Artificial Neural Network Project

Arno Deceuninck

August 28, 2023

1 Settings used

When loading the data, the same transformations were used for all different models, which were the same as the pre-trained EfficientNet-B0 architecture from the Pytorch library. The batch size the data loaders will use is determined in the hyperparameter search.

1.1 Parameters same for all models

For all models, cross-entropy was used as loss functions, since this loss function is commonly used for classification models (also during the practical sessions).

All models use Adam as optimizer. It is often said that Adam converges faster, while SGD often converges to solutions which generalize more. However, some other papers mention fine-tuned Adam is always better than SGD [6]. Because there is still discussion, I mainly based the decision of the optimizer on the limited resources of my working station, so I have chosen Adam, since it converges faster. Another options would have been to add it to the hyperparameter search. This would however introduce a new parameter with two options to the search space, which would double the size of the search space.

The number of fully-connected layers is also the same for all our models. Only the classification head was changed from the EfficientNet-B0 architecture into a new fully-connected layer with the number of outputs I needed (15 for the scene classification task, 4 for the rotation pretext task and 2 for the perturbation pretext task).

EfficientNet-B0 uses one module 1 block, six module 2 blocks and eight module 3 blocks, which contain respectively one, two and two Conv2D layers [2]. This results in 29 Conv2D layers. Module 3 also contains a global average pooling layer, resulting in 8 global average pooling layers [2]. Besides this, there are also some other kinds of layers. However, all those layers (including the Conv2D and global average pooling layers), are not a fully connected layers according to the slides about convolutional neural networks, since the term fully-connected layer is there only mentioned in the part about dense fully-connected layers and online articles also see them as two separate kinds of layers [3, 5, 1, 7] (even though they are sometimes named 'fc' in the model). This means the only fully-connected layer for all our models is in the classifier part of the models, which is only 1 fully-connected layer.

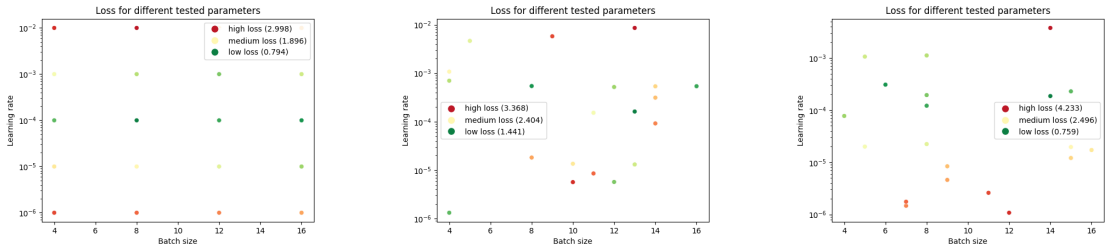
I could change this number of fully-connected layers in the classification part, but did not do that, since we are going from a model that classifies images into 1000 different scenes (according to the number of output nodes of the classifier layer) to only 15 different scenes. Adding more layers would make the model more complex than necessarily (with the risk of overfitting more).

1.2 Parameters determined by search algorithms

The learning rate and batch size was determined by automatically searching the search space. The number of epochs during this search was determined using an early stopping algorithm (with a maximum of 25 epochs). If the validation loss of two consecutive epochs was worse than the validation loss of the epoch before those two, the highest loss gets returned as final output for this parameter set test.

The maximum number of epochs was set to 25, which should be large enough to get an idea of what the best parameters are. For the final outputted parameter set, the exact epoch number is determined later (to prevent over- and underfitting) by running the models with the best parameter combinations for 40 epochs (so the 25 epochs were only for searching hyperparameters). How I achieved those finally taken epoch number is described in section 2.

The search space consisted of batch size 4 to 16 and a learning rate between 10^{-6} and 10^{-2}. Higher batch sizes were not possible because of limited GPU memory size on my laptop. There are different methods for searching hyperparameter values in a given search space. I tried using grid search, random search and bayesian optimization. Since grid search goes over all possible values, but the learning rate is a continious number and we want the search to be done in a reasonable amount of time, the tested values for the learning rate were $10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}$ and 10^{-2} and the tested values for the batch size were 4, 8, 12 and 16. Random search and the bayesian optimization could both support continuous values, so we only made the learning rate discrete for the grid search. The values of the learning rate were log-uniformly selected (between 10^{-6} and 10^{-2}) and the batch size was a random integer, uniformly selected between 4 and 16. Both random search and bayesian optimization got 20 trials (which are 20 guesses of parameter value combinations).



(a) Grid search (b) Random search (c) Bayesian optimization

Figure 1: Comparison of tested values for different hyperparameter search algorithms on a smaller dataset (containing 10% of the images of the original dataset). All algorithms tested 20 different parameter combinations. Bayesian optimization (1c) resulted in the lowest validation loss (0.759), followed by grid search (0.794) and random search had the worst lowest loss (1.441).

Those three algorithms were tested on a smaller version on the dataset (to demonstrate the difference), which consisted of 10% of all images, randomly sampled (in both the train and test dataset). Figure 1 shows the difference between the hyperparameter search algorithms. Grid search only searches for the given values and tries all different combinations for those values. Random search randomly selects values from our defined ranges, not taking into account which values were already tested and not focusing on ranges that yielded better results in the past. Bayesian optimization solves this by building a probability model of the objective function and using this model to select the most promising hyperparameters to evaluate in the true objective function by taking the exploitation-exploration dilemma into account [4].

Since bayesian optimization performed best, this one was used for finding the hyperparameters for our models on the complete parameters, using the same search space as earlier mentioned for the learning rate. The search space of the batch size was set to uniformly search between batch size 8 and 16, to reduce some time of the long searching times (since lower batch sizes than this use a GPU less optimally) and keep the focus mainly on the learning rate.

An overview of the different settings used for training each model is given in table 1. The hyperparameter combination for the pertrurbation pretext task were manually selected. The reason for this is explained in section 3.

