forward way to reduce overfitting is to train using more data. The availability of finite data especially in the supervised setting, along with the ability to generate models of very high complexity (vast parameters), renders regularization a crucial need when training Machine Learning models in general. The number of model's parameters, the size of the training set and the generalization ability of the model are correlating factors. As shown in the learning curve of figure 2.9, given more data the gap between the apparent and true error, or just the overfit, decreases[8].

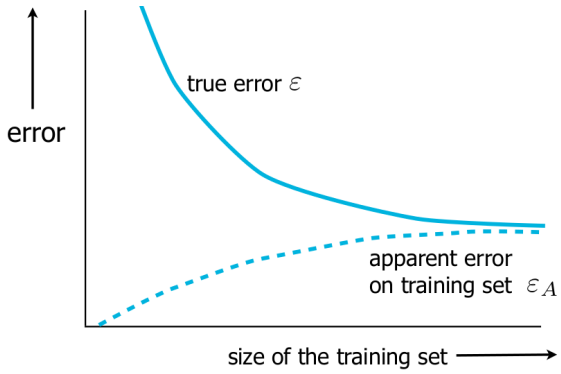


Figure 2.9: A learning curve, plotting apparent and true error, ϵ and ϵ_A respectively versus the size of training dataset for a model of given complexity.

Regarding now the complexity of a model, a simple model has low representational power, or equivalently high bias and low variance. On the contrary, a complex model has high representation capabilities, thus low bias and high variance. Higher complexity though leads to more data required for training successful models, with limited overfitting. The impact of increased complexity on figure 2.9 is that the saturated area on the right part of the graph, with minimal overfitting, captures less and less part of the graph.

Under normal circumstances, acquiring more data is infeasible thus reducing the complexity of the model is the realistic alternative to reduce overfitting. Regularization penalises the complexity of the model, forcing the extraction of a more generalizable representation and there are different techniques to this goal. Let's now examine the main regularization methods.

2.3.1. Early stopping

Quite an efficient and simple solution. A model is trained, while extracting the validation loss at the end of each training epoch. As soon as the the validation loss stops decreasing training stops. After the validation loss plateaus it will start increasing, a sign of overfitting.

2.3.2. Parameter sharing

This technique is aligned towards reducing the number of parameters of a network. We have already introduced weight sharing in section 2.1.1. The same filters are applied for detecting an object, regardless of its exact spatial location in the image. A fully-connected layer instead would lead to explosion of the number of trainable parameters and consequently the generalization capabilities of the network.

2.3.3. Weight decay

Instead of minimizing the loss function in (2.7), we minimize a regularized loss $\tilde{J}(\theta)$ as in (2.10), where $\Omega(\theta)$ the regularizer and α the regularization parameter, in essence the contribution of Ω to the regularized loss. Minimizing $\tilde{J}(\theta)$ leads to the corresponding gradients. Two of the most common options are the L1 and L2 regularizations given in (2.11) and (2.12) respectively, along with the respective gradients.

[8] the error on the training data constitutes the apparent error, whereas the true error corresponds to error on novel, unseen data.

$$\tilde{J}(\theta; x, y) = J(\theta; x, y) + \alpha\Omega(\theta) \tag{2.10}$$

$$\begin{aligned} \tilde{J}(w; x, y) &= \alpha||w||_1 + J(w; x, y) \\ w &\leftarrow wi - \epsilon(\alpha sign(w) - \nabla_w J(w; x, y)) \end{aligned} \tag{2.11}$$

$$\begin{aligned} \tilde{J}(w; x, y) &= \frac{\alpha}{2} w^T w + J(w; x, y) \\ w &\leftarrow w - \epsilon(\alpha w + \nabla_w J(w; x, y)) \end{aligned} \tag{2.12}$$

Both these methods penalize large weights, but there is a subtle difference. As seen in equations (2.11), (2.12) the weight reduction is proportional to the weight for L2 regularization, but fixed for L1. Hence, regularization is faster for L2 and larger weights, and vice versa for L1. Consequently, L1 regularization leads many small weights to zero.

2.3.4. Dropout

Dropout [22] is another technique against overfitting. During each training step "jittering" is applied to the model, by removing a random fraction p of the original nodes and performing standard backpropagation to the remaining nodes. This leads to slightly different variations of the original model to come up during each training step, with the final outcome being the averaged outcome of this realizations. This is quite similar to the bagging technique, for reducing variance of Machine Learning models.

There is a discrepancy in the application of Dropout between training and test time. During training time the aforementioned process is followed. During test time though, no nodes are dropped but the weights are instead scaled by $1 - p$.

Dropout has been the prevalent option for reducing overfitting. However, the following method, namely Batch Normalization, has significantly reduced the need for applying dropout. Finally, the compatibility between Dropout and Batch Normalization is severely questioned in [14].

2.3.5. Batch Normalization

Despite not being a method directly proposed against overfitting, we still place it here due to its unquestionable regularizing effects. Batch Normalization [9] is a method for addressing the phenomenon of Internal Covariate shift, resulting in significant speed-up of the training process of DNNs. The activations of each subsequent layer depend on the parameters of the previous layer, which are tuned during the training of the network. This induced change in the distribution of the activations of a certain layer is introduced as Internal Covariate Shift.

To deal with the Internal Covariate Shift, the activations distribution of each layer is fixed by scaling features to have zero mean μ and unit variance σ^2. Complying with the mini-batch SGD optimization algorithm, mini-batch instead of global statistics are utilized. The learnable parameters of the algorithm γ and β ensure that the applied transform does not tamper with the representation capabilities of the network; the network can undo the applied transform simply by setting $\beta_i = \mu$ and $\gamma_i = \sqrt{\delta + \sigma^2}$. The Batch Normalizing transform of feature x on mini-batch B of m samples is shown in (2.13). There is a debate regarding the position of Batch Normalization, but generally it precedes activation functions and thus the output of convolutional layers are normalized. At test time, the learned parameters γ and β are utilized, along the weighted average for mean μ and variance σ^2.

$$\begin{aligned} \mu_B &\leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma_B^2 &\leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \\ y_i &\leftarrow \gamma\hat{x}_i + \beta \end{aligned} \tag{2.13}$$

The applied normalization and the fixing of internal activations distributions led to multiple beneficial effects to the training of networks. A significant training speed-up was achieved, since networks incorporating Batch Normalization facilitated the use of higher learning rates and thus faster convergence; consequently the learning rate decay scheme is also accelerated. Moreover, due to its regularizing effects[9] it decreased the need for other regularization methods such as weight decay and Dropout [22]; regarding the latter, its compatibility with Batch Normalization is questioned in [14].

2.3.6. Data augmentation

Starting this current section for Regularization, we mentioned that under normal circumstances more training samples, especially labelled ones, are quite hard to get. However, there is an efficient solution; data augmentation performing some jittering to the already available samples, enables the artificial synthesis of additional samples, from the existent training distribution. By employing random transformations such as rotations, shifts and flips, which are also expected in the test set, the training distribution is essentially expanded. This method does not affect the complexity of the model, but rather feeds it with more data.

Augmentation during training time essentially moves us more to the right in the learning curve of figure 2.9. It adds up to the "experience", what the network has already seen and would be able to apply inference on test time. The improvement of generalization abilities utilizing augmentation is questioned, the only sure thing is that it provides more samples to train the model on.

[9] the transform essentially applies some jittering to samples, hence there is no deterministic value corresponding to each sample. Each example is considered now in conjunction with the rest samples comprising the mini-batch[9]; eventually the applied transform depends on the mini-batch itself.

3

Top-Down Networks

In the current chapter, we will give some further information regarding the proposed TD (Top-Down) networks that were skipped in the article. A TD network is extracted by mirroring the feature extraction part of a conventional, BU (Bottom-Up) baseline network. The defining difference between them is that the baseline BU sees and processes input in a fine-to-coarse pathway, whereas the proposed TD moves in the opposite direction. Starting from coarser versions of the input and gradually incorporating higher frequency information, through residual connectivity [7].

