

1 **A Appendices for: The Regularizing Effect of Different Output Layer** 2 **Designs in Deep Neural Networks**

3 **A.1 Implementation and training details**

4 **Architectures** For ResNet [7], VGG [18] and DenseNet [8] we used implementations from PyTorch
5 Hub [15]. For U-Net [16], the implementation of [6] is used. In the decoder of U-Net, the upsampling
6 is learned with transposed convolutions. In the low-resolution datasets CIFAR-100 (C100) [12] and
7 STL-10 [3], the first convolutional layer in both ResNet and DenseNet is adjusted to have a kernel
8 size of 3, a stride of 1, and a padding of 1. In addition, the first maxpool is omitted.

9 The penultimate layer in DenseNet consists of concatenated nodes from different previous layers.
10 Thus, these nodes or activations have different semantics and representational capacities. However, in
11 W^{1to1} and $W^{ensemble}$, it should not matter which specific channels are used as logits vector(s). Due
12 to this specific connectivity pattern, we insert a group of 1x1 conv layer, BN and ReLU with the same
13 number of input and output channels before the output layer in DenseNets for W^{1to1} and $W^{ensemble}$.

14 **Hyperparameters** Both $\alpha = 0.1$ for W^{scale} and $q = 0.9$ for W^{sparse} were motivated by early
15 experiments to reduce neuron dependency and/or increase neuron expressivity but were not tuned to
16 individual settings. In comparison, hyperparameters are tuned for the methods we compare against.
17 Grid search is applied for dropout [19] and dropconnect [23] (options for drop probability: 0.2, 0.5,
18 0.7, 0.9). In both methods, a large value of 0.7 worked best. For additive gaussian noise [4], we
19 found that less noise generally performs better, so we stick to $\sigma = 0.1$ in all settings. If not specified
20 otherwise, hyperparameter H for $W^{ensemble}$ is set to its maximum given any architecture/dataset
21 setting, i.e. $H = \lfloor N/K \rfloor$. For semantic segmentation, this maximum is not defined. We therefore
22 set $H = 10$, which should be large enough to notice a difference compared to the baseline. For
23 the proposed layers, we tune only α in $W^{ensemble}$ (options: 0.5, 1.0, 2.0). Larger values result in
24 faster training as smaller activation values are amplified. In settings without pre-training, $\alpha = 0.5$
25 and $\alpha = 1.0$ work best. For fine-tuning, $\alpha = 2.0$ is beneficial in multiple settings. In general,
26 hyperparameter tuning is conducted on a small development set, which is constructed from the
27 original training set. See also the ablation study in Sect. 5.7, which gives more insights about
28 appropriate hyperparameter choices.

29 **Optimization** Our setup is similar to prior works (e.g., [7, 8]). In most settings, we use stochastic
30 gradient descent (SGD) with Nesterov momentum [20] of 0.9 without dampening and weight decay
31 of 0.0001. For medical imaging datasets, we chose Adam [10] because it consistently performed
32 better in early experiments. The learning rate is 0.1 in SGD and 0.001 in Adam. For fine-tuning and
33 SGD, the initial learning rate is reduced to 0.01. During training, the learning rate is reduced by a
34 factor of 10 each time performance reaches a plateau with a patience level of 15 epochs. The number
35 of epochs varies for different datasets. In ImageNet (IN) [17] we train for 90 epochs, for the other
36 datasets the number of epochs varies between 200 and 300 depending on the training progress. The
37 batch size for IN and APTOS [1] is 256. In APTOS, we achieved better results with larger batch
38 sizes, possibly due to class imbalance. In order to run multiple experiments in parallel, we chose a
39 smaller batch size of 32 for all other datasets. For segmentation, the batch size is set to 16 due larger
40 image dimensions of 512×512 .