Model	Learning rate	Batch size	Number of epochs
Scene classification (fully-supervised)	$2.405 * 10^{-5}$	16	40
Rotation classification task	$1.055 * 10^{-4}$	14	40
Scene classification with rotation pretext	$1.320 * 10^{-3}$	16	40
Perturbation classification task	$1.000 * 10^{-5}$	15	40
Scene classification with perturbation pretext	$2.018 * 10^{-4}$	13	40

Table 1: Settings used for training the different models. The optimizer was always Adam and the number of fully-connected layers was always 1. The number of epochs is the number of epochs for the finally selected hyperparameter combination, while searching hyperparameters, the models were ran for 25 epochs.

Model	Training accuracy	Validation accuracy	Number of epochs
Scene classification (fully-supervised)	100.00%	93.74%	32
Rotation classification task	99.47%	92.56%	15
Scene classification with rotation pretext	99.40%	84.19%	27
Perturbation classification task	100.00%	100.00%	10
Scene classification with perturbation pretext	95.60%	81.57%	37

Table 2: Training and classification accuracies of the different models. The number of epochs is the number of epochs after which the model is selected (which is one plus the epoch number, since the epoch numbers start counting at 0)

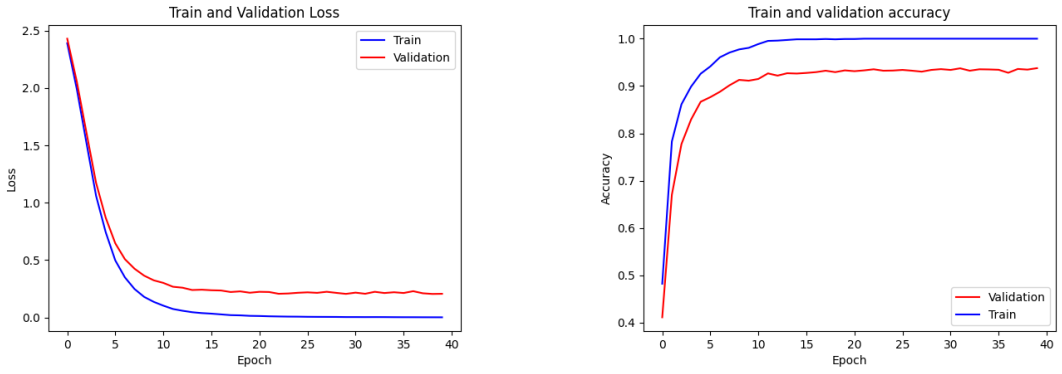
2 Classification accuracies

Table 2 contains an overview of the different classification accuracies of the models. The epoch number from where the trained model was selected is already explained in section 2.1.

2.1 Number of epochs

Once the best hyperparameter set was found accoording to a run of the bayesian optimization algorithm, the actual number of epochs needed to be determined (and not just the number of epochs determined by the early stopping algorithm).

To determine the number of epochs, the models were trained for 40 epochs (a large number, so a good model is clearly visible before this if it exists) and based on the train and validation loss improvements, the actual number of epochs is determined.



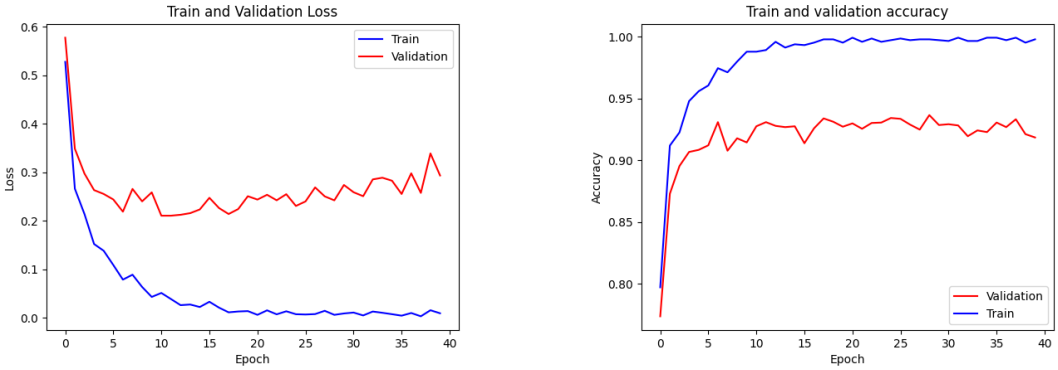
(a) Loss

(b) Accuracy

Figure 2: Loss and accuracy for different epochs for the supervised model with optimized parameters (learning rate $2.405 * 10^{-5}$ and batch size 16)

Figure 2 shows the loss and accuracy for different epochs of the supervised model with optimized hyperparameters. The first increase in the validation loss is after epoch 14. However, the validation

loss keeps decreasing in the runs after this. Both graphs have a nice curve, going to an almost horizontal line at the end. The finally used number of epochs used is 32, since after this, the validation loss gets a bit worse for a few epochs and there is not much improvement afterwards.



(a) Loss

(b) Accuracy

Figure 3: Loss and accuracy for different epochs for the rotation pretext task with optimized parameters (learning rate $1.673 * 10^{-4}$ and batch size 12). The best validation accuracy is 93.77% at epoch 28 and the best validation loss is 0.2105 at epoch 10.

Figure 3 shows the loss and accuracy for different epochs of the model of the rotation pretext task after optimizing the hyperparameters (only the rotation part, not further trained to classify images). This graph however, showed some overfitting happening, starting somewhere around epoch 12, the validation loss keeps increasing, while the training loss is still decreasing. Besides this, the graph also seems to go up and down a lot, which might indicate a learning rate which is too high, even though it had the lowest loss from our hyperparameter search. Because of this, I also took a look at other outputs of the hyperparameter search with low losses (the other green dots shown in figure 4). Figure 5 shows the loss and accuracy for the finally used hyperparameters, which overfits less and has a similar loss. The loss is best at epoch 14 (so after 15 epochs, since the epochs start counting at 0), so this I will use 15 epochs for the rotation model, from which we can train the model further for the image classification task.