TD networks mainly inspired by the way human beings perceive a scene; firstly consider a more holistic representation of it and gradually enrich it with higher information for segmentation and finer objects detection[1]. In fact, this reversed CNN architecture leading to a coarse-to-fine processing in the network, is a novel approach that brings a new perspective to the field. We have thoroughly experimented with our TD networks and demonstrated the applicability of the proposed approach to a variety of existing architectures.

3.1. How to extract a TD

A TD network is essentially a mirror to the baseline BU. Figure 3.1 provides an overview of the extraction of a TD network. The first step is the reversal of the feature extraction part[2] of the baseline BU, thus rendering a pathway from lower to higher spatial dimensions. The input to the TD network is the original input downscaled by a factor 2^s, where s the number of pooling operations[3] in the BU network; anti-alias blurring is performed prior any downscaling operation. Next, any pooling layers (or strided convolutions) are mapped to nearest-neighbour upscaling layers. Finally, to restore high frequency information lost during the initial downscaling, residual connectivity is utilized. However, a founding difference of the proposed networks, are that contrary to existing multi-scale architectures, a higher frequency feature map of given scale is extracted based solely on that scale[4].

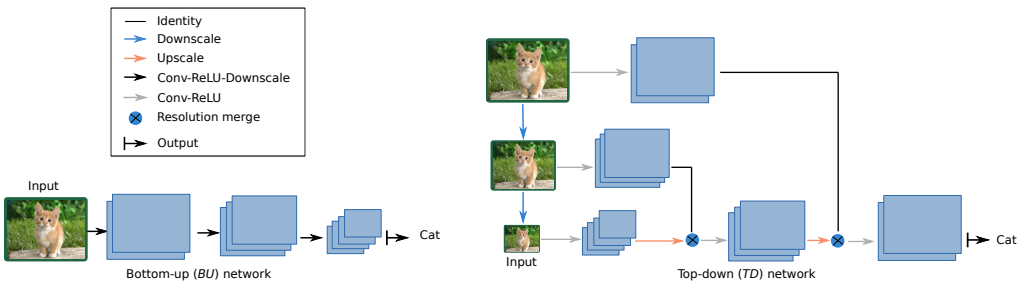
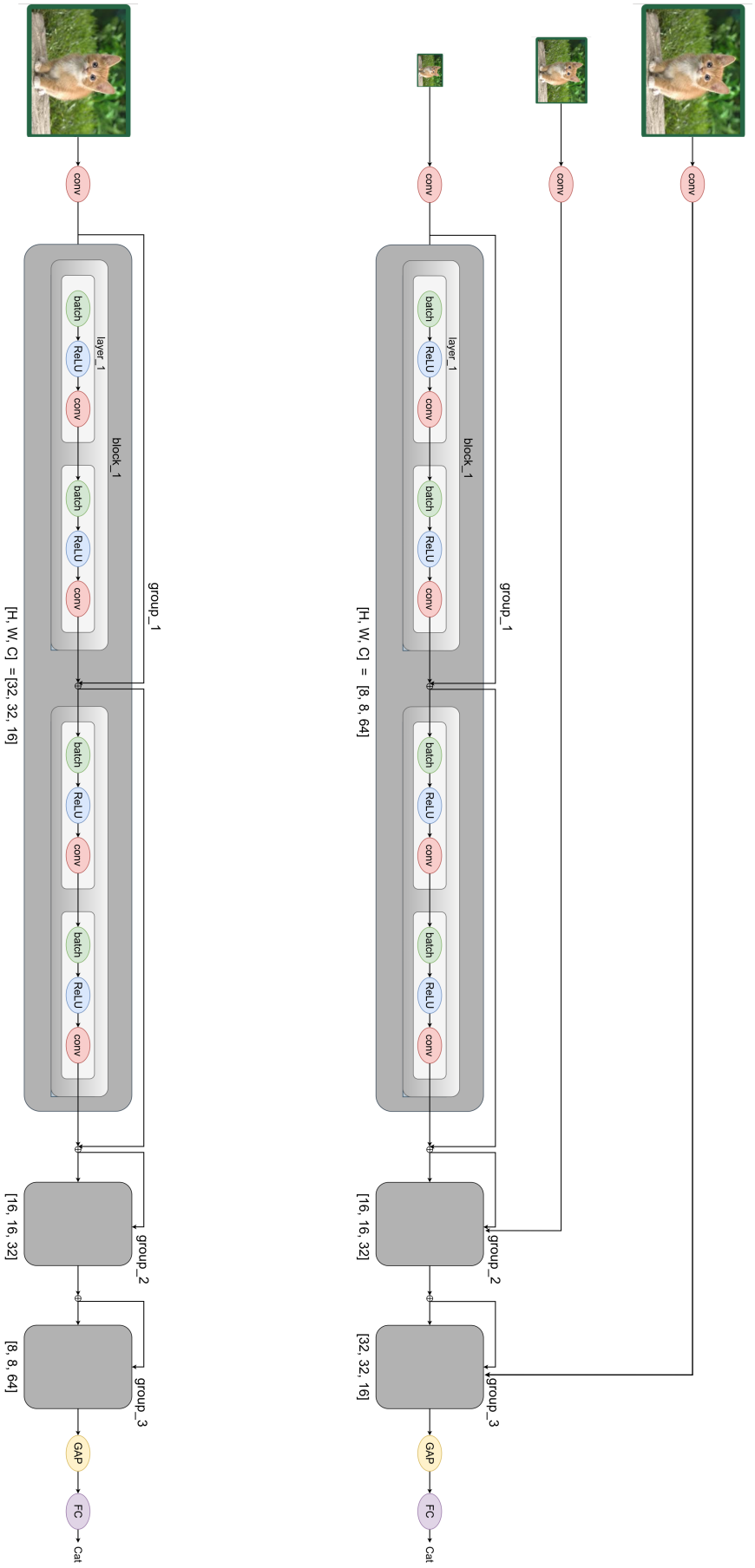


Figure 3.1: Demonstration of the extraction of a TD network based on a baseline BU. On the left, the BU network is visualized, comprised of 3 convolutional layers. On the right, the respective TD is the mirror to the BU, moving from lower to higher frequencies; its initial input scale matches the minimum scale seen by the BU and gradually restoring higher frequency information through residual connectivity.

[1] this claim refers to the conscious process of visual perception; from a biological point of view visual receptors work as a conventional CNN, gradually aggregating finer details extracted from earlier layers.
[2] the layers performing the final classification, e.g. GAP + Softmax or any other conventional classifiers remain fixed.
[3] or any other downscaling operations.
[4] no Bottom-Up pathway.



(a) BU ResNet

(b) TD ResNet

Figure 3.2: ResNet architectures for BU and TD networks. The feature extraction part of the BU network is rotated; groups are rotated as a unit.

3.1.1. TD ResNet

To shed some further light to the process we will next provide a concrete example of extracting the TD network corresponding to the ResNet architecture [7]. We are considering the ResNet34 architecture for the CIFAR10 dataset introduced in [7]. The network architecture comprises of an initial convolutional layer for depth initialization, followed by 3 groups of convolutional blocks; the output of the final group's block is introduced to a GAP layer, with a fully-connected layer performing the final classification. Residual connectivity as introduced in [7] is used, refeeding the output of a each block to the consecutive one.

Now for the extraction of the respective TD, we first have to reverse the feature extraction part, which corresponds to the three groups of the network. These groups are flipped as a whole, with the last group now corresponding to the initial one of the TD. The filters of the initial convolutional layer are also adapted to match the dimensionality of the first group. The layers corresponding to the final classification remain intact, with the final output of the ultimate group being again introduced to a GAP layer and then to a fully-connected one. The output of each block is fed again to the next block. In addition, higher frequency feature maps are introduced to the first block of each group, where spatial dimensions rise. For the merging, any of the three methods introduced in the Method section 3 of the article may be used.