41 **Preprocessing and augmentation** Images from all datasets are standardized by subtracting the
42 mean and dividing by the standard deviation of each channel of the training data. In fine-tuning
43 settings, we use the statistics of IN. In CUB-200 [22], Cars-196 [11], Food-101 [2] and IN, images
44 are resized with the shorter side randomly sampled in [256, 480] while maintaining the aspect ratio,
45 and then randomly cropped to 224×224 . For evaluation, images from these datasets are resized
46 to 256×256 and 10-crop testing [13] is applied. Except for segmentation datasets, we use random
47 horizontal flips. In APTOS, images are resized to 224×224 , randomly flipped along the vertical
48 axis and scaled (85 – 115% of original dimensions). In SLIVER [21] and CHAOS [9], we clamp
49 intensities according to the 99th intensity percentile to remove outliers. In C100 and STL-10, small
50 translations are added, which are implemented as random crops with a padding of 4 on all sides.

51 **Compute resources** The experiments were performed on a cluster consisting of 5x GeForce RTX
52 2080 Ti, 2x TITAN RTX and 4x A100-SXM4-40GB. These resources were used according to their

53 availability. The project ran over a period of about 3 months. With an estimated usage of one GPU
54 for 12 hours per day, the total emissions amount to an estimated 117 kg of CO₂eq, of which 0% was
55 directly offset (estimation according to [14]). We plan to track resource usage more closely in future
56 projects.

57 **Licenses and miscellaneous** We use PyTorch as deep learning framework, as well as its pre-
58 trained models, which are open-source and released under a BSD license. The U-Net implementation
59 of [6] is released under GNU General Public License v3.0. Captum is leveraged to measure neuron
60 dependencies/expressivities and is released under the BSD license. Lucent [5] is applied for feature
61 maximization visualizations in Sect. A.6 under the Apache 2.0 license. All datasets in this paper are
62 available for research purposes. ImageNet partly contains images showing persons (see also [24]).
63 Datasets mentioned in Sect. 5.3 were used in past machine learning challenges and contain de-
64 identified data to the best of our knowledge. Further details about all datasets, including how they
65 were acquired, can be found in the respective references.

66 A.2 Code

67 We provide a colab notebook as supplement which implements all output layers for a ResNet. All
68 layer types are easy to implement and require only a few lines of code to replace the standard output
69 layer. This is shown for PyTorch in the following.

70 In W^{random} , parameter updates are deactivated as follows:

```
71 self.fc = nn.Linear(N, K)  
72 self.fc.requires_grad_(False)
```

73 In W^{scale} , activations resulting from the encoder are scaled before the fc output layer:

```
74 def forward(self, x):  
75     x = self.encoder(x)  
76     x = x * alpha  
77     return self.fc(x)
```

78 In W^{sparse} , an additional sparsification function is applied to W^{random} :

```
79 def sparsify(self, q):  
80     K, N = self.fc.weight.data.shape  
81     num_to_remove = int(q * N)  
82     idx = torch.stack([torch.randperm(N)[:num_to_remove] for _ in range(K)])  
83     for i in range(K):  
84         self.fc.weight.data[i, idx[i]] = 0.
```

85 In W^{1to1} and $W^{ensemble}$, one can take a slice from the activations according to the number of classes
86 and heads without affecting the backbone specifications. For W^{1to1} , $H = 1$ and $\alpha = 1$.

```
87 def forward(self, x):  
88     x = self.encoder(x)  
89     x = x * alpha  
90     return x[:, :K*H]
```

91 In addition, $W^{ensemble}$ requires to compute separate losses for each head:

```
92 loss = 0  
93 for h in range(H):  
94     loss += nn.CrossEntropyLoss()(pred[:, h*K:h*K+K], y)  
95 loss /= H
```

96 A.3 Additional dependency/expressivity results

97 This section extends Sect. 3 from the main paper. Table 7 shows instance-based dependencies for
98 all proposed layers and the ResNet backbone in several datasets. In C100, STL-10 and CUB-200,
99 instance-based dependencies are high compared to the other output layers. In CUB-200, e.g., ablating

Table 7: Instance-based dependencies as avg. reduction in probability when ablating the most important neuron per instance w.r.t. the output class. + denotes fine-tuning from a pre-trained model.