Figure 6 shows the graphs for the image classification task after a rotation pretext. The final model was selected here after 27 epochs, which had both the validation accuracy and the best validation loss of all epochs.

In section 3, I will give more information on how the number of epochs for the perturbation model is determined.

3 Perturbation pretext model at different epochs

Initially, I selected the hyperparameters for the perturbation pretext model using bayesian optimisation search. Figure 7 shows the loss and accuracy graphs of the best hyperparameters accoording to this search.

With this, I selected the model after 7 epochs (since the validation accuracy was already perfect after the second epoch and afer 8 epochs was the first validation loss increase) and the validation loss I obtained was very low (0.00001443) and a validation accuracy of 100%.

However, when freezing the feature part and finding the right hyperparameters when training the classifier part for the scene classification task, the performance of the image classification task was a

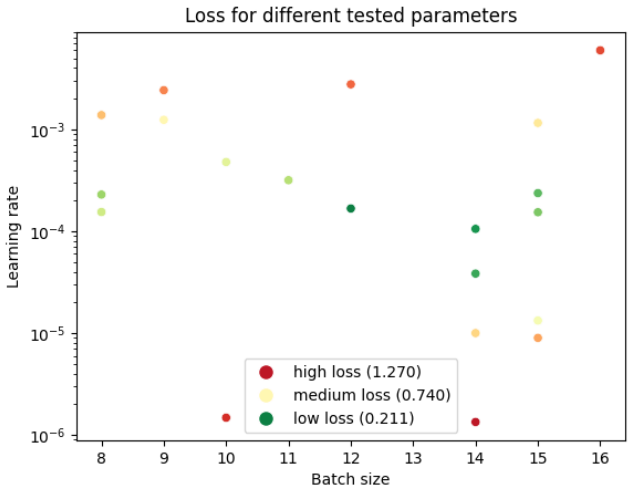
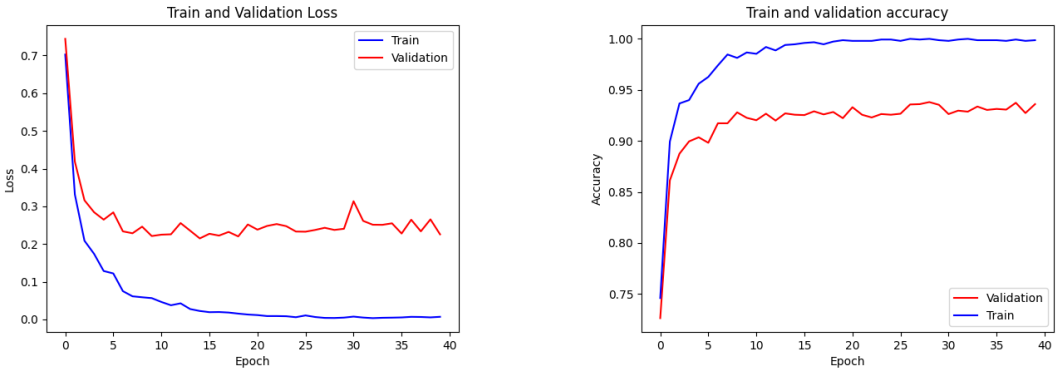


Figure 4: Tested hyperparameters for the rotation pretext task using bayesian optimization



(a) Loss

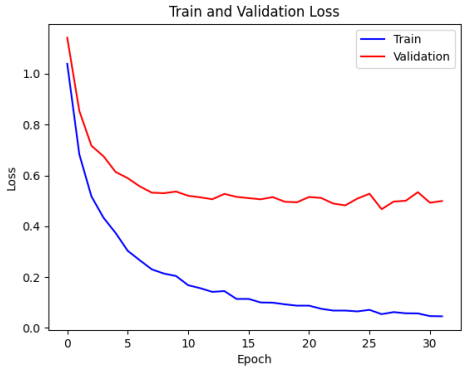
(b) Accuracy

Figure 5: Loss and accuracy for different epochs for the rotation pretext task with parameters similar to the optimized ones (learning rate $1.055 * 10^{-4}$ and batch size 14). The best validation accuracy is 93.80% at epoch 28 and the best validation loss is 0.2149 at epoch 14.

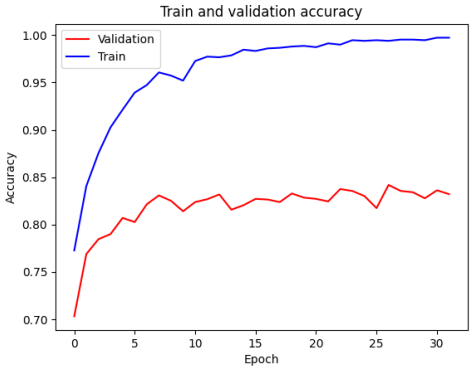
lot worse, reaching only a train accuracy of 46.20% and validation accuracy of 43.55%, even though the hyperparameter search space did not change from the previous methods.

I also tried training the scene classifier based on the model at some lower epochs, which also had a 100% train and validation accuracy, but a higher loss. After 5 epochs, the validation loss of the perturbation task was 0.00003849, but the validation loss of the scene classification task was only 1.846127 (with a validation accuracy of 43,79%). After two epochs, we also already had a train and validation accuracy of 100% and a validation loss of 0.00021150 for the pretext task. Searching hyperparameters for the scene classification task starting from this epoch also led to a validation accuracy of only 41.14% (and a validation loss of 1.890116).

