CS7643 Spring '24: Assignment 2

Tyler Burki
tburki3@gatech.edu

February 18, 2024

1 Theory Problem Set

1.1 Convolution as a matrix operation

Since we are zero padding the input with a size of two, let's first write out the input X with the padding applied:

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_{(0,0)} & x_{(0,1)} & x_{(0,2)} & 0 & 0 \\ 0 & 0 & x_{(1,0)} & x_{(1,1)} & x_{(1,2)} & 0 & 0 \\ 0 & 0 & x_{(2,0)} & x_{(2,1)} & x_{(2,2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

Then, if we apply the 3x3 filter W to X with a stride of four, we derive the following four equations:

$$W \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & x_{(0,0)} \end{bmatrix} = w_{(2,2)} x_{(0,0)} \tag{2}$$

$$W \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ x_{(0,2)} & 0 & 0 \end{bmatrix} = w_{(2,0)} x_{(0,2)} \tag{3}$$

$$W \cdot \begin{bmatrix} 0 & 0 & x_{(2,0)} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = w_{(0,2)} x_{(2,0)} \tag{4}$$

$$W \cdot \begin{bmatrix} x_{(0,2)} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = w_{(0,0)} x_{(2,2)} \tag{5}$$

We note that this filter and stride effectively multiplies the corners of W with the corresponding, opposite corner of X. The four values derived above represent the solution y:

$$y = \begin{bmatrix} w_{(2,2)}x_{(0,0)} & w_{(2,0)}x_{(0,2)} & w_{(0,2)}x_{(2,0)} & w_{(0,0)}x_{(2,2)} \end{bmatrix}^T \tag{6}$$

Therefore, to construct a matrix A representing the affine transformation, we can define a 4x9 (the number of elements in X) matrix in which each row contains the value of W in the position of the index of X corresponding to the values derived in equations (2) through (5):

$$A = \begin{bmatrix} w_{(2,2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{(2,0)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{(0,2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{(0,0)} \end{bmatrix} \tag{7}$$

Using these values we can see the equation $y = Ax$ is satisfied:

$$Ax = \begin{bmatrix} w_{(2,2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{(2,0)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{(0,2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{(0,0)} \end{bmatrix} \cdot \begin{bmatrix} x_{(0,0)} \\ x_{(0,1)} \\ x_{(0,2)} \\ x_{(1,0)} \\ x_{(1,1)} \\ x_{(1,2)} \\ x_{(2,0)} \\ x_{(2,1)} \\ x_{(2,2)} \end{bmatrix} = \begin{bmatrix} w_{(2,2)}x_{(0,0)} \\ w_{(2,0)}x_{(0,2)} \\ w_{(0,2)}x_{(2,0)} \\ w_{(0,0)}x_{(2,2)} \end{bmatrix} = y \tag{8}$$

(a) What are the dimensions of A?

4x9

(b) Write down the entries in this matrix A

$$A = \begin{bmatrix} w_{(2,2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{(2,0)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_{(0,2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_{(0,0)} \end{bmatrix} \tag{9}$$

1.2 ReLU network

Let $h(x) = W^{(3)}(h_2(x)) + b^{(3)}$ where

$$h_1(x) = max\{0, W^{(1)}x + b^{(1)}\} \tag{10}$$

$$h_2(x) = max\{0, W^{(2)}h_1(x) + b^{(2)}\} \tag{11}$$

Then

$$\frac{\partial h_1(x)}{\partial x} = \begin{cases} W^{(1)} & W^{(1)}x + b^{(1)} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

$$\frac{\partial h_2(x)}{\partial x} = \frac{\partial h_1(x)}{\partial x} \cdot \begin{cases} W^{(2)} & W^{(2)}h_1(x) + b^{(2)} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

$$\frac{\partial h(x)}{\partial x} = \frac{\partial h_2(x_o)}{\partial x} \cdot W^{(3)} \tag{14}$$

The ReLU activations mask the negative weights for each layer where they are applied with 0. Therefore, we are essentially calculating $W^{(1)}W^{(2)}W^{(3)}$ where any weights that cause their respective linear operation to be negative are zeroed out.