	C100	STL-10	CUB-200	CUB-200+
$W^{trained}$	0.1014	0.1375	0.2395	0.0165
W^{scaled}	0.0108	0.0065	0.0208	0.0107
W^{random}	0.0043	0.0032	0.0125	0.0033
W^{sparse}	0.0103	0.0104	0.0185	0.0122
W^{1to1}	0.8923	0.9195	0.7186	0.8898
$W^{ensemble}$	0.0183	0.0020	0.0845	0.0446

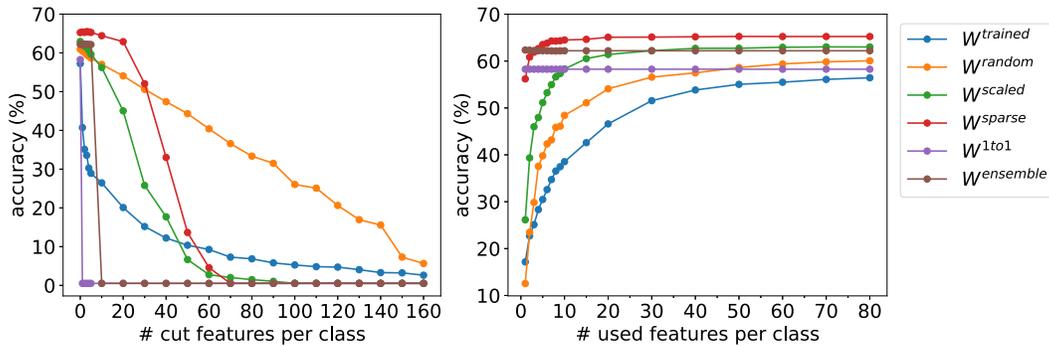


Figure 7: Neuron dependency (**left**) and expressivity (**right**) in a ResNet-50 with 2048 penultimate layer channels trained on **CUB-200** for different output layer designs, showing the change in accuracy on the test set. Best viewed in color.

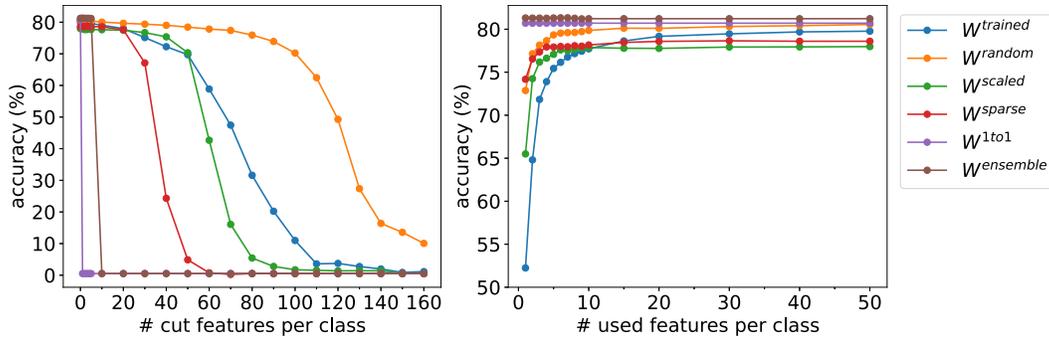


Figure 8: Neuron dependency (**left**) and expressivity (**right**), fine-tuned on **CUB-200** from a pre-trained IN model.

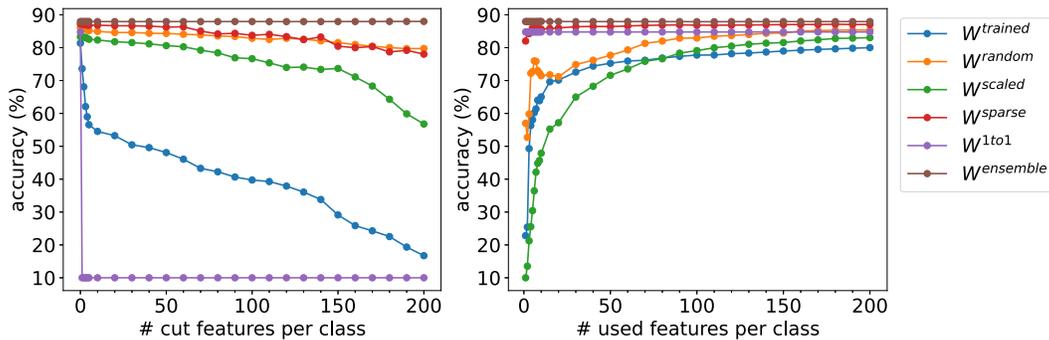


Figure 9: Neuron dependency (**left**) and expressivity (**right**), trained on **STL-10**.