When doing some manual hyperparameter tests for the perturbation task, I noticed that with a learning rate of 10^{-5}, I could also obtain a validation accuracy of 100% with a loss graph with a better curve, which is shown in figure 8. The previous curve went very fast down after the first epoch and stayed low then, while this one lowers gradually. The train loss after 10 epochs here was however only 0.009403, which is higher than the previous model (which makes sense, since our hyperparameter

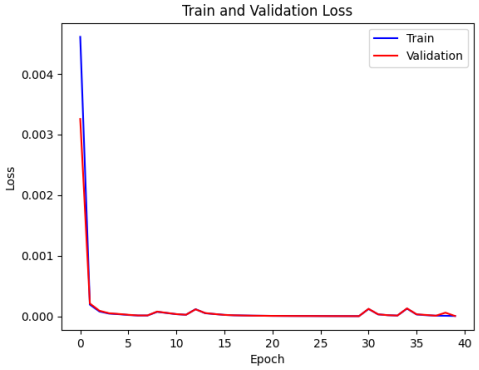


(a) Loss

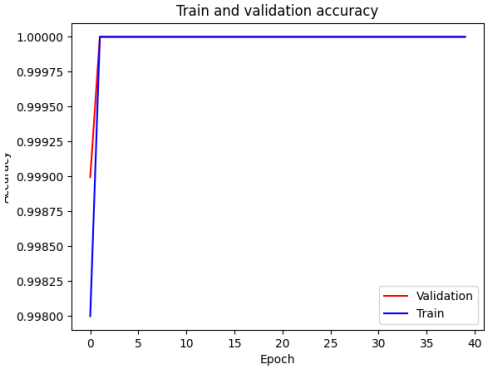


(b) Accuracy

Figure 6: Loss and accuracy for different epochs for the image classification task after a rotation pretext task



(a) Loss



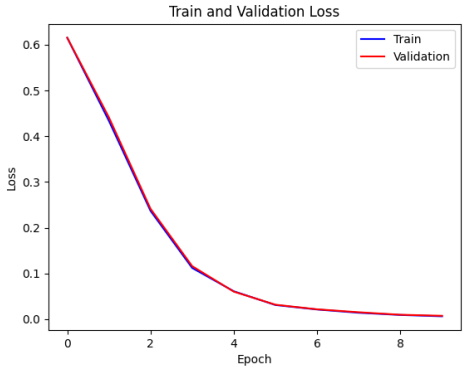
(b) Accuracy

Figure 7: Loss and accuracy for different epochs for the perturbation pretext task for the best hyperparameters found with bayesian optimization (learning rate of $1.830 * 10^{-3}$ and batch size 15)

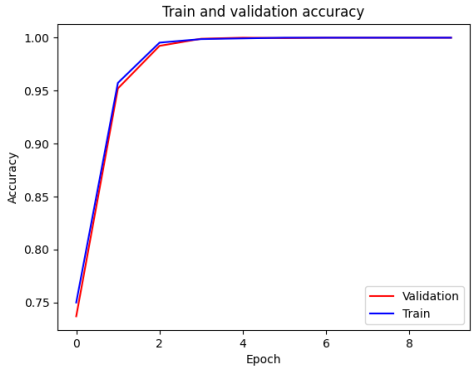
optimization algorithm tries to minimize the loss as much as possible). Running the hyperparameter search to train for the scene classification task based on this model gave significantly better results on the image classification task than using the perturbation model with a lower loss (even though the perturbation classification loss was worse).

For this model, I evaluated the classification models trained based on different epochs of the pretext task (10, 6 and 3). The different losses and accuracies are shown in table 3. There is not much difference in the accuracies, but both the train and validation accuracy decreases for models trained from higher epochs of the pretext model, even though the validation loss of the perturbation task at 3 epochs is 0.4756, while it lowers to 0.00940 after 10 epochs). The model where the perturbation task has trained for 3 epochs has the highest train and validation accuracy, while the model where the perturbation task has trained for 10 epochs has the lowest train and validation accuracy. For all three of those models are the accuracies a lot better than when using our initial perturbation model, which had a way lower loss on the perturbation task (which is also what you would get after training for a lot more epochs).

The better the model has adapted to the perturbation task, the worse the performance on the scene classification task seems to be. This is probably because the perturbation task does not create features that are relevant for the scene classification task. This makes sense, since predicting the color of a square with a solid color in a random location does not need special features from an image. It looks



(a) Loss



(b) Accuracy

Figure 8: Loss and accuracy for different epochs for the perturbation pretext task when training with a learning rate of 10^{-5} and a batch size of 15

Perturbation task epochs	**Train accuracy**	**Validation accuracy**
3	98.93%	84.46%
6	99.27%	82.65%
10	95.60%	81.57%

Table 3: Train and validation accuracies on the scene classification task when training from different epochs of the perturbation task

like it is training the features too much for the perturbation task, which decreases the performance on the scene classification task.

4 Performance comparison

The fully-supervised model performs best, with a validation accuracy of 93.74%. This might be because model was entirely trained for this specific task and the EfficientNet model we started from already had relevant features for this task. For example the "bedroom" label from our scene classification task might be similar to the "day bed" label of the ImageNet dataset our weights our initial weights were trained on, the "street" label could use features similar to the "street sign" label, ...

However, the ImageNet dataset does not have scenes specifically as labels, so some form of pretext task might be useful for extracting more relevant features for this specific classification task. From the two pretext tasks, the scene classification with rotation pretext resulted in a better validation accuracy (of 84.19%), which is slightly better than the validation accuracy of the perturbation pretext task (81.57 %). This makes sense, since for determing the rotation applied to an image, you need to be better aware of the actual context (by training more relevant features) than to just recognize a square of pixels in the same, solid color (which requires less contextual information).

Self-supervised methods can certainly be useful for generating artificial data when the training dataset is too small. However, you do not need relevant features to perform the perturbation task, so I would only consider the rotation pretext useful from those two. The rotation pretext requires more contextual information than the perturbation pretext. However, since the supervised performance is a lot better (probably because there was already enough data in our training set), the pretext task were not useful for this classification task with the given dataset, but might certainly be useful in other cases.

