Fovea-inspired Convolutions: Larger Receptive Fields for Parameter-Efficient and Robust Image Classification

Anupkrishnan, Sidharth
University of Massachusetts Amherst
140 Governors Dr, Amherst, MA 01002, USA
sanupkrishna@umass.edu

Gavier, Ignacio
University of Massachusetts Amherst
140 Governors Dr, Amherst, MA 01002, USA
igavier@umass.edu

Abstract

Large receptive fields in the form of convolutional filters incur computational and space complexity penalties when scaled with the network depth. Additionally, there is a possibility of overfitting due to the larger parameter sizes. Inspired by the retina, we introduce foveated convolution kernel - a multi dilated convolution kernel with a dense center and sparse peripheries, towards parameter efficient learning through bigger receptive fields. We also evaluate its adversarial robustness and its ability to integrate non-local features for alleviating texture bias in CNNs.

1. Introduction

Current Convolutional Neural Network (CNN) models are extremely reliant on local features (called texture bias in recent literature [17]) and layers do not make use of global relationships between them. Increasing the receptive field is computationally expensive and leads to overfitting. However, Vision Transformers have overcome this problem as they capture global and local features, but they need attention mechanisms that are computationally expensive. Furthermore, despite decent test scores on benchmarks, CNNs and Transformers are extremely vulnerable to adversarial examples, and as a result are not quite robust.

The human eye, on the other hand, is much more robust than those models and is not biased by local features. The structure and distribution of the cones in the retina is peculiar in that they are distributed radially with a high concentration of them in the central area known as the fovea and a sparse distribution around the fovea (in the peripheries) [5]. In this project, we draw inspiration from such a biological design and propose to introduce and experiment with foveated convolution modules in visual systems. We draw inspiration from the retina: high acuity vision in the fovea and fast, sparse low-resolution aggregation in the periphery [5], leading to convergence of both local and non-local features, to model a specialized type of filters.

The contributions of this work to the current literature are:

1. More global context per foveated convolution - similarities to vision transformer which encodes more global attention information starting from the early layers. We believe this would improve the accuracy of image recognition tasks.
2. The foveated filters are dense only in the center and sparse on the periphery. As a whole the entire filter, though bigger in the window size, still does not incur a blowup in the number of parameters.
3. The randomness introduced in the periphery improves robustness of the model towards adversarial perturbations.

2. Background

After the success of LeNet-5 [10], Convolutional Neural Networks (CNNs) have been ubiquitous in vision tasks in the deep-learning field. This network architecture has been successful in image classification because of its simple implementation and its ability to efficiently learn local feature relations in the image. These relations provide a large amount of scenic information in real-world images, which is sufficient for a NN to perform relatively well in image classification. However, real-world images may present relations between features that are key for the task, so increasing the receptive field (filter size) is a solution to capture them, but the number of parameters grows quadratically with its size. GoogleNet [18] included "inception blocks" that handled farther relations between features by branching and using different receptive fields, but they had to use 1×1 convolutions to reduce the number of parameters sacrificing information among input channels, and also they had to concatenate the different branches resulting in large output channels.

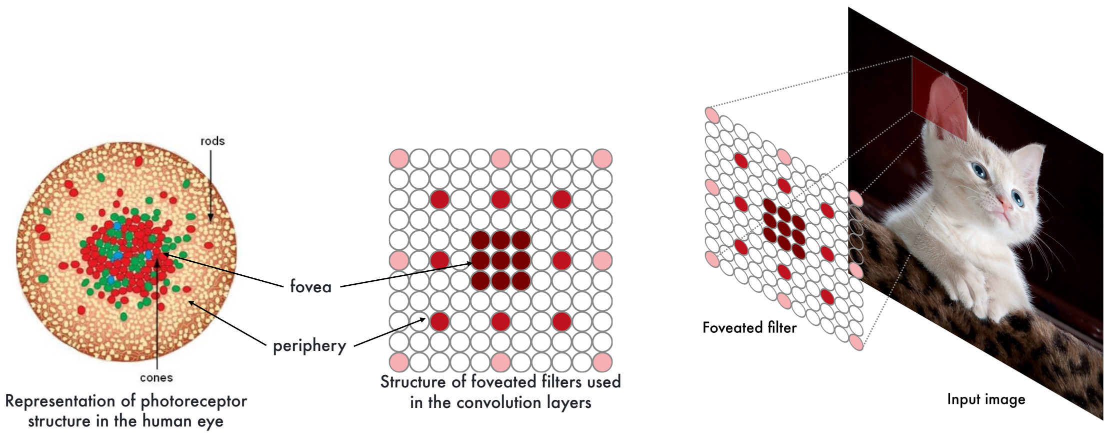
rods
fovea
periphery
cones
Representation of photoreceptor structure in the human eye
Structure of foveated filters used in the convolution layers
Foveated filter
Input image