Finally, the input to the TD corresponds to the input of the BU downscaled by a factor of four[5]. For the extraction of the downscaled variants of the input we anti-aliased the input prior the downsampling. Resulting BU and TD architectures are shown in figure 3.2; for the sake of space merely 2 block in the first group are visualized, adding more though would be trivial.

3.2. More experiments

A long path was covered till the final proposal of the TD networks, with numerous experiments-efforts that were not eventually incorporated to the final article. We will concisely present some main efforts that are still worth-mentioning here.

3.2.1. Downscale experiment

It all started here, chronologically as well as from an experimental point of view. Since, the TD networks would start processing from lower resolution input and only later add higher frequency, we wanted to examine how conventional BU performed in the setting of the blurred input. This experiment aimed at providing some baselines regarding the performance we could expect on behalf of the TD, trying to answer the question "What is the impact of suppressing higher frequency information in classification performance?". The setup was identical to the respective one of the first experiment "BUvsTD" in the Experiments section 4 of the article.

In order to extract valuable insight regarding the effect of scale we extracted multiple variants of the original input blurred with $[\sigma = 0.6, 2\sigma, 4\sigma, ...]$; figure 3.3 provides an example of the extraction of blurred variants of an input image from the CIFAR10 dataset. To assess the impact of losing higher frequency, as well as the generalization abilities of the networks against this resolution change we trained and tested on all cases. Corresponding results for matching training and test scales are presented in figure 3.4. Inspecting the figure, we may safely conclude that the loss in classification performance is only minimal, growing higher for increased σ. The loss is higher for Fashion-MNIST and especially CIFAR10, where the dataset contains information in the high frequency region and blurring suppresses this information. For the ResNet a roughly 10% loss from a baseline accuracy of 87.7% is measured; the respective figures for the NIN are 15% and 89.4%.

Next, figure 3.5 presents the complete set of results for the NIN architecture and all the datasets[6]. Clearly, generalization to a different scale is a quite challenging task, with the version trained on blurred with σ input generalizing better as suggested in [10]. The blurring, if not excessive, removes mostly noise thus proving to be helpful to the classification task. Additionally, as expected performance scores are not symmetrical, in essence classification performance is lower when trained on low and tested on higher scale, than vice versa. However, we would expect this gap to be even higher as for the case of lower testing than training scale, all required information, in fact more, had been given to the network for the completion of the task. The other way around, this doesn't hold with the network needed to extrapolate to unknown higher frequency information.

[5] two downscaling operations present in the baseline BU.
[6] results from other architectures tell the same story.

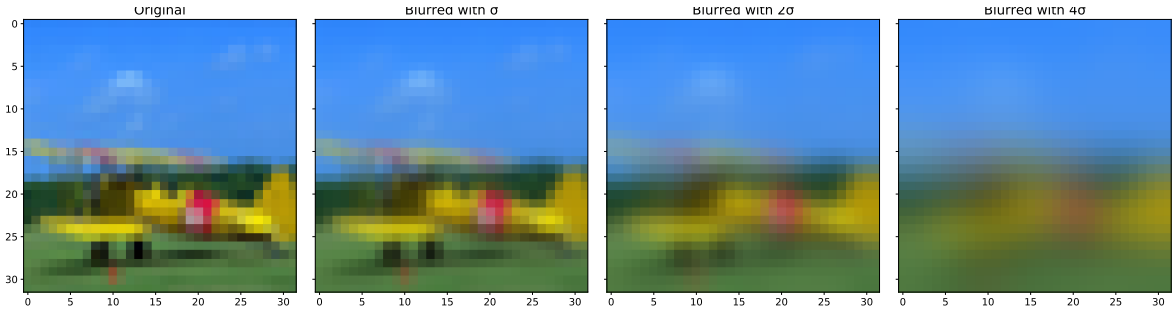


Figure 3.3: Original scale image from CIFAR10 dataset, along with blurred variants with $\{\sigma = 0.6, 2\sigma, 4\sigma\}$; the image corresponds to label "airplane".

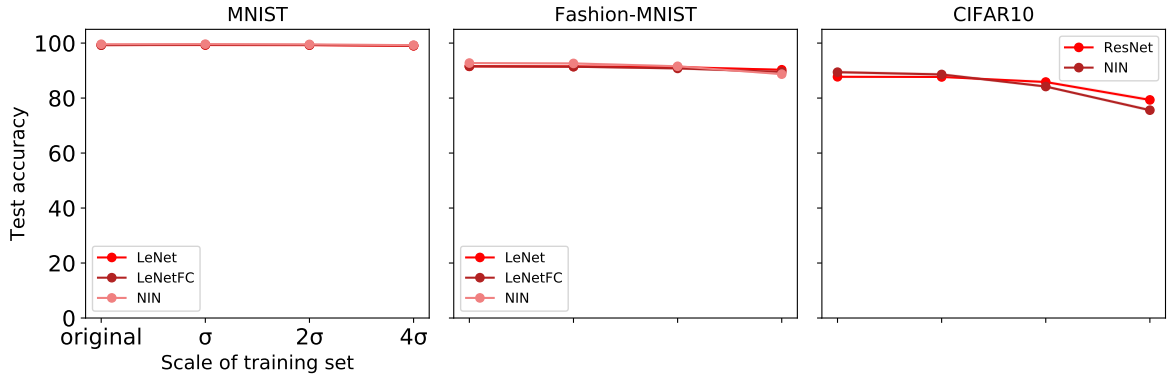


Figure 3.4: Test accuracy on similar scale test set, for all datasets and networks utilized. There is no substantial loss in classification accuracy, meaning that the networks are still able to carry out the classification task, even after the loss of higher frequency. As expected, the impact of the loss rises as we move to the more challenging tasks.

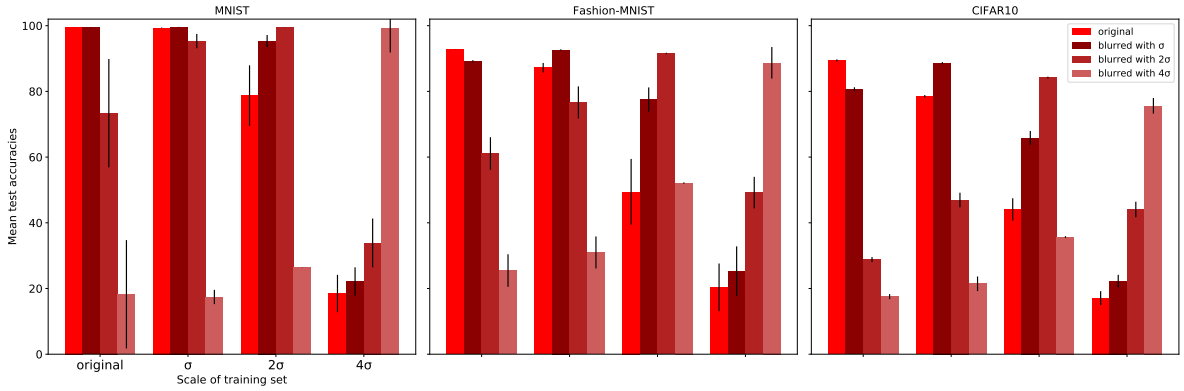


Figure 3.5: Test accuracies for the Network-in-Network architecture, for the three datasets when trained and tested on all scales. The x-axis corresponds to the training scale, whereas each bar colour corresponds to a single test scale. Once more it is clear that when training and test scale matches, there is no substantial loss in performance. When the scale changes though classification becomes harder. Trying to classify samples of higher scale than the traning scale is harder, as expected.