For $x_o = 2$:

$$h_1(x_o) = max\{0, \begin{bmatrix}1.5\\0.5\end{bmatrix} \cdot 2 + \begin{bmatrix}0\\1\end{bmatrix}\} = max\{0, \begin{bmatrix}3\\2\end{bmatrix}\} = \begin{bmatrix}3\\2\end{bmatrix} \tag{15}$$

$$h_2(x_o) = max\{0, \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix} \cdot \begin{bmatrix}3\\2\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\} = max\{0, \begin{bmatrix}7\\9\end{bmatrix}\} = \begin{bmatrix}7\\9\end{bmatrix} \tag{16}$$

$$h(x_o) = \begin{bmatrix}1 & 1\end{bmatrix} \cdot \begin{bmatrix}7\\9\end{bmatrix} - 1 = 16 - 1 = 15 \tag{17}$$

We can then derive W from equations (12) - (14):

$$\frac{\partial h(x)}{\partial x}|_{x=x_0} = \begin{bmatrix}1 & 1\end{bmatrix} \cdot \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix} \cdot \begin{bmatrix}1.5\\0.5\end{bmatrix} = W = 6 \tag{18}$$

and solve for b

$$6 \cdot 2 + b = 15 \rightarrow b = 3 \tag{19}$$

Therefore $W = 6$ and $b = 3$.

For $x_o = -1$:

$$h_1(x_o) = max\{0, \begin{bmatrix}1.5\\0.5\end{bmatrix} \cdot (-1) + \begin{bmatrix}0\\1\end{bmatrix}\} = max\{0, \begin{bmatrix}-1.5\\0.5\end{bmatrix}\} = \begin{bmatrix}0\\0.5\end{bmatrix} \tag{20}$$

$$h_2(x_o) = max\{0, \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix} \cdot \begin{bmatrix}0\\0.5\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\} = max\{0, \begin{bmatrix}1\\1.5\end{bmatrix}\} = \begin{bmatrix}1\\1.5\end{bmatrix} \tag{21}$$

$$h(x_o) = \begin{bmatrix}1 & 1\end{bmatrix} \cdot \begin{bmatrix}1\\1.5\end{bmatrix} - 1 = 2.5 - 1 = 1.5 \tag{22}$$

We can then derive W from equations (12) - (14):

$$\frac{\partial h(x)}{\partial x}|_{x=x_0} = \begin{bmatrix}1 & 1\end{bmatrix} \cdot \begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix} \cdot \begin{bmatrix}0\\ 0.5\end{bmatrix} = W = 1.5 \tag{23}$$

and solve for b

$$1.5 \cdot (-1) + b = 1.5 \rightarrow b = 3 \tag{24}$$

Therefore $W = 1.5$ and $b = 3$.

For $x_o = 1$:

$$h_1(x_o) = max\{0, \begin{bmatrix}1.5\\ 0.5\end{bmatrix} \cdot 1 + \begin{bmatrix}0\\ 1\end{bmatrix}\} = max\{0, \begin{bmatrix}1.5\\ 1.5\end{bmatrix}\} = \begin{bmatrix}1.5\\ 1.5\end{bmatrix} \tag{25}$$

$$h_2(x_o) = max\{0, \begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix} \cdot \begin{bmatrix}1.5\\ 1.5\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix}\} = max\{0, \begin{bmatrix}4.5\\ 5.5\end{bmatrix}\} = \begin{bmatrix}4.5\\ 5.5\end{bmatrix} \tag{26}$$

$$h(x_o) = \begin{bmatrix}1 & 1\end{bmatrix} \cdot \begin{bmatrix}4.5\\ 5.5\end{bmatrix} - 1 = 10 - 1 = 9 \tag{27}$$

We can then derive W from equations (12) - (14):

$$\frac{\partial h(x)}{\partial x}|_{x=x_0} = \begin{bmatrix}1 & 1\end{bmatrix} \cdot \begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix} \cdot \begin{bmatrix}1.5\\ 0.5\end{bmatrix} = W = 6 \tag{28}$$

and solve for b

$$6 \cdot 1 + b = 9 \rightarrow b = 3 \tag{29}$$

Therefore $W = 6$ and $b = 3$.

2 Paper Review

This section reviews and analyzes the 2018 paper *ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.* by Geirhos et al. All citations in this section will be denoted solely by the page number from this text.

2.1 Review

Geirhos et al. challenge the "shape hypothesis" in convolutional neural networks (CNNs) that states CNNs build representations of image subjects' shapes (p.2). They claim that CNNs tend to rely heavily on object textures in practice with an opposing theory they have deemed the "texture hypothesis" (p.2). The authors provide evidence that humans identify objects in images through shapes (p.2) and that this method is more robust to noise than the use of textures (p.9).

Texture bias in CNNs was shown by comparing performance on sets of images stylized in ways that both preserved textures and removed them (p.4). Humans and CNNs were able to correctly classify images in the groups with textures (p.5), but CNNs lagged humans in the texture-less groups. One grouping kept the original shape and stylized the image with the texture from a different category. Humans responded with the category of the shape while CNNs responded with the category of the texture (p.5). The authors present a promising procedure to encourage the use of shapes (p.5) that suggests replacing the textures in images with the style of randomly selected artistic paintings.