Table 8: Comparing precision, recall and overall weight allocation between a trained and random output layer for different diabetic retinopathy severities in the APTOS dataset. The number of class instances are shown for the test fold. Gray background indicates fixed, random weight allocation.

		Recall	Precision	# instances	$\sum W_{:,k}$
$W^{trained}$	Neg.	0.95	0.95	361	1.9
	Mild	0.42	0.53	74	-1.2
	Mod.	0.78	0.63	200	-1.1
	ProL.	0.15	0.38	39	-6.1
	Sev.	0.36	0.41	59	-3.6
W^{random}	Neg.	0.97	0.95	361	-0.9
	Mild	0.39	0.53	74	0.3
	Mod.	0.83	0.69	200	1.2
	ProL.	0.33	0.46	39	-0.8
	Sev.	0.46	0.61	59	0.3

Table 9: Performance comparison of randomizing or scaling different parts of a ResNet-50. *Block* denotes fixing/scaling the last convolutional block in the network.

	STL-10	CUB-200
$W^{trained}$ (baseline)	81.36	57.18
$W^{trained}$ 3 blocks	84.21	63.99
W^{random} block	86.59	67.21
W^{scaled}	83.33	63.46
W^{scaled} network	85.66	58.40
W^{scaled} block	86.42	66.74

100 a single neuron reduces the absolute probability for the predicted class by $\sim 24\%$. Lowest neuron
 101 dependencies are observed in W^{random} , but the differences to other layers are subtle. The values for
 102 W^{1to1} can be interpreted as average confidence of the model, since there is a direct correspondence
 103 between conv channels and classes. We also include results for a pre-trained model for CUB-200. It
 104 is worth noting that in this setting, neuron dependencies are reduced for the standard output layer,
 105 which corroborates the link to overfitting and generalization. Note also that neuron dependencies tend
 106 to decrease with increasing number of heads for $W^{ensemble}$. For example, for STL-10, $H = 204$,
 107 so ablation of a single head does not reduce the output probabilities much. This is in contrast to
 108 CUB-200, where $H = 10$ leads to larger neuron dependencies.

109 Next, we include additional class-based neuron dependency/expressivity plots for above datasets in
 110 Fig. 7, 8 and 9 (see Fig. 4 in the main paper for C100; center crop is used for CUB). As expected, the
 111 accuracies in W^{1to1} and $W^{ensemble}$ drop to random performance after removing more neurons than
 112 heads available, and show (near) maximum performance already when using a single node per class.
 113 As for C100, accuracy drops slowest for W^{random} when ablating a large number of neurons. Both
 114 W^{sparse} and W^{scale} exhibit relatively small class-based neuron dependencies for small numbers
 115 of ablated neurons per class. In comparison, standard output layers perform worse in CUB-200
 116 and STL-10 even for small changes to the network. This changes for pre-trained networks, where
 117 $W^{trained}$ shows comparable class-based neuron dependency/expressivity to other output layers.

118 A.4 Weight allocation in output layers

119 The APTOS dataset consists of 3662 images, is imbalanced and can be subdivided by diabetic
 120 retinopathy (DR) severity: Negative DR (1805 images), Mild DR (370), Moderate DR (999), Prolifer-
 121 ative DR (193) and Severe DR (295). Table 8 shows performances for individual classes for a random
 122 cross-validation fold. Both recall and precision are improved especially in underrepresented classes.
 123 The last column in Table 8 lists aggregated weight values for individual classes. Interestingly, it indi-

124 cates a correlation between weight allocation and class sizes in $W^{trained}$, with the negative DR class
 125 being given the most weight. This suggests a bias in the output layer where over/under-represented
 126 classes are weighted more/less. This is prevented by design with randomization, but also inherently
 127 with 1-to-1 and ensemble layers where there is a single node per class.