5 Preventing overfitting

To prevent underfitting and overfitting, the loss on both the train and validation set was monitored after each epoch. To prevent underfitting, each model was trained for 40 epochs. The early stopping mechanism was in place to prevent overfitting during the hyperparameter search (and not lose too many time on models that were overfitting). More details of this early stopping mechanism were given in section 2.1.

After the optimal hyperparameters were found accoording to my search algorithm, I always took a look at the train and validation loss and accuracy graph. For example, for the rotation classification task, the model was overfitting in figure 3, which is why I also looked at other outputs of the hyperparameter search and changed the parameters to try to prevent this, as I did to obtain the finally selected hyperparameters from figure 5.

Besides this, the classifier part also consisted of a dropout layer for all mode (in front of the fully-connected layer) with a dropout rate of 20% to prevent overfitting.

6 Explanation

To focus on the differences between our models, all explanations of the same block are in the same table. Table 4 shows the output after the first block, table 5 after the second block and table 6 after the last block. All those images are correctly classified.

The outputs of the first and second blocks seem similar for all three models. The first block focuses on some edges, like the contrast between sky and mountain, and sometimes already larger bodies, like the mountains and part of the sea for the "coast" label and the windows of the first "suburb". The model of the perturbation pretext seems to focus less intensely, e.g. the coast is less highlighted and the windows from the second suburb are not highlighted. The sky of first suburb is an exception for this, since it is only highlighted in the perturbation pretext task and not in the others.

In the second block, this sky is not highlighted anymore in the perturbation pretext task, but it is highlighted in both the rotation pretext and fully-supervised model. The blocks still look similar for the different model, again with the perturbation pretext having less intense areas. This block selects more areas in the scene, like the mountain, the sky or the water in the forest.

The final layer contains most of the differences. The supervised model focuses mainly at things in the center of the images and selects relevant objects, like the actual trees in the forest, the mountain or the suburbs. The model trained with a rotation pretext focuses mainly on areas at the borders of the images (not in the center), which might be more useful for knowing the rotation. It often selects multiple objects, e.g. in the coast it selects both a mountain and the water, since this can help rotating the image (since the bottom should be on top). The pattern in the model with the perturbation pretext is less clear In some images, it looks like their is a strong focus in a specific area. For example in the mountain image is the lower left corner strongly focused, possibly because it is a darker part of the image and the perturbation task focused on detecting white or black squares. Also for the forest and coast image, the main focus is on a small area.

7 Interpretation

Table 7 shows the indices of the selected filters per model (the five with the highest sum of weights of the convolutional layer in the last features block). The top 3 has the same index for all models.

In order to create inversions from this layer, the code used during the practical sessions was adapted to maximize the mean value of the weights for all filters of given layer (minimize the opposite of this) and to also support the initial learning rate as input parameter. Note that I used the mean instead of the sum, but this does not make a difference, since each filter contains the same amount of weights and this keeps the magnitude of the loss similar to the weights.