Figure 1. Distribution of photoreceptors (cones and rods) in the human eye (*left*), indicating fovea and periphery. Example of a foveated filter (*center*) used in this work. Convolution applied to an image (*right*) using the foveated filter.

In recent years, however, Vision Transformer architectures have been able to outperform CNNs [4]. They exploit feature relations that are not necessarily local, at early depths of the Neural Network. This process results in learning relationships between parts of the image that CNNs are not capable of learning at early depths. Consequently, transformers have turned out to be the best image classifiers in recent years. Recent work has combined the locality and non-locality of the relations between image features, achieving an intermediate architecture between CNNs and Transformers. In this work, the filters described in Figure 1 attempt to significantly increase the receptive field of convolutional filters without sacrificing the number of parameters in the NN to exploit relations between image features that are not necessarily close. In this way, we aim to capture far feature relations in the image as Transformers, but keeping the simplicity of CNN architectures.

Conventional convolutional networks have been inspired by the mammalian visual system, where synaptic connections between neurons are localized. However, CNNs need more and more layers to achieve human-like performance in tasks as simple as image classification. The human visual system, on the other hand, does not have so many 'layers' but relies on a more complex network of synaptic connections. In addition, the structure of the retina presents an intelligently distributed distribution of receptors (cones), with more concentration in the center (fovea) and more dispersion in the periphery, as shown in Figure 1. And the area of cone reception is also variable depending on whether it is in the fovea or in the periphery [7] [5]. Within the fovea, the reception area is small and focused, whereas in the periphery it is scattered and random.

Foveated filters have previously been used for vision tasks [1] [12], where the accuracy was close to a normal CNN, but using fewer parameters. Another work has developed a vision algorithm that sequentially identifies fixation points in the image to integrate information obtained through convolution with foveated filters [9]. This algorithm is computationally more expensive than Transformers and its performance is similar, since it only uses five fixation points so as not to make it more expensive.

Foveation has also shown robustness against adversarial attacks [13] of NNs because they can alleviate the effect of the attack in the peripheral zone. Another ways of achieving robustness in Machine Learning models are to smooth the input image [19] or to artificially inject randomness [20] [17] [11] during training or testing. Other works attribute the robustness against white-box attacks on Spiking Neural Networks to the randomness of integration in the classification [15] [16], since this prevents an attacker learning about the gradients, and must be based on an estimated value of the gradients. This has motivated in our work the development of induced randomness in the foveated filters to obtain greater robustness against white-box attacks.

3. Methodology

3.1. Implementation of foveated filters

The implementation of the foveated filters was performed based on the cone density structure present in the human eye as indicated in Section 1. That is, a high density of cones in the center of the retina, called fovea, which results in high spatial resolution; and a low density in the peripheral part. An illustrative example is shown in Fig-