Having extracted some baselines, we then repeated the experiment on the corresponding TD networks. Similar conclusions where drawn from this, with the networks exhibiting robust performance against high frequency loss. In fact, we observed enhanced performance comparing to the BU baselines, when there was a mismatch between training and testing scale. The TD networks were able to perform better in face of this distributional shift and this could be attributed to the network architecture, with the TD network "seeing" multiple scales of the input during training, thus being more resilient to the shift. This was definitely an interesting and encouraging result for the TD endeavours to come.

3.2.2. Toy dataset

Trying to extract some further insight regarding perceptual differences between BU and TD and the input frequency, we extracted a toy dataset with images containing a single frequency, as a 2D-cosine wave as shown in figure 3.6. The dataset contained classes 0-4 with corresponding frequencies in {0,1,2,4,8} Hz, incorporating rotations in $(0, 2\pi; 10)$[7], phases in $[0, 2\pi; 10)$ and amplitude transformations in $(0, 1; 10]$. After the extraction of the dataset we followed a similar path with the downscale experiment in section 3.2.1, extracting downscaled variants. Again networks were trained and tested on all versions of the input.

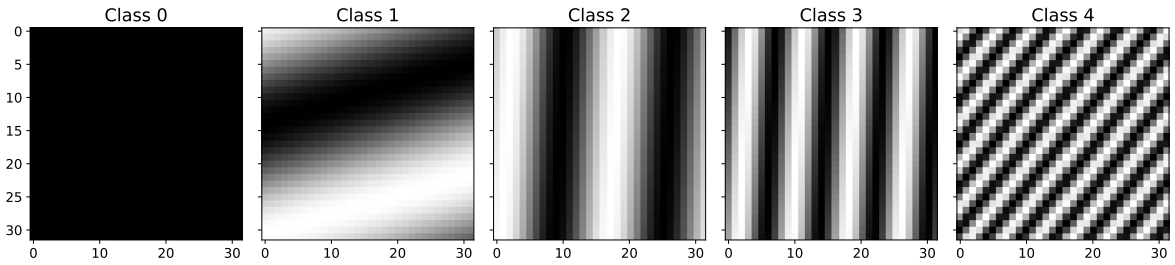


Figure 3.6: Toy dataset; class indices 0-4 map to frequencies {0,1,2,4,8} Hz. Transformations in amplitude, phase and rotations are incorporated.

This experiment was mostly a debug for how the networks react on suppressed higher frequency. Unfortunately, it didn't reveal any major differences between the networks, with the predictions gradually moving to the DC class with more blurring applied.

3.2.3. Fourier domain filtering

This was another experiment investigating the effect of scale of the input. Based on initial input x and applying filtering in the Fourier domain, we extracted variants x_{low} and x_{high} as shown in (3.1) and figure 3.7. This filtering corresponds in essence to blurring for x_{low}, while x_{high} is the complementary set in the Fourier domain. We then trained on both x_{low} and x_{high} and tested on the original test set.

$$\begin{aligned} x_{low} &= IFT\{FT\{x\} * m\}, \quad m(u,v) = 1 \; for \; u^2 + v^2 \leq 1, \; 0 \; otherwise \\ x_{high} &= IFT\{FT\{x\} * (1-m)\} \end{aligned} \tag{3.1}$$

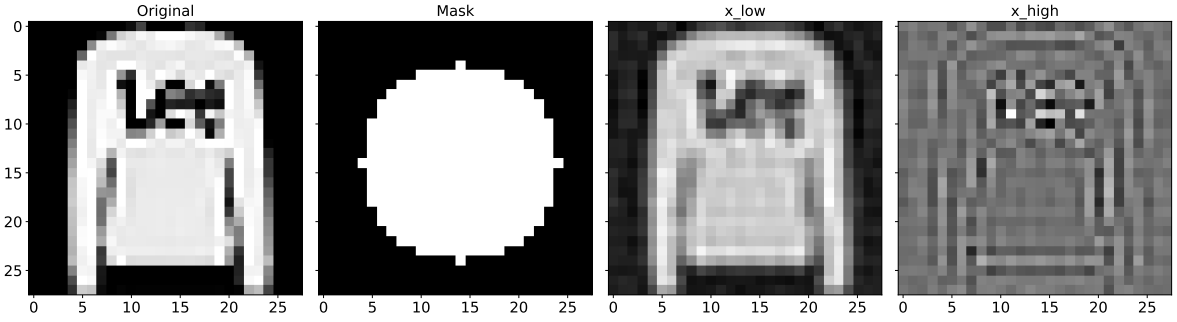


Figure 3.7: From left to right: original image x_i, mask for the filtering and $x_{low,i}$, $x_{high,i}$

The crash test between the BU and TD revealed no substantial difference when the networks where trained on the blurred variant x_{low}. In fact, the networks were able to perform almost as good as when trained

[7] the notation $(0, 2\pi; 10)$ symbolizes 10 transformations equidistant in range $(0, 2\pi)$.

on the initial training set. This can be justified as the blurring (if not excess) suppresses mainly noise information which can hamper classification performance. Additionally, in the tested datasets and especially MNIST and Fashion-MNIST, a more holistic representation of the input is usually sufficient for successful classification[8].

On the contrary, when trained on the x_{high} both networks reported deteriorated performance. The loss was higher regarding the TD, revealing a difficulty of the networks to exploit high frequency information. This led us to increase the number of parameters of the TD networks, while using more advanced approaches for merging than simple element-wise addition[9].

Furthermore, this Fourier filtering gave us another idea for a quite different approach to the TD networks. The coarse-to-fine processing pathway finally adopted, moves from the minimum scale extracted from the initial input downscaled by a factor of 2^s, where s the pooling operations in the initial BU, to the initial spatial resolution. Instead, we tested with extracting equal number of non-overlapping concentric regions in the Fourier domain as shown in figure 3.8, with the inner-most circle corresponding to the blurred version of the initial input; clearly, this is equivalent to Laplacian filtering[10]. This was motivated by the redundancy encountered in the original approach; there is significant overlap in the low-frequency region between the downsampled variants, hence concatenation of low and high resolution feature maps builds up this recurring region. The modified approach with the Fourier filtering vanishes this redundancy, feeding information corresponding to separate frequency regions at each stage. This could potentially facilitate learning, by allowing the network to deal with each frequency region separately. However, slightly lower performance was measured indicating that this redundancy is actually helpful to the network's training. The idea is still quite interesting though and will surely return in any future ventures.

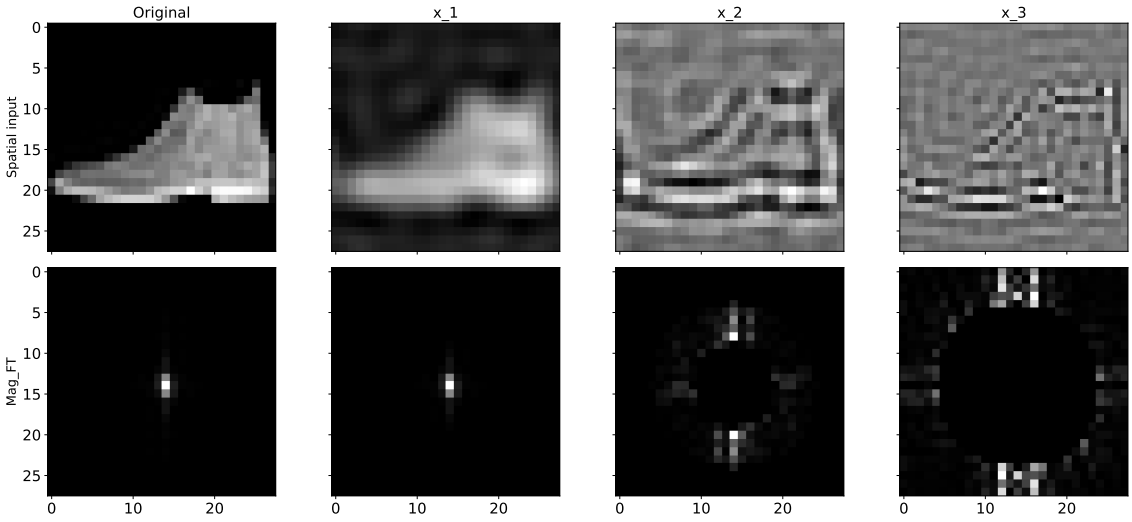


Figure 3.8: Filtering for extracting non-overalapping regions in the frequency domain. On the top row, the original and the images corresponding to the concentric areas are given from inner to outermost. On the bottom, the respective magnitudes of the Discrete Fourier Transform are given. The extracted regions correspond to radiuses $r = [5, 10]$.