128 A.5 Block scaling/randomization revisited

129 This section extends Sect. 5.5 from the main paper, which showed that fixing the last conv block,
 130 or scaling the last conv block next to the output layer, can regularize the network and improve
 131 performance. We want to answer two more questions and refer to Table 9 in the following. First, is
 132 randomizing the last conv block the main driver of regularization in small datasets, or is it rather a
 133 reduction of network capacity? We design another ResNet variant, which uses only 3 conv blocks
 134 followed by GAP, thus omitting the last conv block. We see improved results compared to the
 135 deeper baseline. However, randomizing the last conv block instead of omitting it again increases
 136 performance. Next, we are interested whether activation scaling benefits generalization when applied
 137 to every activation in the network (i.e. after each group of conv layer, BN and ReLU). Again, scaling
 138 the whole network performs better than $W^{trained}$. However, it also performs worse than W^{scaled} in
 139 CUB-200. Also, scaling only the last conv block outperforms the other variants in both datasets.

140 A.6 Visualization of heads in ensemble layers

141 Why does $W^{ensemble}$ show better performance than W^{1to1} ? After all, both consider the same
 142 channels from the previous conv layer. We conjecture that this is due to different initializations in
 143 the heads of $W^{ensemble}$ that might lead to varying local minima and thus complement each other.
 144 This is illustrated in Fig. 10 where activations for all 10 heads are maximized for two example
 145 classes in ResNet, fine-tuned on CUB-200. Although the learned visualizations of each head’s class
 146 node exhibit redundancies in color/structure, subtle differences are observable. For example, similar
 147 abstract patterns appear in all heads, but vary in orientation, rotation, or occur in different locations.