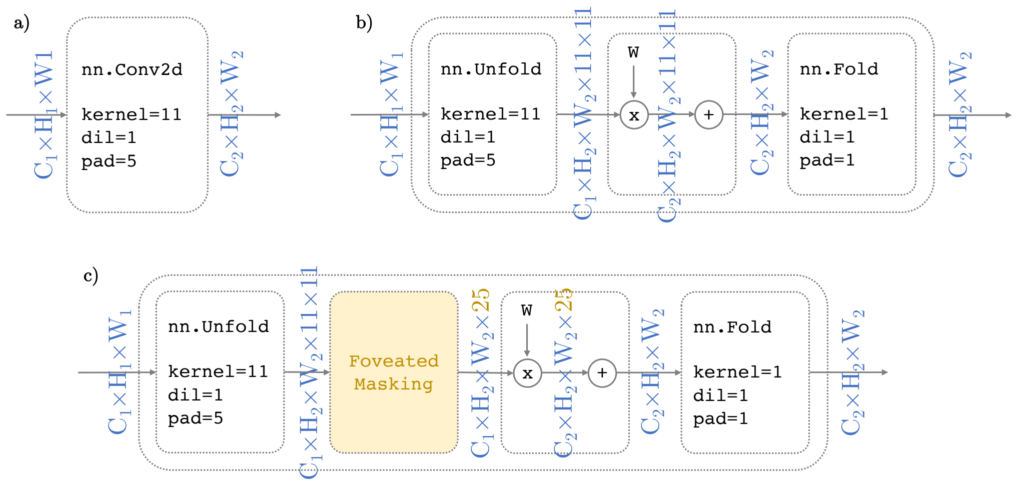

Figure 2. (*a*) Built-in convolution module for images implemented in PyTorch. (*b*) Equivalent operation of the built-in convolution module using unfolding to obtain the sliding windows, matrix multiplication with the weight matrix and folding. (*c*) Foveated convolution obtained by applying a mask to the inputs to reduce parameter storage of the weights. For all the three modules, the dimensions of data have been included in blue, and in the last implementation, the orange of the masking highlights the reduction of parameters from $11^2 = 121$ to 25. Notice that the input and (final) output dimensions are the same for all the implementations.

ure 1, which shows a dense foveal (dark) region of 3×3, a peripheral region with dilation 3 (distance between two receptors) and a peripheral region with dilation 5.

In order to benefit from the potential offered by GPUs to simulate neural networks, we use the PyTorch library. However, the operations offered in this library are not flexible enough to define the desired foveated filters without sacrificing a large amount of memory and computational cost. That is, the filters could well be implemented as conventional convolutional filters, forcing the values of the weights to zero where no receivers are desired (white circles in Figure 1). The problem with this implementation is that an excessive amount of memory is wasted to store the Neural Network, while wasting computational capacity in the forward and backward pass. In the example of the figure, it would require almost $5\times$ more memory and computation.

One way to save memory and computational cost would be to create specialized kernels written in CUDA, such as those developed by PyTorch for conventional convolutions. However, there are a large number of optimizations and heuristics on GPU grid processing parameters (depending on batch size, input dimensions, etc.) that are beyond the scope of this project. Therefore, the foveated convolution operation was performed with the `nn.Unfold` and `nn.Fold` functions provided by PyTorch. The `nn.Unfold` operation returns as result the succession of overlaps between the input and a sliding window, in the same way as the conventional convolution, but without processing the weighted sum with the filters. The `nn.Fold` operation performs the inverse operation to `nn.Unfold`, with the difference that where there are overlapping values from different slides, they are summed. With these two operations, a conventional convolution can be emulated by combining `nn.Unfold`, matrix multiplication and `nn.Fold`. The process is summarized in Figure 2. In order to assure that this process is equivalent to directly apply `nn.Conv2d`, we compared the result of both layers with different parameters using normalized random inputs, and then checked squared error difference between them. The relative difference was in the order of $10^{-6}\%$, which we assume is a rounding error produced by the difference in implementation of the two operations.

For foveated convolutions, however, prior to matrix multiplication, we apply a masking of the inputs to match the dimension of the weights, as indicated in Figure 2. The masks are built during the initialization of the layer, and the stored weights are the ones corresponding to that mask. The construction of the foveal structure of the filters is user-defined. That is to say, the class that we developed in PyTorch has the flexibility so that the user can define the regions that believes necessary, each one with its parameters. For example, the structure generated in Figure 1 is obtained with the following code:

```
Conv2dFoveatedDilated(inp, out,
  kernel_sizes=[3,3,3],
  dilations=[1,3,5]),
```

where `inp` and `out` are the input and output planes, there are three regions, all with kernels 3×3, although they differ in the dilations, which are $1, 3, 5$. Note that although there are three regions of 3×3 filters, the total amount of weights for that filter is 25, different from $3^3 = 27$. This is because the three regions are collapsed into the same filter and there is an overlap (redundancy) of parameters.