3.2.4. Progressive growing

Merging of high and low resolution pathways is a common concept mainly in segmentation networks (please refer to the related work of the article). The Generative Adversarial Network (GAN) of [11] also adopted this merging, with an interesting though adaptation. Each time higher frequency was introduced, the network's capacity was increased by adding extra layers to be able to handle the finer information. Sudden introduction of new layer's would "shock" the already trained previous layers, hence the progressive growing is proposed in [11]. The training then incorporates two phases, namely the staging in and the fine-tuning. After each expansion, the network is retrained while all the weights remain trainable. This way the network can adapt to the increased capacity, whereas the second stage clearly fine-tunes the weights of the updated model.

[8] a blurred shirt can be easily discriminated from a blurred shoe.

[9] a combination of element-wise addition and concatenation on the channel dimension was finally adopted; please refer to the method section of the article.

[10] difference of Gaussians.

The main motivation behind progressive growing is to facilitate the learning process, with the network experiencing and "learn" each scale of the input progressively, in a "divide-and-conquer" fashion. We did adopt progressive growing for our TD networks, without a substantial impact in final classification performance. We also tested with transfer learning, in essence the predecessor of the progressive growing, where after each expansion the network was again retrained but with previous layers' weights frozen this time. This led to significant performance deterioration.

3.3. Gradcam

Despite not being a component itself of TD networks, Gradcam [19] has been extensively used in the current graduation project. Gradcam based on CAM [28] contributes to the transparency and understandability of CNNs, by extracting localization maps of the most informative features for the classification task. To this goal, the feature maps A from the final convolutional layer is utilized. The class-discriminative gradient is computed based on the output y^c as $\frac{\partial y^c}{\partial A^k_{ij}}$ for feature map k. Then, these backpropagating gradients are global-average-pooled as in (3.2). The weights α^c captures the contribution for the target class c. The final heatmap is a weighted sum of feature maps A, followed by a ReLU as in (3.3). Higher frequency visualizations are extracted using Deconvolution and Guided Backpropagation; for further information on those please refer to [19].

$$\alpha^c_K = \frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A^k_{ij}} \tag{3.2}$$

$$L^c_{Grad-CAM} = ReLu(\sum_k \alpha^c_k A^k) \tag{3.3}$$

In figure 3.9 we share a very nice visualization of [28], where class-discriminative localization maps corresponding to VGG16 are visualized. The transition from more local, edge-detecting filters, to more global ones representing abstract patters is clearly demonstrated. We advise the reader to compare this visualization with the respective ones given in the Discussion section 5 of the article for the TD networks. The impact of the adopted coarse-to-fine processing paradigm affects these localization maps, with later layers corresponding to higher frequency maps. For more visualization please refer to the Discussion section of the article and of course to [28].

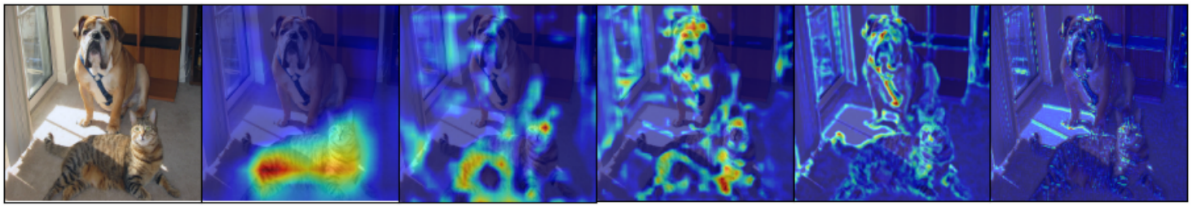


Figure 3.9: Class-discriminative localization maps for VGG16. Firstly the original image is given and then heatmaps from the final to earlier convolutional layers as we move to the right are plotted [28]. The transition from local, edge-detecting filters, to more global representing abstract patters is clearly demonstrated.

4

Adversarial attacks

Ever since the seminal work of [6], adversarial attacks have been an active field of research in the Computer Vision community. In the context of adversarial attacks, an adversary attacking a trained model, tries to find the minimum required perturbations to fool a model. Interestingly, these perturbations are of small magnitude, imperceptible to the human eye and lead a well-generalizing model to misclassify the input with high confidence. Another interesting aspect is the transferability of those attacks, which remain agnostic to the network topology.

A taxonomy based on the access of information regarding the model and its training process is:

- white-box attacks: the adversary has full access to the model and its parameters θ, as well as to the training algorithm utilized. Clearly, this category leads to more powerful attacks.
- black-box attacks: require no internal knowledge of the model and/or the training algorithm, but simply for the final output of the model. These attacks can be further categorized to,
  - decision-based: the final output label is sufficient for the adversary. Perturbations are then generated till the input is misclassified.
  - score-based: output class scores are required by the adversary.

For our project, the main emphasis was laid on black-box attacks as they do correspond to more realistic scenarios (e.g. autonomous self-driving cars). Before proceeding to a nice overview of some infamous attacks, let us give some useful definitions applying in the context of adversarial attacks.

4.1. Definitions

- adversary: the attacker
- adversarial perturbation: the attack in form of noise applied to the input by the attacker, for leading the model to misclassification,
- adversarial example/image: the adversarially perturbed input,
- adversarial distance: the distance $d(x, x_{adv})$ between the original and the adversarially perturbed input; in our case d corresponds to the L2 distance,
- attack success-ratio: the ratio of adversarially perturbed images, successfully leading the attacked model to misclassification.

4.2. White-box attacks

Clearly, transparency to the internal architecture of the model and its training algorithm enables the attacker to devise efficient, well-tailored perturbations. The network gradients constitute the main tool used for devising the attacks of this category. As mentioned in section 2.2, training a neural network is equivalent to minimizing a loss function. Instead of moving along $-\nabla_\theta J(\theta)$ (2.7), one can simply move on the opposite

direction thus maximizing the network's loss. The effectiveness of such an attack, even of quite small magnitude is demonstrated in figure 4.1.

Some characteristic examples of white-box attacks are, the Fast Gradient Sign Method (FGSM)[23], the DeepFool propose in [16] by Moosavi-Dezfooli et al. and Carlili and Wagner (C&W) attacks.

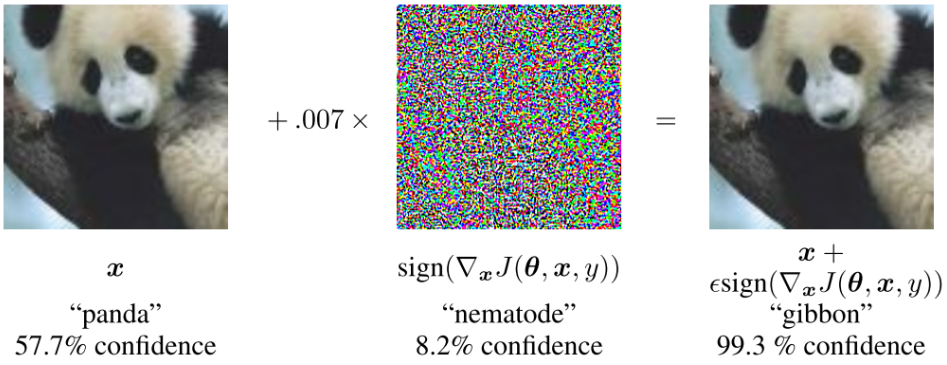


Figure 4.1: Even a perturbation of small magnitude in the direction of the loss gradient, can lead GoogleNet [24] trained on Imagenet to misclassification [6]. Notice the imperceptibility of the perturbation and the high confidence with which the model misclassifies now the input.

