Artificial Neural Networks
Self-Supervised Scene Classification

Maksim Karnaukh
1st year Masters at Department of Computer Science
University of Antwerp
Antwerp, Belgium
Email: maksim.karnaukh@student.uantwerpen.be

1. Introduction

In this project, we look into some self-supervised learning techniques applied to scene classification, comparing their efficacy against traditional fully-supervised methods. The motivation behind exploring self-supervised learning lies in its potential to alleviate the burden of annotated data dependency and the associated costs.

This project focuses on training and evaluating two distinct approaches: fully-supervised, which leverages pre-labeled data for training a convolutional neural network (CNN), and two self-supervised pipelines. These pipelines introduce pretext tasks, i.e. predicting Gaussian Blur kernel sizes and classifying image perturbations, as initial learning steps. The underlying hypothesis might be that these pretext tasks enable the network to learn generalized feature representations that can subsequently be fine-tuned for the scene classification task.

To facilitate experimentation, we utilize the 15-Scene Dataset, encompassing a diverse range of indoor and outdoor environments, with 15 different classes.

By investigating both supervised and self-supervised learning strategies within the context of scene classification, this project aims to provide insights into the relative strengths and trade-offs of these approaches in handling complex visual data tasks.

2. Data Preparation

This dataset is partitioned into training and test sets (80-20 ratio), enabling rigorous evaluation of model performance across different methodologies.

For the Gaussian Blurring pretext task, the images in the dataset are processed by applying Gaussian blur filters with five different kernel sizes: 5x5, 9x9, 13x13, 17x17, and 21x21. This augmentation generates five variants of each image, effectively expanding the dataset. Each image variant is labeled according to the kernel size of the applied Gaussian blur.

For the Black and White Perturbation pretext task, each image undergoes a perturbation process where a random 10x10 square region within the image is selected and modified. The perturbation involves setting the pixels within this square to either zero (black) or 255 (white), resulting in two variants of each image. Each image variant is labeled as either ”black” or ”white” perturbation based on the modification applied. The original images get excluded.

3. Methodology

In this section, we outline the main structure employed for training and evaluating the models using both fully-supervised and self-supervised learning techniques. The core of our approach involves leveraging a pre-trained EfficientNet-B0 model from the PyTorch library, and fine-tuning it for scene classification.

3.1. Fully Supervised Learning Scheme

For all learning schemes, we use a pre-trained EfficientNet-B0 model, which has been trained on the ImageNet dataset. The following steps were undertaken:

Model Preparation:
We load the pre-trained EfficientNet-B0 model. We ensure all layers in both the feature extraction and classifier parts are trainable. We apply the same image transformation function (in data loading) used during the pre-training phase, accessed via torchvision.models and then EfficientNet_B0_Weights.IMAGENET1K_V1.transforms().

Training Process:
We train the model on the 15-Scene dataset using the training set. We monitor performance using the validation set (throughout this paper, we might use validation and test interchangeable, and while there is in fact a difference between the two, we only use two sets: train and test, which we sometimes thus refer to as validation). This is also a simplified explanation, since we will use hyper-parameter tuning, but this will be covered in the next section on experimental setup.

3.2. Self-supervised Learning Schemes

In the self-supervised learning approach, we design two independent pipelines, each comprising a pretext task followed by the main (downstream) task of scene classification.

3.2.1. Pipeline 1: Gaussian Blurring Pretext Task.

- **Pretext Task - Gaussian Blurring:**

We first modify the pre-trained EfficientNet-B0 by replacing the original classifier with a new classifier that distinguishes between five different Gaussian blur kernel sizes (5x5, 9x9, 13x13, 17x17, 21x21). We train this modified model to predict the kernel size of the Gaussian blur applied to the input images. We also ensure that all layers in both the feature extraction and classifier parts are trainable during this phase.

- **Downstream Task - Scene Classification:**

Here, we replace the Gaussian blur classifier with the original scene classification head from EfficientNet-B0. It's important that we freeze the feature extraction layers, making only the classifier part trainable. We fine-tune the model on the 15-Scene dataset, training the classifier to categorize scenes based on the feature representations learned during the pretext task.

3.2.2. Pipeline 2: Black and White Perturbation Pretext Task.

- **Pretext Task - Black and White Perturbation:**

This is analogous to the parts before, we modify the pre-trained EfficientNet-B0 by replacing the original classifier with a new classifier that distinguishes between black and white perturbations. We train this modified model to classify images based on whether a 10x10 square region is perturbed to black or white.

- **Downstream Task - Scene Classification:**

Just like before, we replace the perturbation classifier with the original scene classification head from EfficientNet-B0. We freeze the feature extraction layers, making only the classifier part trainable and fine-tune the model on the 15-Scene dataset, training the classifier to categorize scenes based on the feature representations learned during the pretext task.

3.3. Model Comparison

To compare the effectiveness of the supervised and self-supervised learning schemes, we evaluate each model's classification accuracy on the test set and analyze the differences in performance between the fully-supervised model and the models trained via the self-supervised pipelines by using, for comparison purposes, the same number of epochs as we used in the fully-supervised method.

This way, we can assess the impact of the pretext tasks on the feature representations and their transferability to the downstream scene classification task.