The publication does well laying out its argument: from providing evidence of humans' use of shapes for classification to showing how CNNs prefer textures. Providing a framework for shifting the bias of CNNs towards shapes and then showing improvement using that framework is an important facet of this work that aids future researchers in improving outcomes. While I personally did not find the improvements to be particularly compelling, I admit that I am a novice in the area and likely have a distorted perception of the significance.

I am used to systems where all inputs, states, representations, etc. can be known and understood at all points during an application's lifecycle. The way the authors tied success to modeling the human method gave me some perspective. How can we understand and model human methodology for this task? That is an extremely complicated question, and knowing that researchers are not only attempting and succeeding at this feat shows me the power and importance of this field of study.

2.2 Questions

Why would we care about the biases of the neural network, even if it can do well on the training set? Should we desire the network to have the same biases as humans or not? Why or why not?

My intial reaction to the question placed me on the side of "accuracy at all costs". What does it really matter if the model behaves in the same manner as humans as long as the outcome is correct? However, I came to ask myself what the underlying goal of such a task is. Is it to uncover ground truth even if that truth is not useful to or interpretable by humans?

The authors of this paper suggest that CNNs' selection of the elephant class for an image of a cat texturized as an elephant is the incorrect choice (p.2). For argument's sake, imagine the ground truth is that the image is of an anatomical elephant that happens to look very much like a cat. Because that sounds ridiculous, humans would likely interpret the texture as noise in the image and select the cat category. If *every* human looks at the image and believes it is a cat, and that something certainly must be wrong with the model because we *desire* it to be a cat, is the truth of consequence?

I will therefore rest on the only solution I believe is valid: models are built to do work for humans and should model the behavior expected of a high-performing human. We can still alter the model on the arrival of new data or domain knowledge. For example, if the existence of a herd of cat-shaped elephants is discovered or biologists reclassify animals into taxonomies based on texture.

That is not to say that innovation should be suppressed. Encouraging progress to occur through channels accessible to humans can improve not only our comprehension of the model itself but the domain it operates in.

Why would training on stylized images change the bias of the network, and why might this bias generalize better to datasets with corruptions?

The authors show that networks trained on the images styled by random paintings evaluate more poorly than those trained on the original images (p.5). It follows that classifying the stylized images is a more difficult problem. However, it was also shown that the stylized model generalizes well to the non-stylized data but the reverse is not true (p.5). We can then conclude that the stylized model is learning features that are more indicative of the ground truth of the image. Randomized textures remove the feature as a viable decision factor and force the model to learn long-range spatial information (p.6).

This is important for generalization because local textures can be easily distorted (e.g. rain, snow) while shape is not as easily manipulated in practice (p.9). The argument can be made that because humans take a shape-based approach to object recognition (p.2) it is the best way to navigate the world (p.9) since we also could have evolved to rely on textures but did not.

For corruptions not normally seen in nature, the authors show that the shape-based model handles them nearly as well as humans (p.7,8) even if the network was not trained on the distortion. These are distortions, of course, that keep the shape of the subject relatively similar to the original via scaling, re-texturizing, blurring, etc. The authors do note that blurring did not perform as highly as expected with the shape-based model, but point out potential failures in the ability to properly stylize the images for this type of distortion (p.7). This is a level of robustness not available to texture-based models, as any degredation of the texture removes the cues necessary to correctly classify the image.

3 Coding

3.1 CNN from scratch

Figure 1 is the result of training on the CNN.

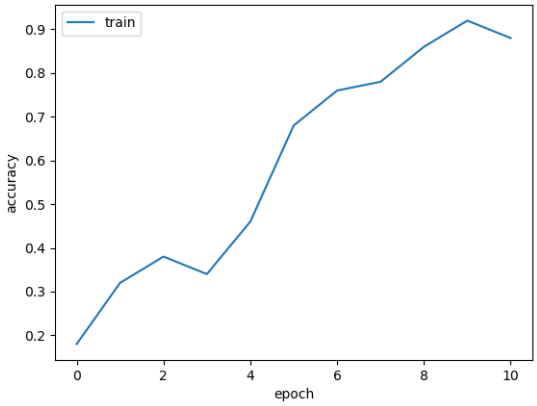

Figure 1: Accuracy of CNN over 10 epochs

3.2 PyTorch

Custom model design

I began training with vanilla CNN on default parameters. I noticed that learning was not happening quickly enough (very small improvements in accuracy over many epochs) and that the model was significantly underfit (validation scores much higher than training). I knew this meant I needed to increase the rate of learning as well as boost the representational power of my model.