4.3. Black-box attacks

In the black-box scenario the attacker has no access to the model, or its training algorithm. Deprived of gradients knowledge, the adversary can no longer apply perturbations maximizing the loss function. Numerical methods are applied though trying to approximate the gradients. Despite generally being less powerful, they can lead to quite universal perturbations, sometimes following a quite simple approach. Additionally, black-box attacks are robust against a quite common defence, namely the gradient obfuscation.

We will next concisely present the attacks considered for the current project, as well as provide some characteristic examples for each type of attack. The attacks are grouped based on the type of noise, or type of transformations applied to the image to correlated noise, uncorrelated noise, blurring and spatial transformation attacks.

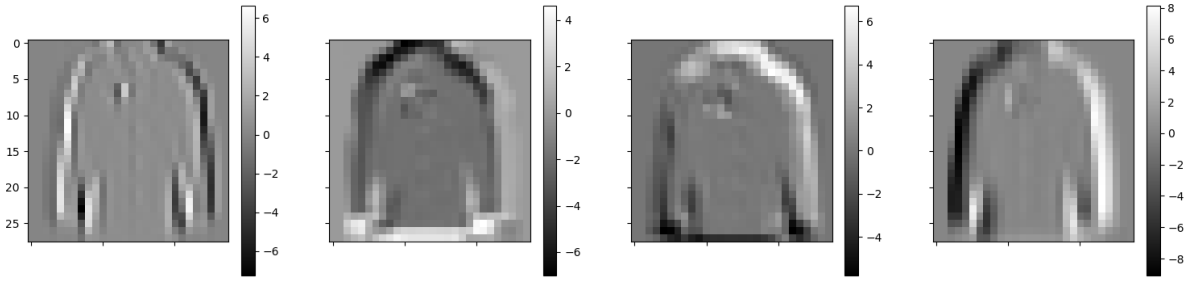
4.3.1. Correlated noise

Single-Pixel attack

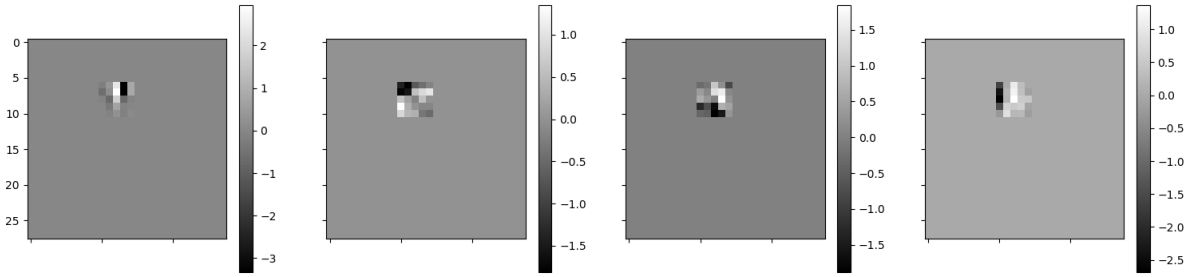
As the name already suggests, the adversary simply by modifying a single pixel, manages to fool the model. This is a particularly interesting type of attack; introducing 0s/1s in smooth areas of 1s/0s of the input image, pollutes the input with imprints of the learned filters[1], thus introducing correlated noise. A demonstration of the introduced pollution is provided in figure 4.2, whereas figure 4.3 provides examples of adversarial images generated by this type of attack.

The coarse-to-fine processing pathway adopted by our TD networks effectively deals with this type of attack, as the blurred and downscaled input significantly reduces the introduced noise; uncorrelated noise would be completely eradicated. On the contrary, regarding BU networks the polluted input gets propagated all the way down the network, spreading out the introduced pollution.

[1] the perturbed pixel is essentially a delta function and convolving a function with a delta returns the function; in case of imperfect delta blurred out version of the filters are returned instead.



(a) Output corresponding to four masks of the first convolutional layer filter of LeNetFC to the perturbed by the Single-Pixel attack input



(b) Difference in outputs between clean and perturbed input.

Figure 4.2: Demonstration of the polluting effects of Single-Pixel attack. The perturbed pixel essentially operates as a delta function and based on the convolutional nature of the network, imprints of the learned filters (correlated noise) are introduced. The introduced pollution "spreads out" covering bigger region of the input as we move to deeper layers and hence more convolutions are applied.

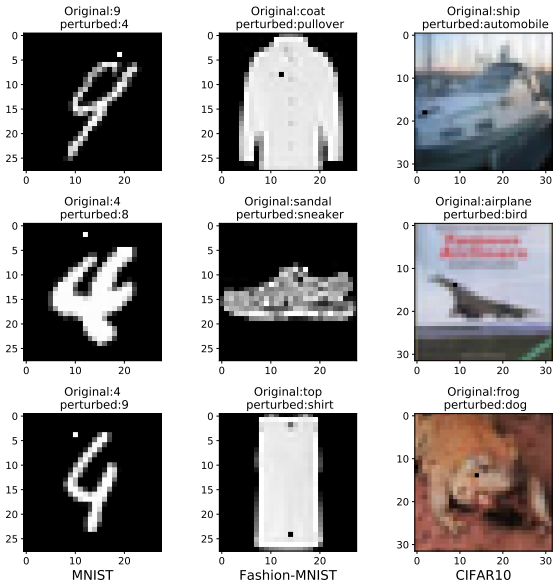
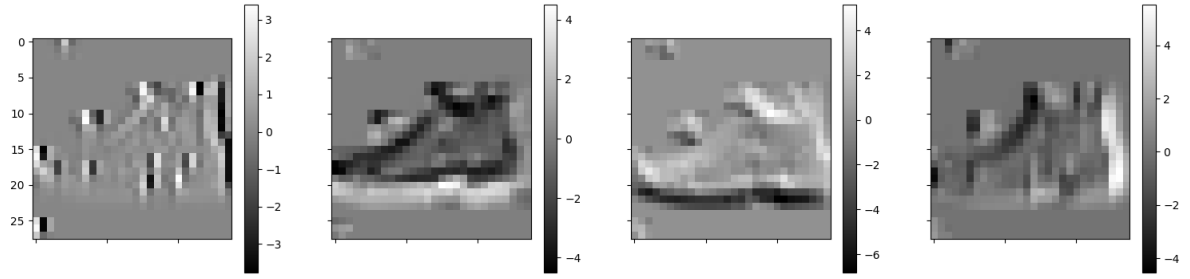


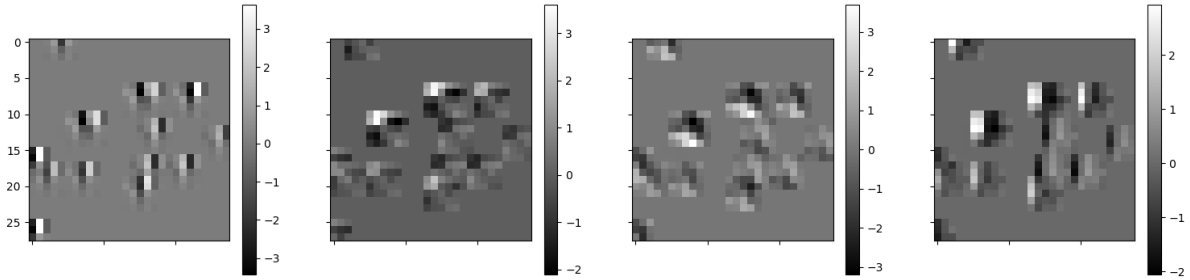
Figure 4.3: Adversarial images generated from Single-Pixel attack. On the top of each image the original and the perturbed label is indicated.