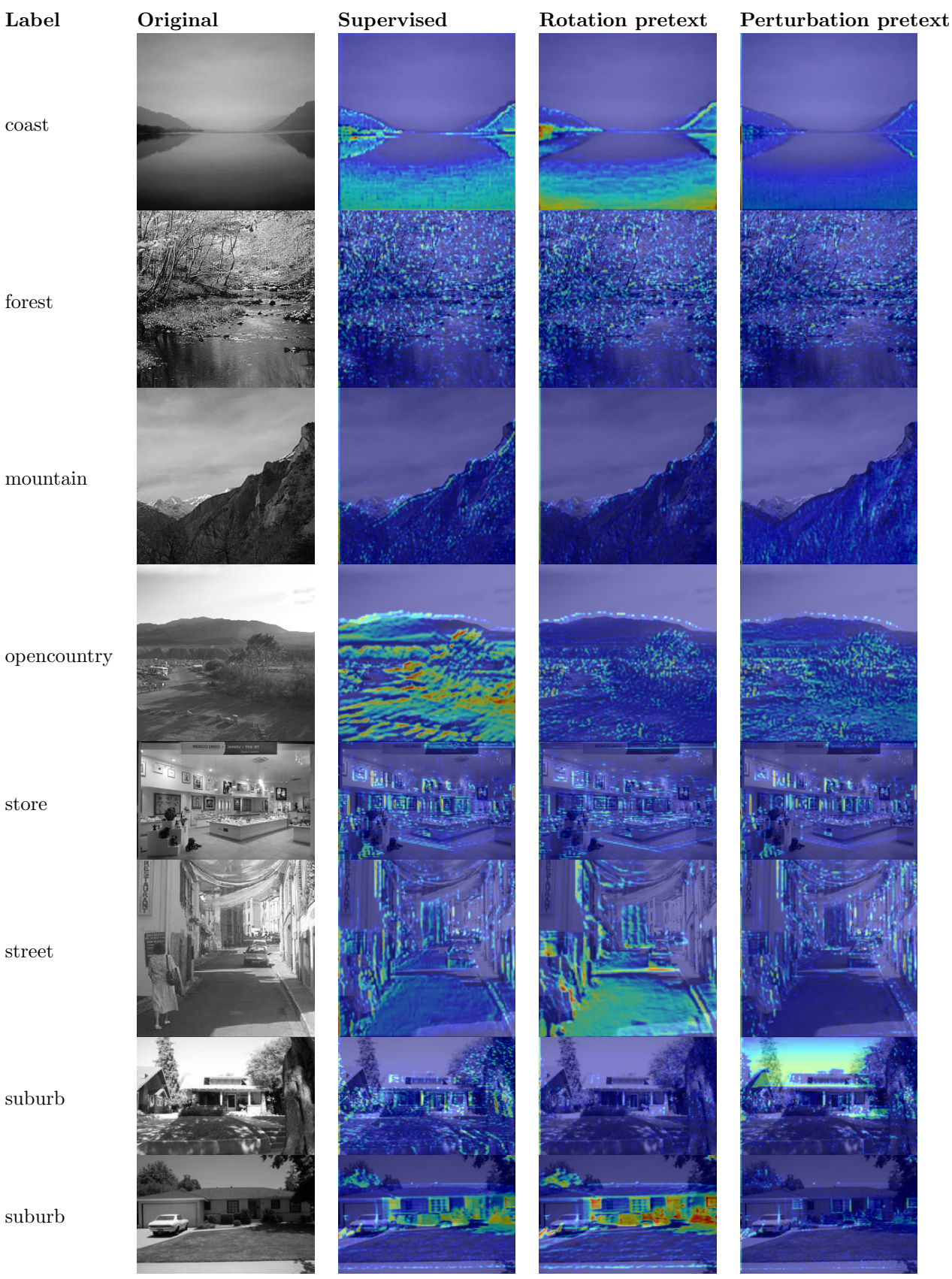


Table 4: Explanations of the first block

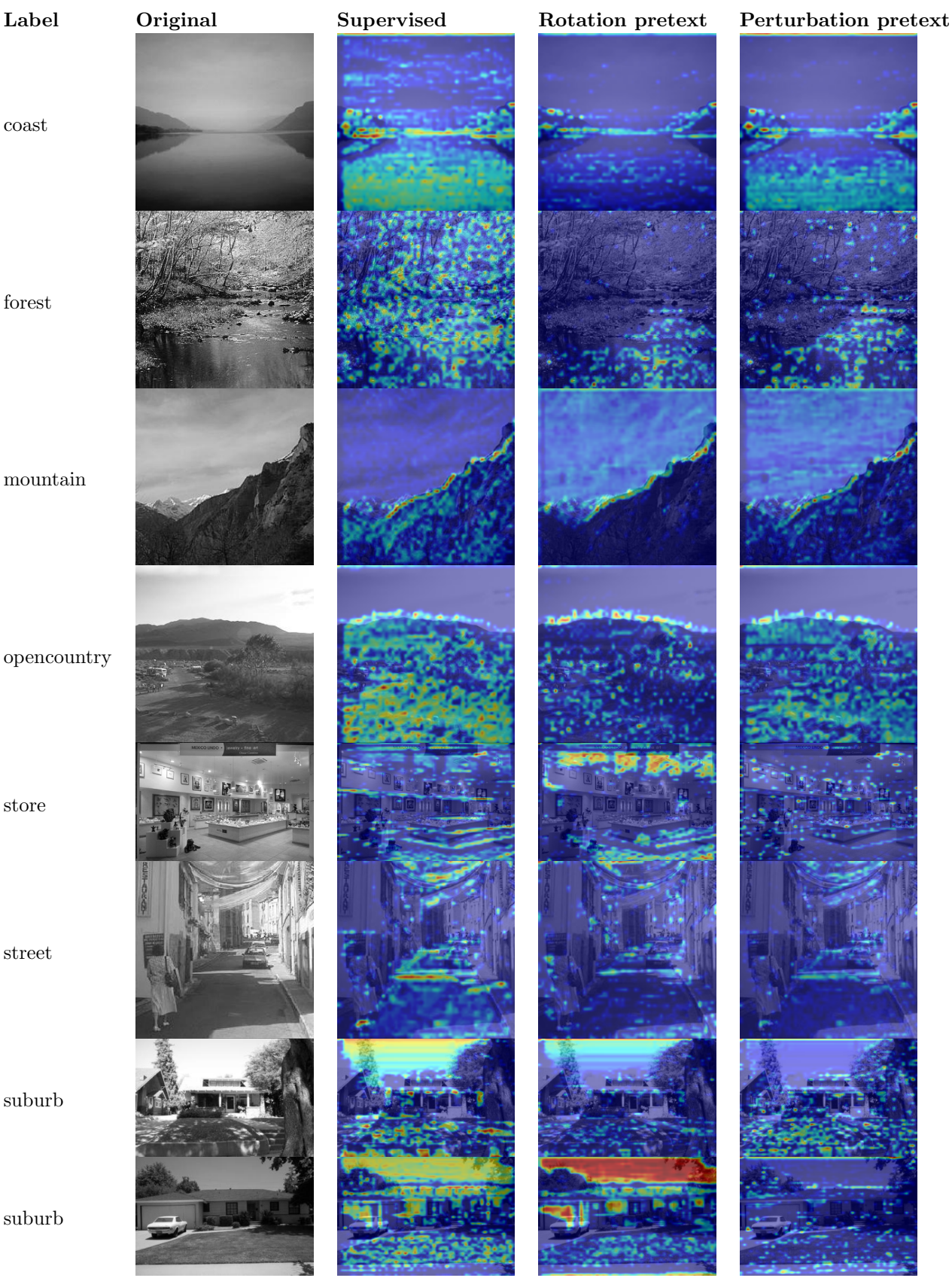


Table 5: Explanations of the second block

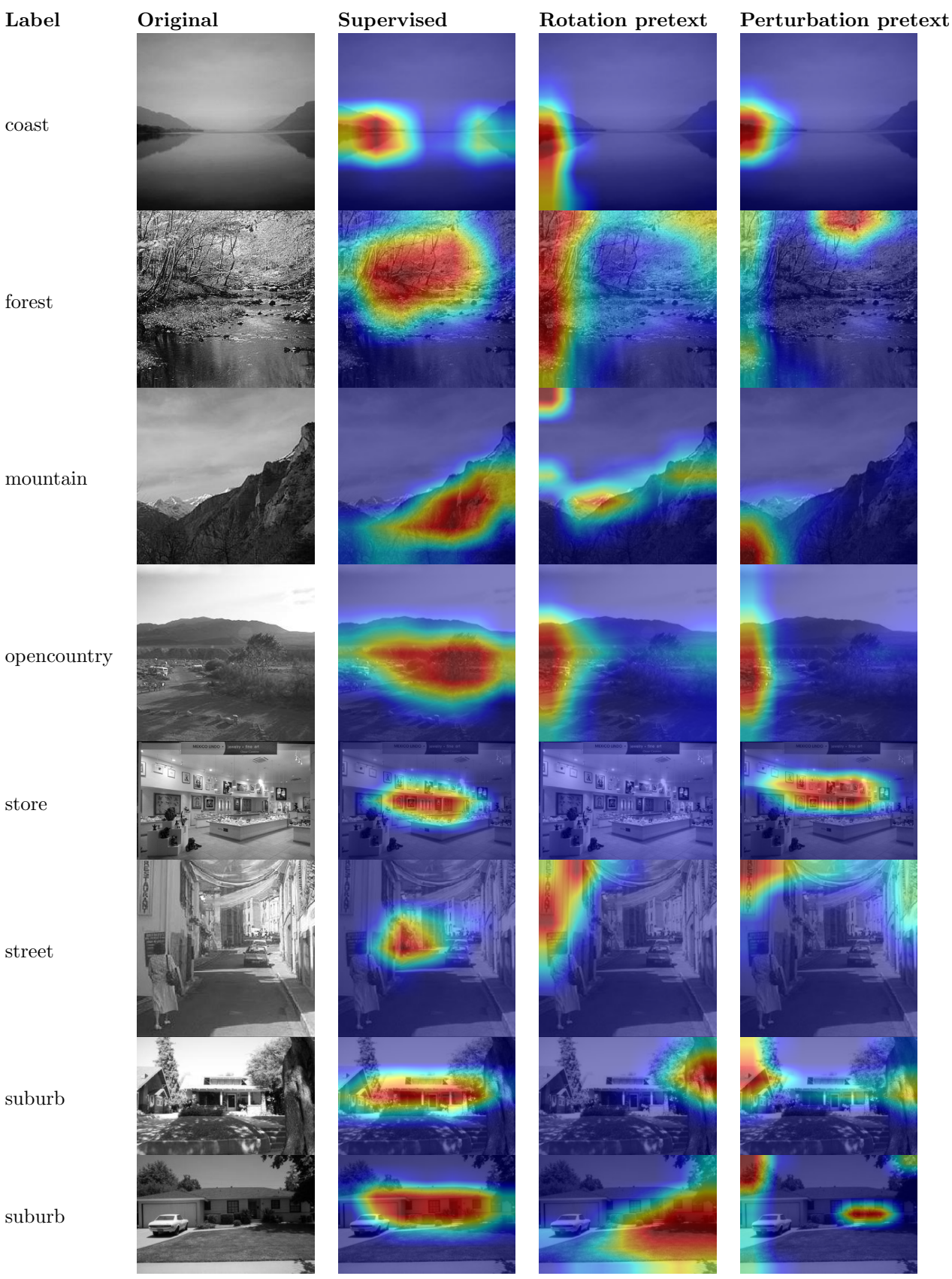


Table 6: Explanations of the last block

Model name	Highest sum	Lowest sum	Filter indices
fully-supervised	4.1870	-3.240	70, 1250, 944, 833, 768
rotation pretext	4.0137	-3.274	70, 1250, 944, 571, 330
perturbation pretext	4.0604	-3.341	70, 1250, 944, 768, 833

Table 7: Indices of the filters with the highest sumation of weights of the convolutional layer in the latest block. The highest and lowest sum are of all filters (and not only the top 5). The filter with index 70 has for all three models the highest sum of weights.

After this, bayesian optimization was ran to search hyperparameters that worked. I initially started the search space to find a blur frequency between 2 and 10, blur radius between 0.5 and 2, weight decay log-uniformly between 10^{-6} and 10^{2}, clipping value between 0.01 and 0.2 and the initial learning rate also between 10^{-6} and 10^{2}. However, after reading the inversion paper ([8]) to see how the variables could be optimized more, I only focused on the weight decay and initial learning rate, since the other parameters might actually result in a higher loss (but better understandable by humans).