Justification of model design

To increase reprsentational power, I added additional convolutional layers at the beginning of the network in an attempt to extract more features. I had the best luck by increasing the number of filters in each layer by a power of two. I added a batch normalization before each non-linearity in the convolution steps to prepare the output for ReLU activation. Max pooling was employed after each activation to reduce dimensionality, provide robustness through translation invariance, and help the network focus on the most pertinent features. On the back end of the model, I employed additional linear layers to help guide the features derived in the convolutional layers back towards the classifications that needed to be made. These improvements over the vanilla CNN greatly increased the representational power of my model and allowed me to achieve close to the desired accuracy.

Justification of model parameters

After increasing the representational power, learning was still happening too slowly and the model would routinely become stuck at local minima/saddle points. To handle this, I began by increasing the learning rate significantly from the default. While learning was now happening more quickly, the model began to oscillate around local minima for an epoch or more before finally finding its

way out. This ultimately lead to less than desired performance over the number of epochs trained. To alleviate the issue I nearly cut the momentum parameter in half to account for the larger steps taken during gradient descent as a result of the increased learning rate. The reduction in momentum caused the model the follow gradients more closely while still providing a mechanism for pushing the model out of small local minima. As a result, training was happening more quickly and reliably. The final values I settled on changing from the default were $learning\_rate : .0001 \rightarrow .1$ and $momentum : .9 \rightarrow .5$.

Final accuracy on validation set

The changes above produced a final accuracy of 0.8162 on the validation set.

3.3 Data wrangling

Best accuracy using CE loss

The best accuracy I could produce occurred when increasing the *learning_rate* from 0.0001 to 0.1 and decreasing the *momentum* from 0.9 to 0.3. The overall accuracy was 0.2876. Results can be reproduced with the *config_resnet32_ce.yaml* configuration.

Class	0	1	2	3	4	5	6	7	8	9
Accuracy w/ CE Loss	0.9210	0.8510	0.4610	0.3170	0.1850	0.0420	0.0990	0.0	0.0	0.0

Table 1: Per-class accuracy on imbalanced CIFAR-10 dataset using cross entropy loss

Tuning β with CB-Focal loss

In this experiment, we re-use the best parameters found in the previous section and switch out cross entropy loss for class-based focal loss. Only the new β parameter is modified in order to observe its effect on an established result. The γ parameter has been left at 1 (default) for this experiment. As β grows the under-represented classes increase in accuracy. Results can be reproduced with the *config_resnet32_focal.yaml* configuration.

Class	0	1	2	3	4	5	6	7	8	9
$\beta = 0$	0.9330	0.8880	0.4730	0.3790	0.1300	0.0090	0.0740	0.0090	0.0	0.0
$\beta = 0.25$	0.9200	0.8310	0.4530	0.3670	0.0850	0.0	0.0880	0.0010	0.0	0.0
$\beta = 0.5$	0.9460	0.8910	0.4810	0.3970	0.1540	0.0100	0.0620	0.0100	0.0	0.0
$\beta = 0.75$	0.9380	0.8950	0.4820	0.3650	0.1110	0.0050	0.1200	0.0440	0.0	0.0
$\beta = 0.9999$	0.6740	0.7330	0.3170	0.2970	0.1690	0.2740	0.4020	0.4340	0.3710	0.3170

Table 2: Per-class accuracy on imbalanced CIFAR-10 dataset using CB-Focal loss by β

Best results CE loss and CB-Focal loss

Parameters that were changed from default for CB-Focal loss: *learning_rate* from 0.0001 to 0.1,

momentum from 0.9 to 0.3, γ from 1 to 0.5. The β parameter was left at 0.9999. Results for CB-Focal loss can be reproduced with the *config_resnet32_best_focal.yaml* configuration.

Class	0	1	2	3	4	5	6	7	8	9	Valid.
CE	0.9210	0.8510	0.4610	0.3170	0.1850	0.0420	0.0990	0.0	0.0	0.0	0.2876
CB-Focal	0.6800	0.7730	0.3940	0.2300	0.1520	0.3150	0.4440	0.4410	0.3710	0.2840	0.4084

Table 3: Best per-class accuracy on imbalanced CIFAR-10 dataset for both CE and CB-Focal loss

Correctness of focal loss solution

After implementing focal loss, I noticed that the loss essentially becomes cross entropy when $\gamma = 0$ and $\beta = 0$. Therefore, I began by comparing the results of training against focal loss with the same hyperparameters as cross entropy and with $\gamma = 0$, $\beta = 0$. This test gave me a nearly identical result for both loss functions, indicating that I was on the right track.