3.2. Induced randomness

The use of a masking after `nn.Unfold` as shown in Figure 2 not only allows to reduce the number of multiplications performed in the convolution, but also allows to induce randomness in the operation by suppressing True elements of the mask. Randomness induced during training is not a novel concept, as it has been widely demonstrated to strengthen the generalization of Neural Networks, and is known as dropout. Although the dropout normally used is applied after summation, that is, on the activations, the approach of this work is slightly different. The dropout is applied on the filter weights and, therefore, it is also applied to the foveated mask of the inputs.

Another difference with the dropout normally used in the literature is that during the inference process, instead of disabling it, the user can choose to retain or reduce it. While the accuracy of the network is affected by this induction of randomness during inference, the purpose is to provide robustness to the Neural Network, analogous to Spiking Neural Networks. It is worth noting that the random mask is the same for every sliding window, but different among different planes, since generating random mask for every sliding window would become a bottleneck in the forward pass.

4. Results

4.1. Time analysis

Although the implementation of the foveated convolution shown in Figure 2 allows significant memory savings compared to the dense version, it is necessary to compare it in terms of execution time with the native `nn.Conv2d` version implemented in PyTorch and with the custom version of `nn.Conv2d`. To do so, we generated different varieties of layers of these three types, varying the parameters and introduced random inputs. This experiment was repeated 200 times for each configuration and we calculated the mean and standard deviation. To get a parameter-wise analysis we first set the base parameters and then modified one parameter at a time. The parameters analyzed were batch size (N), input and output planes or channels ($C_{\text{in}}, C_{\text{out}}$), input size ($H_{\text{in}}, W_{\text{in}}$) and kernel size ($k_H, k_W$). The base parameters are $N = 32, C_{\text{in}} = 16, C_{\text{out}} = 16, H_{\text{in}} = 64, W_{\text{in}} = 64$, $k_H = 11, k_W = 11$. All the experiments were run on GPU, and the results are shown in Figure 3.

First, we can observe that the foveated convolution is affected by the batch size (N), while for the other two versions the execution time remains stable. This is because different batch elements are processed on different GPU kernels. On the other hand, we can observe that when more input and output planes or channels ($C_{\text{in}}, C_{\text{out}}$) are used, the foveated convolution is deteriorated. This implies that foveated convolution becomes slow in the deeper layers of a neural network, when the spatial dimensions are reduced and the channels are larger. This deterioration is compensated by the improvement in execution time when the spatial dimensions of the input are reduced ($H_{\text{in}}, W_{\text{in}}$). Finally, we can observe that for small kernel sizes (k_H, k_W) of the convolution, the foveated implementation is even as fast as the native one, although not as fast as the custom one.

4.2. Parameter efficiency

Our foveated convolution module offers the benefit of bigger receptive fields while accounting for reduction in the effective number of parameters due to the inherent nature of the small dense centre, bigger dilated peripheries. Although the filter is a sparse weight matrix, it can be stored as a reduced weight matrix after applying a sparse boolean mask (of the same structure as the foveated filter) during initialization. This can be possible because during the forward pass we can use the same boolean mask on the unfolded activations so that we can matrix multiply it with the reduced weights.

To quantify the efficacy of our filters and showcase the performance of our network we consider a ResNet 18 model, with it's convolution layers replaced with our foveated convolution layers. A 3x3 convolution filter is replaced with a 3x3 foveated filter with dilations 1, 3 and 5 (See the filter in Figure 1 for a visual representation). The 7x7 convolution filter is replaced with a 7x7 foveated filter with dilations 1, 2 and 4. Since a naive Resnet 18 model has 11.2 Million parameters and our foveated Resnet 18 model (filters replaced) has 30.7 Million parameters, we drop the odd numbered layers from the network, thereby reducing the number of parameters by approximately half. We then reduce the number of filters by half in each layer to further reduce the number of parameters. Figure 4 shows the performance of the reduced foveated Resnet model when compared to a baseline Resnet18 model on CIFAR 10 when trained from scratch.