Salt&Pepper

This is clearly an extension of the Single-Pixel attack as shown in figure 4.4. Some characteristic examples of perturbed images are provided in figure 4.5. The perturbed pixels will result in learned filters imprint (or blurred out variants), which will expand in bigger regions of the image as we move in deeper layers and more convolutions are applied.



(a) Output corresponding to four masks of the first convolutional layer filter of LeNetFC to the perturbed by the Salt&Pepper attack input



(b) Difference in outputs between clean and perturbed input.

Figure 4.4: Clearly the Salt&Pepper is an extension to the Single-Pixel attack, with multiple pixels perturbed and thus resulting in multiple imprints.

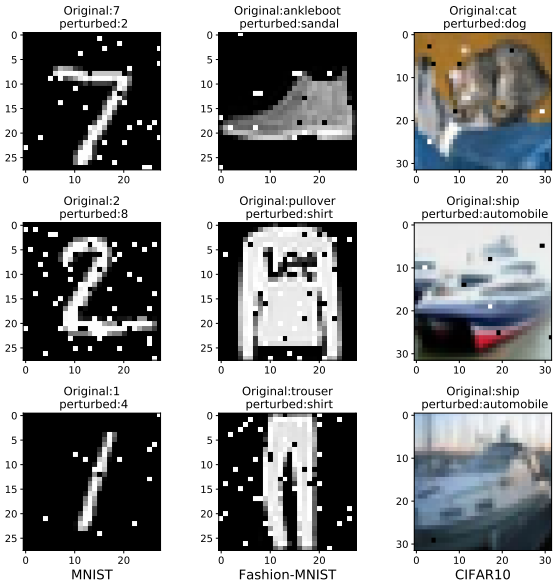


Figure 4.5: Adversarial images generated from Salt&Pepper attack. On the top of each image the original and the perturbed label is indicated.

Pointwise attack

Pointwise attack [18] starts perturbing the input, by adding Salt&Pepper noise until the input is misclassified. Then tries to find the minimal required perturbation, revisiting and resetting part of the perturbed pixels so long as the input remains adversarial. Resemblance to the Salt&Pepper attack is obvious (figure 4.6) and the impact of the attack introducing imprints of the filters is the same.

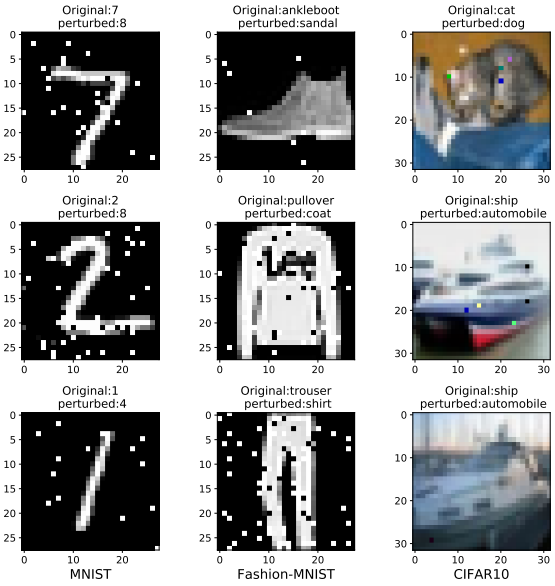


Figure 4.6: Adversarial images generated from Pointwise attack [18]. On the top of each image the original and the perturbed label is indicated.

4.3.2. Uncorrelated noise

Additive Noise attack

For this type of attack, additive noise is introduced in the original input. Uniform or gaussian noise is added, gradually increasing the standard deviation until the input is misclassified. Examples are provided in figure 4.7 and 4.8 for the gaussian and uniform noise respectively. The coarse-to-fine pathway adopted in our TD networks and the original blurring deals effectively with this uncorrelated noise introduced, thus justifying the increased robustness of our TD models compared to the respective BU baselines.

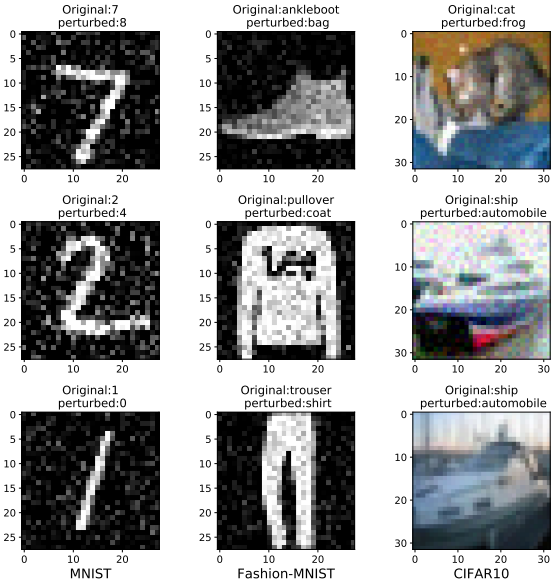


Figure 4.7: Adversarial images generated from Additive Gaussian attack.

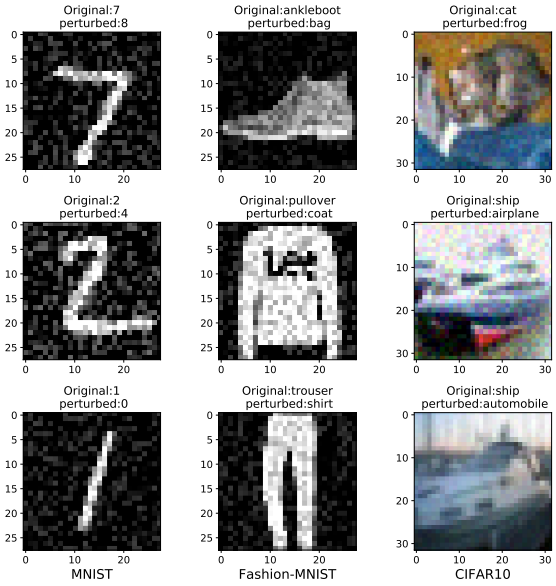


Figure 4.8: Adversarial images generated from Additive Uniform attack.

Blended Uniform Noise attack

Similar to the previous category, but in this case the noise is not added to the input but rather blended in the input as in (4.1), where $\epsilon \in [0,1]$ and n uniform noise in $[x_{min}, x_{max}]$. Yielded adversarial images are visualised in figure 4.9. Similar to the additive Gaussian/uniform noise, the noise introduced here is uncorrelated as well and the TD networks is able to handle it quite well.

$$x_{adv} = \epsilon * n + (1 - \epsilon) * x \tag{4.1}$$

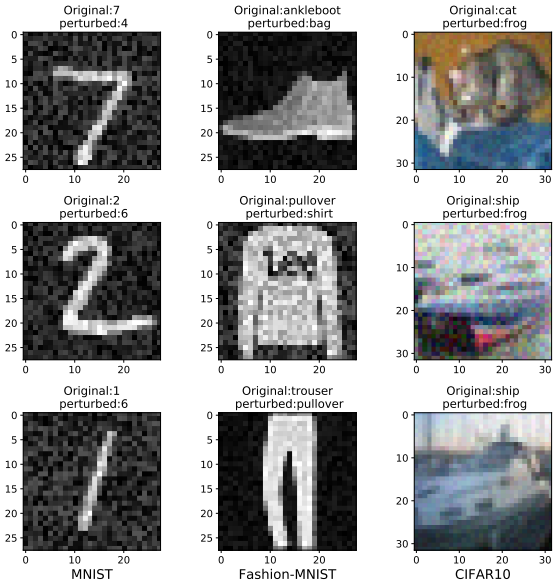


Figure 4.9: Adversarial images generated from the Blended Uniform noise attack.