From this point, we can look at what effect β has on the results and see if it aligns with expectations. When $\beta = 0$ no re-weighting is performed and when β is close to 1 re-weighting is occurring by inverse class frequency (Cui et al., 2019, p.4). This should mean that as we increase β we should also see the representation of sparsely sampled classes in the model increase as well. This appears to be the case from *Table 2* where we see the per-class accuracy decreasing for well represented classes and increasing for the poorly represented classes as β increases towards one.

Finally, we will examine how results on an imbalanced data set using CB-Focal loss compare to those from a balanced data set using cross entropy. While the results should not be identical due to the loss of information in the imbalanced set, we should be able to obtain similar results when $\beta = 0.9999$, assuming the same hyperparameters. For this experiment, I used the default hyperparameters and tested both cases.

Class	0	1	2	3	4	5	6	7	8	9
CE: balanced	0.4300	0.5260	0.1610	0.1500	0.3080	0.2710	0.4600	0.3360	0.4440	0.4290
CB-Focal: imbalanced	0.5640	0.5740	0.1810	0.0840	0.3230	0.1400	0.1370	0.0550	0.2340	0.1410
CE: class rank	1	0	8	9	6	7	2	5	3	4
CB-Focal: class rank	1	0	4	8	2	6	7	9	3	5

Table 4: Per-class accuracy on balanced data with CE and imbalanced data with CB-Focal loss

While not as definitive as desired, we can see that the two methods generate (very) roughly the same accuracy distributions. The "rank" rows in *Table 4* order the classes from highest accuracy (0) to lowest accuracy (9) within each test. The mean difference between the ranks is 2.3333, inidicating the tests are relatively similar in the indicative features extracted. We should be able to attribute these differences to the imbalanced data set because, while we did our best to nullify the constraint, the balanced data set still has information that the imbalanced set does not and likely was able to derive more descriptive features.

With all tests generally following expectations, we can conclude that the implementation is correct.

Observation of result

With standard cross entropy loss, the model significantly overfits the training data. This is due to the imbalance in sampling causing the model to capture features from classes with more examples and ignore features of classes with fewer. To combat this, I thought I needed to slow down learning and increase regularization in an attempt to avoid having the weights associated with the more highly sampled examples explode and cause the model to generalize poorly. I needed to find the balance between learning *nothing* and learning enough from the pervasive classes to be able to generalize decently well.

However, I found that in practice the above led to worse generalization error than I was expecting. By turning up regularization and reducing the learning rate I was receiving a *worse* approximation of the target concept over more epochs. In fact, the best results were produced when I leaned into the overfitting by keeping regularization at default and turning up the learning rate. Then, at least the over-represented samples were classified at a high level and that "made up" for the inability to learn anything about the other classes.

It is clear that using class-based focal loss over cross entropy for imbalanced data sets is a worthwhile endeavor, as *Table 3* shows nearly a 12% increase in validation accuracy for the former. Intuitively, focal loss of this type makes sense as we are attempting to mitigate the imbalanced distribution with one that more closely reflects the true distribution. It is also no surprise that all models performed more poorly on the imbalanced data since we are approximating the distribution *on top of* attempting to extract features representative of the classes. In many cases, data for teaching the model critical features for classification may simply not appear in the imbalanced set. Even with tools to temper the influence of the important features that do exist in the set, without the necessary data it can be difficult or impossible to arrive at a good approximation of the concept.

The most interesting result from tuning the model with CB-Focal loss on the imbalanced data set was the effect of the γ parameter. Defaulted to 1, I found the most success with my model with values around 0.5. The further the value strayed from the middle, the worse validation accuracy became. This is due to how γ is applied in the loss function, where it is an exponent applied to a formulation of cross entropy:

$$loss = mean(CE * (1 - e^{-CE})^{\gamma}) \tag{30}$$

Because $\gamma \, \epsilon \, [0, 1]$, as γ decreases the value of the second term in equation (30) increases and vice-versa. Therefore, with too small of a γ the influence of class-balancing is minimized and with too large of a γ the impact could be too great. When $\gamma = 0$, the second term is 1 and $loss = CE$. When $\gamma = 1$, the second term is maximally impactful on the cross entropy.

References

[Cui et al., 2019] Cui, Y., Jia, M., Lin, T.-Y., Song, Y., & Belongie, S. (2019). Class-Balanced Loss Based on Effective Number of Samples. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2019.00949.

[Geirhos et al., 2018] Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F.A., & Brendel, W. (2018). ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *ArXiv (Cornell University)*.