4.3. Performance

To further evaluate the behavior of the foveated Resnet 18 model, we trained and evaluated it on the Caltech101 dataset. This dataset was chosen because it had relatively high resolution images (256x256 after downsampled resizing) and it was computationally inexpensive to train as it had few examples per class. We compare our model (30.8M parameters) with Resnet 50 (23.7M parameters) and Resnet 101 (42.7M parameters).

Figure 5 compares the validation accuracy trajectories of

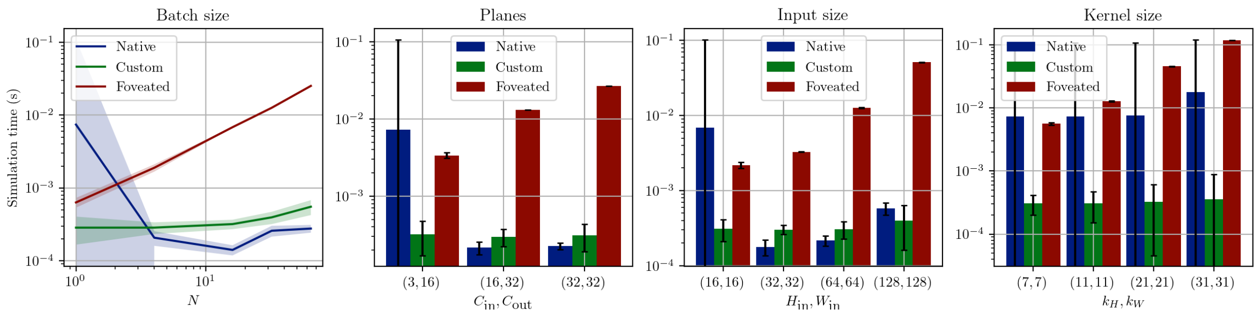

Figure 3. Execution time of three types of layers, native, customized and foveated, described in Figure 2. Different parameters were varied, once at a time. The base parameters were $N = 32$, $C_{\text{in}} = 16$, $C_{\text{out}} = 16$, $H_{\text{in}} = 64$, $W_{\text{in}} = 64$, $k_H = 11$, $k_W = 11$.

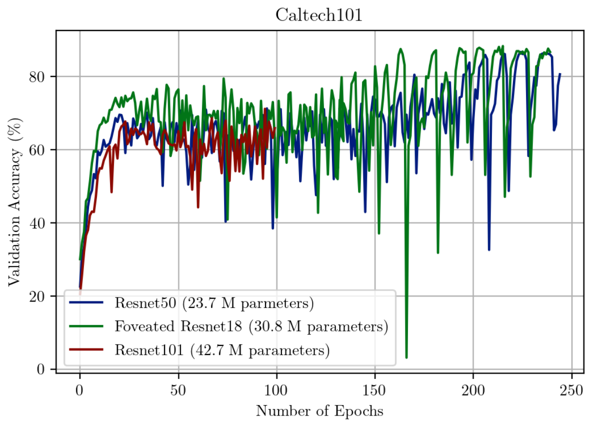

Figure 4. Validation accuracy vs Number of Epochs: The reduced parameter foveated Resnet18 model in-spite of being underparametrized and depth reduced in half, follows a similar validation accuracy trajectory to a naive Resnet 18 model, when training from scratch on CIFAR 10

our foveated Resnet 18 model with Resent50 (lesser number of parameters than ours) and Resnet101 (more number of parameters than ours). As seen in Figure 5, the foveated model reaches high validation accuracies faster than both the baselines especially in the first quarter of epochs. In the later epochs there is a high variance in the accuracy/loss. We hypothesised that this is due to a variety of factors i.e loss surface being too jaggedy, learning rate scheduler etc. However, we later found out that this may be due to the number of layers, as we observed the same behaviour in a naive Resnet18 (although the validation accuracies were lower than all the other models).

4.4. Robustness

Robustness against adversarial attacks is one of the features we hypothesized for the foveated convolutional model proposed in the Methodology Section. One of the arguments we put forward to justify this hypothesis is that randomness is a cause of robustness in Spiking Neural Networks, and is used for both training and inference. Similarly, our model is trained with randomness to acquire generalization, and during inference it preserves randomness, with the possibility of reducing it.