So I fixed the values of the blur frequency to 4, blur radius to 1 and clipping value to 0.1 (which were the default values for those parameters in the code) and only searched for the best combination of the initial learning rate and weight decay, both between 10^{-6} and 10^{2}. This kept giving random images as results (e.g. image 9). The high losses from the hyperparameter search were mainly caused by some random peaks instead of a good loss curve. As an attempt to solve this, I changed the loss to the median of the loss of last 10 iterations instead of only the loss of the last iteration of the inversion algorithm. Since bayesian optimization might focus too much on peaks, I also tried running grid search.

Figure 10 shows the loss functions for different learning rate and weight decay combinations. A learning rate of 0.1 and weight decay of 1 gives the best loss function, but only has -12.629 as loss and gives figure 11. The lowest loss found using the grid search was -7545.226 and is shown in figure 12. This is however the result of some random peak in the loss function.

Since the images that resulted from a random peak resulted in more randomly looking images, I used models that had a good loss curve for the image generation of all models, even though the loss is lower than the random images. Table 8 contains all generated images, for the top-5 filters of each of the models. However, they look all very similar like a gray-brown image with some random pattern on them.

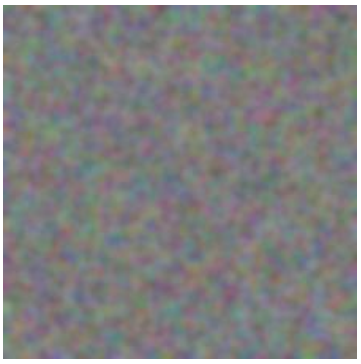


Figure 9: Random pixels as result of the image generation

Table 8: Generated model inversions (in the order of the list in table 7, the top image is for filter 70)