4.3.3. Blurring attacks

Gaussian blur attack

A very interesting type of attack, where the input is blurred with a gaussian kernel of increasing $\sigma \in (0, max(H,W)]$, where H, W the height and width of the input respectively. The attack smooths out the image as shown in fig-

ure 4.10. The gaussian blurring baked in the TD architecture, with the network utilizing multiple scales during training leads to increased robustness to this attack.

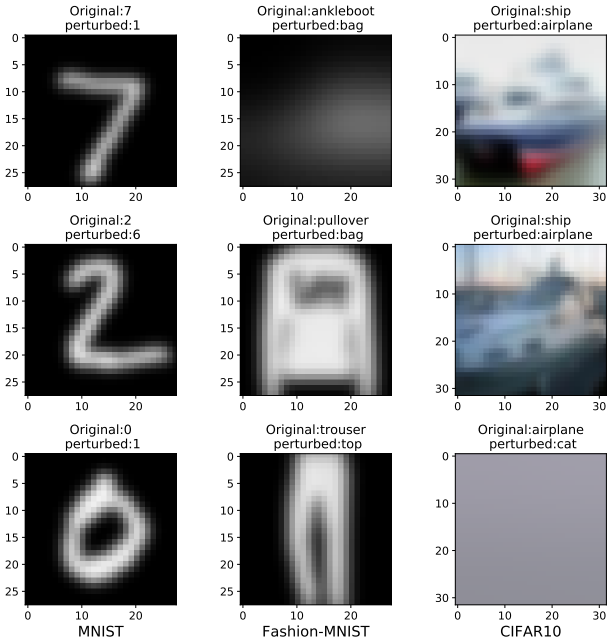


Figure 4.10: Adversarial images generated from the Gaussian blur attack.

Contrast Reduction attack

The Contrast Reduction attack decreases the input contrast until misclassification is achieved. For given bounds $[0,1]$ of input images, the contrast reduction is performed towards the mean 0.5. Thus, the set of values of images are driven towards 0.5 with some resulting perturbed images presented in figure 4.11.

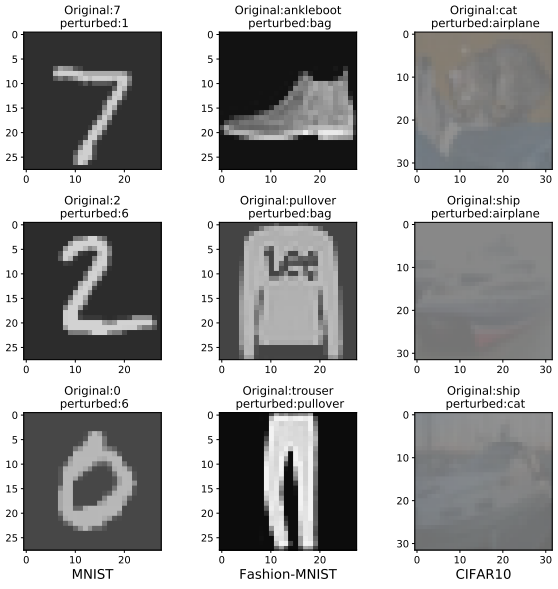


Figure 4.11: Adversarial images generated from Contrast Reduction attack. On the top of each image the original and the perturbed label is indicated.

4.3.4. Spatial transforms attacks

The Spatial Attack [4] perturbs input, by performing geometrical transformations in our case rotations and translations to it. Based on the findings of [26] regarding the beneficial effects of proper anti-aliased down-

sampling, to the preservation of CNN's shift invariance, we expected enhanced robustness of TD networks. To test this we also applied a variant of the attack considering only spatial shifts.

Inspecting the perturbed images though of figures 4.12, 4.13 the applied perturbation further distorts the image by introducing regions of 0s as a results of the applied spatial transform. This is effectively masquerader in the grayscale MNIST and Fashion-MNIST datasets and the applied attack boils down to a spatial transform.

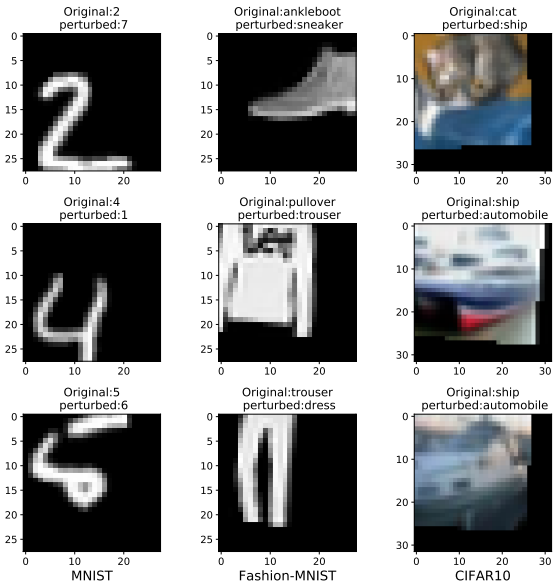


Figure 4.12: Adversarial images generated from Spatial Attack [4]. On the top of each image the original and the perturbed label is indicated.

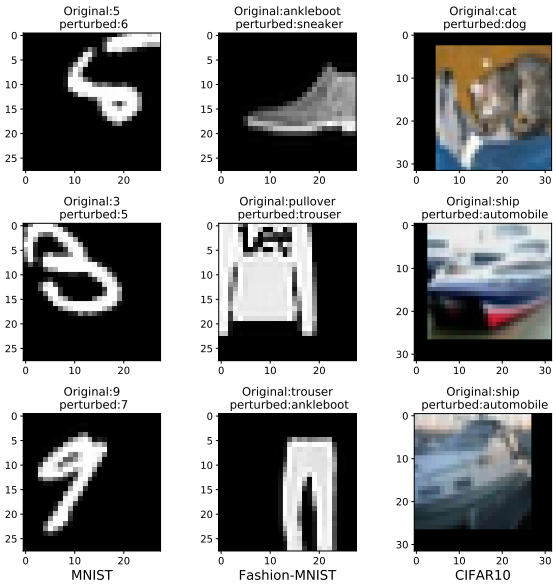


Figure 4.13: Adversarial images generated from Spatial attack [4] variant performing only translations. On the top of each image the original and the perturbed label is indicated.

4.4. Defences against adversarial attacks

There are various methods for defencing against adversarial attacks. The most common one is probably adversarial training, which simply augments training data with similar perturbations, thus actually training the

network to recognize these perturbations. Clearly, this method fails when the actual attack applied has not been seen during training. Other methods are gradients obfuscation, defensive distillation, feature compression; for further information please refer to [3] and [1].

Defence methods, such as adversarial training, can enhance robustness of any network. However, in the current project we focused on inherent robustness, investigating architectural components which could yield enhanced robustness against certain types of attacks (e.g. blurred downsampling and coarse-to-fine pathway of TD networks versus the uncorrelated noise attacks).

Bibliography

[1] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 2018.

[2] Kunlun Bai. A comprehensive introduction to different types of convolutions in deep learning. URL `https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215`.

[3] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

[4] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. *arXiv preprint arXiv:1712.02779*, 2017.

[5] J.v. Gemert. Lectures on deep learning, 2018.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Jay Hegdé. Time course of visual perception: coarse-to-fine processing and beyond. *Progress in neurobiology*, 2008.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[10] Jason Jo and Yoshua Bengio. Measuring the tendency of cnns to learn surface statistical regularities. *arXiv preprint arXiv:1711.11561*, 2017.

[11] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

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

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[14] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2682–2690, 2019.

[15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[17] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[18] Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on mnist. *arXiv preprint arXiv:1805.09190*, 2018.

[19] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International conference on Computer Iision*, 2017.

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

[21] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014.

[23] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[25] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[26] Richard Zhang. Making convolutional networks shift-invariant again. *arXiv preprint arXiv:1904.11486*, 2019.

[27] Xilin Zhang, Li Zhaoping, Tiangang Zhou, and Fang Fang. Neural Activities in V1 Create a Bottom-Up Saliency Map. *Neuron*, 2012.

[28] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.