This robustness is accentuated especially in white-box attacks, where the Neural Network gradients are available. To test these hypotheses, we performed different white-box attacks on our foveated ResNet18 model with half layers. We tested three widely known white-box attacks: FGSM [6], PGD [14] and CW [3].

The results of the white box attacks are summarized in Figure 6. The three models were trained on CIFAR10, and have the same base architecture, foveated ResNet18 with half layers, but their difference is in the randomness of the masking. We analyzed a deterministic model, where all weights are used, a model with 20% suppression randomness during training and testing, and one with 40% suppression randomness during training and 20% during testing. In the FGSM attack, the three models present similar performances, since this attack consists of taking a step towards the sign of the gradient of the neural network loss with re-

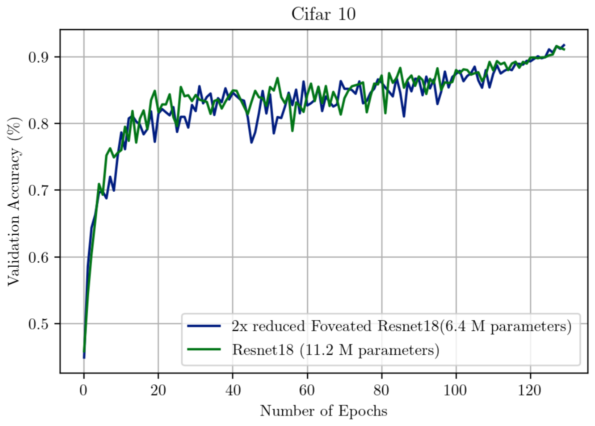

Figure 5. Validation accuracy trajectory of our model vs Resnet101 and Resnet50

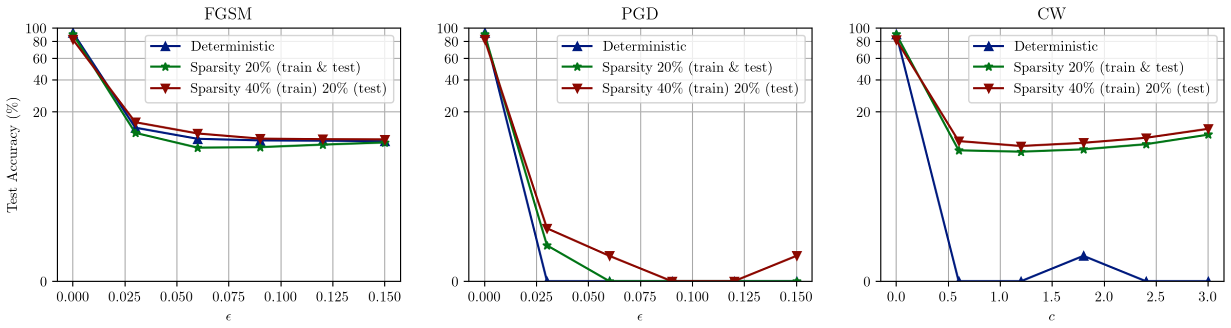

Figure 6. Test accuracy of foveated modules under different types of attacks. For PGD, the number of steps was 10 and $\alpha = 2/255$. For CW, the number of steps was 50 and $\lambda = 0.1$. The dataset utilized was CIFAR10 and the model was foveated ResNet18 with half the layers.

spect to the input. In all three cases, the model is similarly impaired. In the case of PGD attack, a certain degree of robustness of the random models with respect to the deterministic one can be appreciated. This robustness is more noticeable in the case of the CW attack, where it is clear that the random models are not easily fooled, although they do reduce their performance significantly. In these last two attacks, we can observe that the model trained with 40% sparsity and that reduces randomness during inference is slightly more robust than the one that maintains it.

4.5. Other experiments

One of our hypothesis was that such an architecture design for the filters would integrate more non-local (global) features in addition to the local features, and so would not fall prey to the texture bias problem posed by CNNs. To test this, our initial goal was to train the model on the entire Imagenet dataset and then test the model on images which had their textures removed. Removal of texture was done by using a canny edge filter to obtain images with just edges/corners and boundaries highlighted i.e a rough sketch of the image. There has been work that shows CNNs are poor at capturing shape based features that are necessary for classifying these sketch images [2], since they respond mostly to texture.