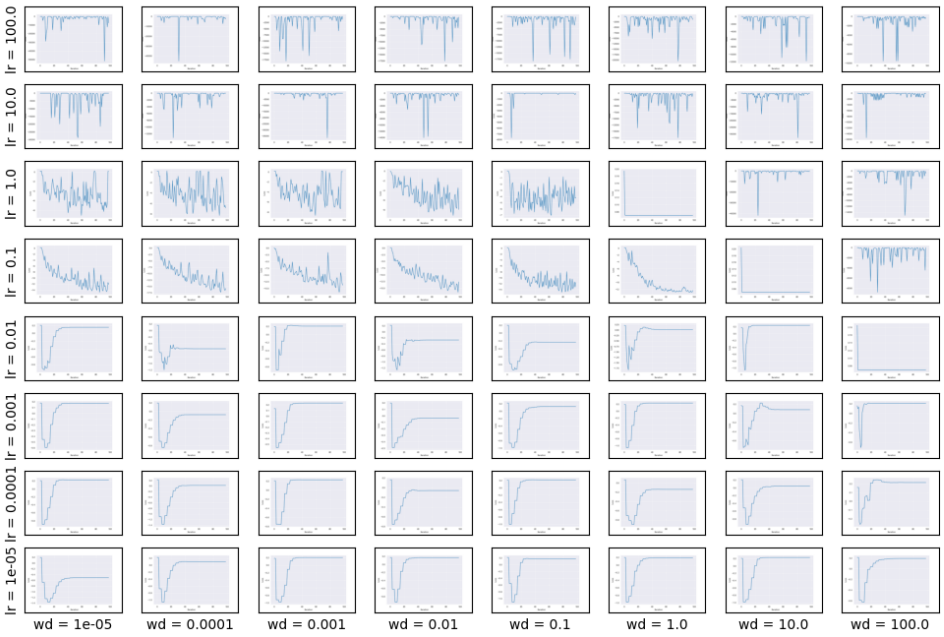
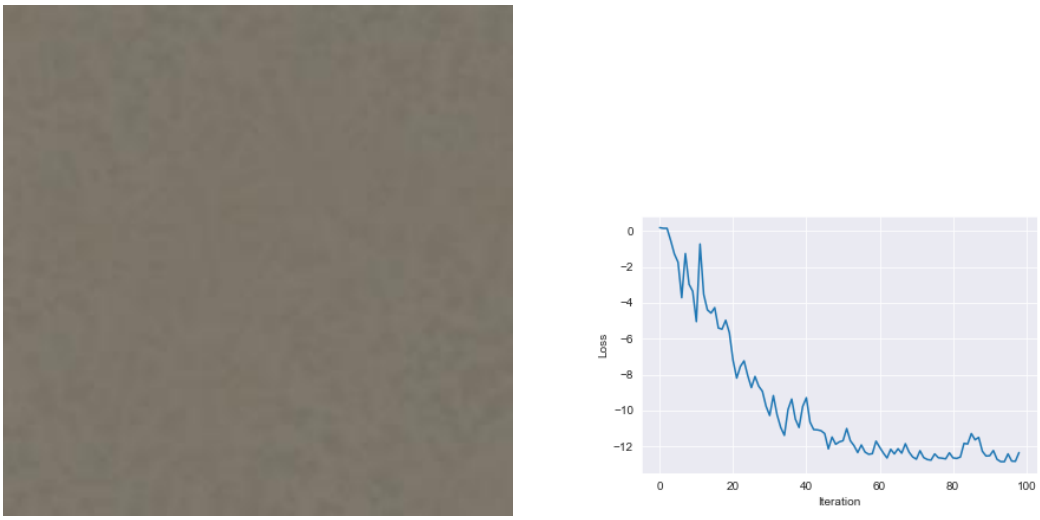


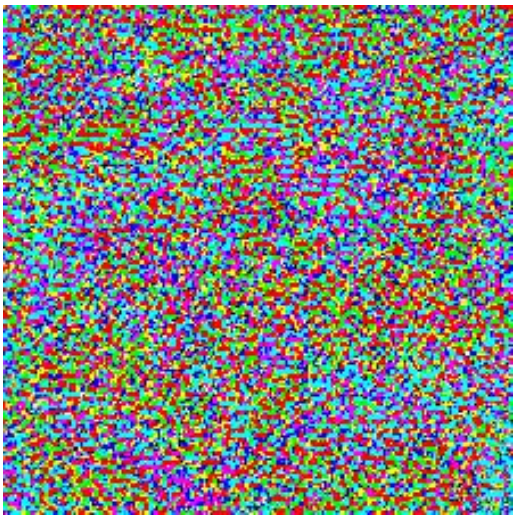
Figure 10: Loss function for different initial learning rates and weight decays from a grid search for filter 70 of the supervised model



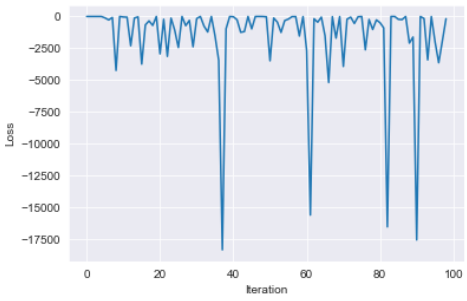
(a) Generated image

(b) Loss function

Figure 11: Generated image (iteration 150) and loss plot for filter 70 of the supervised model for learning rate 0.1 and weight decay 1 giving a loss of -12.344



(a) Generated image



(b) Loss function

Figure 12: Generated image (iteration 150) and loss plot for filter 70 of the supervised model for learning rate 100 and weight decay 0.1 giving a loss of -7545.226

References

[1] fully connected layer vs. convolutional layer: explained, 2022.

[2] V. Agarwal. Complete architectural details of all efficientnet models, May 2020.

[3] W. are. What are the advantages of fc layers over conv layers?, Sep 2020.

[4] W. Koehrsen. A conceptual explanation of bayesian hyperparameter optimization for machine learning, Jun 2018.

[5] P. Mahajan. Fully connected vs convolutional neural networks - the startup - medium, Oct 2020.

[6] S. Park. A 2021 guide to improving cnns-optimizers: Adam vs sgd, Jun 2021.

[7] D. Unzueta. Convolutional layers vs fully connected layers - towards data science, Nov 2021.

[8] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.