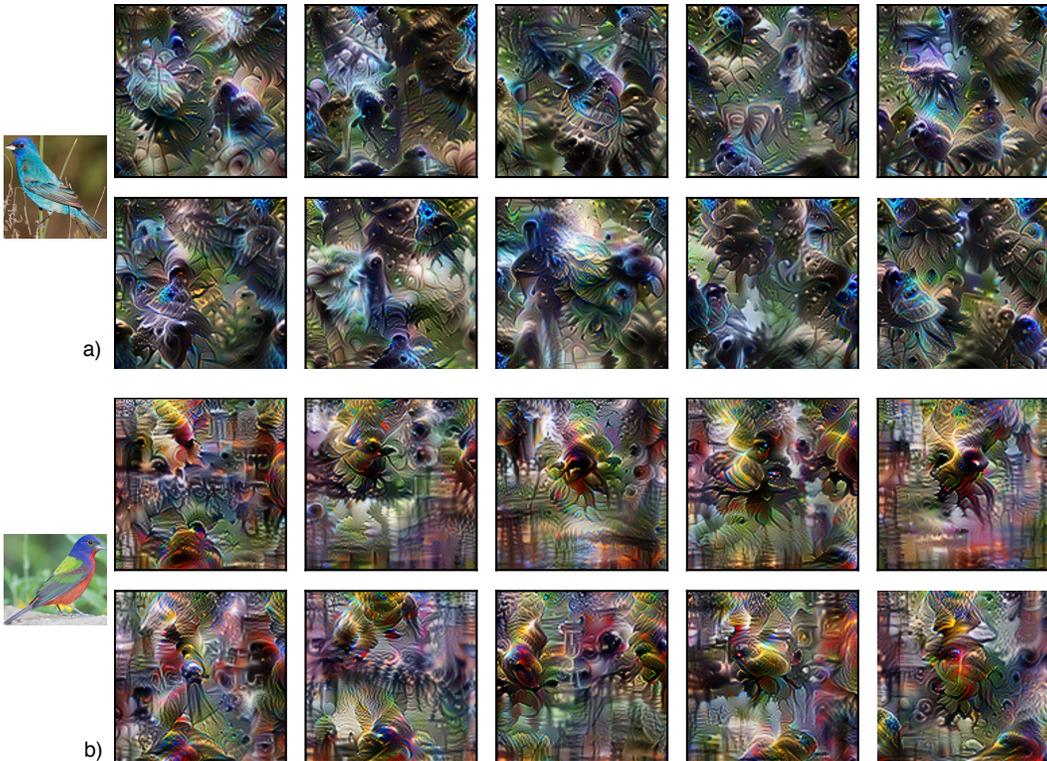


Figure 10: Feature maximization for class nodes across heads in $W^{ensemble}$ for the indigo (a) and painted bunting (b) with exemplars on the left side. We use Lucent [5] to create these visualizations.

References

- 148
- 149 [1] APTOS. Aptos 2019 blindness detection, 2019. URL [https://www.kaggle.com/c/](https://www.kaggle.com/c/aptos2019-blindness-detection)
150 [aptos2019-blindness-detection](https://www.kaggle.com/c/aptos2019-blindness-detection).
- 151 [2] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative
152 components with random forests. In *European Conference on Computer Vision*, 2014.
- 153 [3] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsuper-
154 vised feature learning. In *Proceedings of the fourteenth international conference on artificial*
155 *intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.
- 156 [4] Terrance DeVries and Graham W Taylor. Dataset augmentation in feature space. *arXiv preprint*
157 *arXiv:1702.05538*, 2017.
- 158 [5] Github. greentfrapp’s github repo, . URL <https://github.com/greentfrapp/lucent>.
- 159 [6] Github. Milesial’s github repo, . URL <https://github.com/milesial/Pytorch-UNet>.
- 160 [7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image
161 recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*,
162 pages 770–778, 2016.
- 163 [8] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected
164 convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern*
165 *recognition*, pages 4700–4708, 2017.
- 166 [9] A Emre Kavur, N Sinem Gezer, Mustafa Barış, Sinem Aslan, Pierre-Henri Conze, Vladimir
167 Groza, Duc Duy Pham, Soumick Chatterjee, Philipp Ernst, Savaş Özkan, et al. Chaos challenge-
168 combined (ct-mr) healthy abdominal organ segmentation. *Medical Image Analysis*, 69:101950,
169 2021.
- 170 [10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*
171 *arXiv:1412.6980*, 2014.
- 172 [11] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for
173 fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and*
174 *Recognition (3dRR-13)*, Sydney, Australia, 2013.
- 175 [12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images.
176 2009.
- 177 [13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep
178 convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
- 179 [14] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying
180 the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.
- 181 [15] PyTorch Hub. Pytorch. URL <https://pytorch.org/hub/research-models>.
- 182 [16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for
183 biomedical image segmentation. In *International Conference on Medical image computing and*
184 *computer-assisted intervention*, pages 234–241. Springer, 2015.
- 185 [17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng
186 Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei.
187 ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*
188 *(IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.
- 189 [18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale
190 image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- 191 [19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov.
192 Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine*
193 *learning research*, 15(1):1929–1958, 2014.

- 194 [20] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of
195 initialization and momentum in deep learning. In *International conference on machine learning*,
196 pages 1139–1147, 2013.
- 197 [21] Bram Van Ginneken, Tobias Heimann, and Martin Styner. 3d segmentation in the clinic: A
198 grand challenge. In *MICCAI Workshop on 3D Segmentation in the Clinic: A Grand Challenge*,
199 volume 1, pages 7–15, 2007.
- 200 [22] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011
201 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.
- 202 [23] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of
203 neural networks using dropconnect. In *International conference on machine learning*, pages
204 1058–1066. PMLR, 2013.
- 205 [24] Kaiyu Yang, Jacqueline Yau, Li Fei-Fei, Jia Deng, and Olga Russakovsky. A study of face
206 obfuscation in imagenet. *arXiv preprint arXiv:2103.06191*, 2021.