Unfortunately, we did not have the compute in our budget to train on ImageNet. Instead, we used a subset of ImageNet known as the Imagewoof [8] dataset, that contains images of specific breeds of dogs for classification. Our reason for choosing this dataset was that this dataset was extremely texture heavy and so the models learnt on this would bias more towards being texture-reliant. We trained both our foveated model as well as the naive Resnet18 models on the dataset. T Both of the models performed below our expectations with a test accuracy of only 58%. We carried on with our experiment to test the performance of both these models on the canny edge detector applied on these test images, and both these models gave 12% accuracy i.e slightly above random. Although this provides evidence that the hypothesis is false, we would like to repeat the same test experiment after training the models on complete ImageNet before making any judgements.

5. Conclusion

As part of our project, we evaluated the parameter efficiency, performance and robustness of CNN models with foveated convolutions. For small to medium datasets, we found that our models (with foveated filters) offer the benefits of high test accuracies when trained from scratch while having a reduced number of parameters via halving the network depth and number of filters, compared to naive CNN models. Additionally, we also tested the adverserial robustness of our models with varying levels of randomness (in dropping of the non-zero weights) against the popular adverserial attacks in literature. We observed a correlation in the percentage of randomness induced in the filters and adverserial robustness. As part of future work, we wish to train the models on Imagenet and wish to repeat the experiments. We would also like to explore patchy weight regions in the peripheries instead of dilated weights. This might be more helpful to aggregrate non-local features than our dilated approach. We would also like to implement dropping of the periphery weights rather than dropping over the entire foveated weight matrix. Keeping the structure of the dense filter intact might stabilize training.

References

[1] Emre Akbas and Miguel P Eckstein. Object detection through search with a foveated visual system. *PLoS computational biology*, 13(10):e1005743, 2017. 2

[2] Pedro Ballester and Ricardo Matsumura Araujo. On the performance of googlenet and alexnet applied to sketches. In *Thirtieth AAAI conference on artificial intelligence*, 2016. 6

[3] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017. 5

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020. 2

[5] E Bruce Goldstein and Laura Cacciamani. *Sensation and perception*. Cengage Learning, 2021. 1, 2

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 5

[7] Ethan William Albert Harris, Mahesan Niranjan, and Jonathon Hare. Foveated convolutions: improving spatial transformer networks by modelling the retina. 2019. 2

[8] Jeremy Howard. Imagewang. 6

[9] Aditya Jonnalagadda, William Yang Wang, BS Manjunath, and Miguel P Eckstein. Foveater: Foveated transformer for image classification. *arXiv preprint arXiv:2105.14173*, 2021. 2

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[11] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 369–385, 2018. 2

[12] Hristofor Lukanov, Peter König, and Gordon Pipa. Biologically inspired deep learning model for efficient foveal-peripheral vision. *Frontiers in Computational Neuroscience*, 15, 2021. 2

[13] Yan Luo, Xavier Boix, Gemma Roig, Tomaso Poggio, and Qi Zhao. Foveation-based mechanisms alleviate adversarial examples. *arXiv preprint arXiv:1511.06292*, 2015. 2

[14] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017. 5

[15] Wilkie Olin-Ammentorp, Karsten Beckmann, Catherine D Schuman, James S Plank, and Nathaniel C Cady. Stochasticity and robustness in spiking neural networks. *Neurocomputing*, 419:23–36, 2021. 2

[16] Saima Sharmin, Nitin Rathi, Priyadarshini Panda, and Kaushik Roy. Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations. In *European Conference on Computer Vision*, pages 399–414. Springer, 2020. 2

[17] Baifeng Shi, Dinghuai Zhang, Qi Dai, Zhanxing Zhu, Yadong Mu, and Jingdong Wang. Informative dropout for robust representation learning: A shape-bias perspective. In *International Conference on Machine Learning*, pages 8828–8839. PMLR, 2020. 1, 2

[18] 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. 1

[19] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017. 2

[20] Fan Yang, Zhiyuan Chen, and Aryya Gangopadhyay. Using randomness to improve robustness of tree-based models against evasion attacks. In *Proceedings of the ACM International Workshop on Security and Privacy Analytics*, pages 25–35, 2019. 2