4. Experimental Setup

In this section, we mainly discuss the methodology employed for hyper-parameter tuning, the parameter grid used, and the strategies implemented to avoid overfitting.

4.1. Hyperparameter Tuning

To optimize the performance of our models, we perform hyper-parameter tuning using a grid search approach. Grid search involves systematically exploring a predefined set of hyperparameters to identify the combination that yields the best performance on the validation set. This method makes for a comprehensive evaluation of potential hyper-parameter configurations, thereby enhancing the model's accuracy and generalization capabilities.

4.2. Parameter Grid

For the grid search, we define the following parameter grid:

```
param_grid = {
 'learning_rate': [0.001, 0.01, 0.1],
 'batch_size':    [16, 32],
 'epochs':        [10],
 'linear_layer_in_features':
                  [[1280],
                  [1280, 640],
                  [1280, 640, 320]],
 'optimizer':     ['adam', 'sgd']
}
```

This grid includes a range of learning rates, batch sizes, and configurations for linear layers in the classifier, and optimizer types. By evaluating all possible combinations within this grid, we identify the hyperparameters that maximize the model's performance.

The 'linear_layer_in_features' parameter refers to the configuration of the linear layers in the model's classifier, specifically the number of linear layers and the number of neurons in each layer. The values are lists of integers that indicate the number of neurons for each layer, which our function for replacing the model's classifier part accepts as input.

To best show how to interpret this, let's consider an example: the value [1280, 640, 320]. This means:

- The first linear layer has 1280 input features and 640 output features.
- The second linear layer has 640 input features and 320 output features.
- The third linear layer has 320 input features and 'num_classes' output features, where 'num_classes' is the number of possible classes in our classification problem.

The reason we always start with 1280 input features is because the last layer in the feature extractor of our pre-trained model (EfficientNet-B0) has 1280 neurons.

We use these different linear layers to see if adding more layers can make it so the model can better combine the output of the feature extractor to achieve better classification results. The two optimizers we use are quite well-known and used a lot in the literature. We vary the learning rate in such a way to see if how a larger or smaller step size influences the finding of a local/global optimum for the parameters of the model network.

4.3. Scene Classification Specifics

For the scene classification tasks in the self-supervised learning pipelines, we utilize the best model from the pretext tasks without altering the number of linear layers. The linear layer configurations are kept consistent with those in the fully-supervised model, i.e. we add the original classifier to the end of the feature extraction part.

4.4. Avoidance of Overfitting

Overfitting is a common challenge in deep learning, where a model performs well on the training data but fails to generalize to unseen data. To mitigate overfitting, we implement several strategies:

4.4.1. Monitoring Train and Test Loss/Accuracy. We plot the training and validation loss and accuracy after each epoch to monitor the model's performance. Observing these plots can help us detect overfitting when the training accuracy continues to improve while the validation accuracy plateaus or deteriorates.

We can use this knowledge for early stopping. Early stopping involves halting the training process when the validation loss stops improving for a predefined number of epochs. This prevents the model from overfitting to the training data by stopping the training at the optimal point.

4.4.2. Incorporating Dropout Layers. Dropout is a regularization technique that involves randomly setting a fraction of input units to zero during training. This prevents the model from becoming overly reliant on specific neurons, thereby promoting generalization. We add a dropout layer in the classifier part of the network to reduce overfitting.

4.4.3. Limiting Hyperparameter Tuning. To prevent overfitting due to excessive hyperparameter tuning, we limit the number of hyperparameters we tune. We do not tune parameters that are not necessary to tune for generalization, focusing instead on learning rates, number of linear layers, and the optimizer.

This experimental setup, encompassing systematic hyperparameter tuning and overfitting prevention measures, lays a solid foundation for evaluating and comparing the effectiveness of supervised and self-supervised learning methodologies in scene classification.

5. Results and Discussion

In this section, we discuss the results obtained from training and evaluating the models using both the fully-supervised and self-supervised learning methodologies. We compare their performance, analyze the strengths and weaknesses of each approach, and provide insights into the observed outcomes.

5.1. Supervised Learning Results

For the fully-supervised learning scheme, we trained the EfficientNet-B0 model on the 15-Scene dataset.

We have always trained for maximum 10 epochs, to also limit overfitting. The value in the table just refers to the maximum number of epochs, if we would notice overfitting for example, we will mention here from which epoch we would select the model in question.

As we can also see in table 1, some of the best models include numbers 7, 11, 13 and 25, 29, 31 (which are the same parameters except for a higher batch size than the first three mentioned models).

We can notice that for very low learning rate, the Adam optimizer in general performs better than SGD. For a learning rate of 0.01, Adam is slightly worse and for the highest learning rate that we consider, Adam performs terribly compared to SGD. Except for the lowest learning rate, SGD also seems to perform better using multiple linear layers than Adam. Changing the batch size doesn't have many significant effects on the performance, it mainly just reduces training time.

From figure 1 and figure 2, we can see that it's best to stop the models 7 and 11 early to prevent overfitting because of the validation loss and accuracy plateauing while the training loss is decreasing. We take epoch 4 for model 7 and epoch 6 for model 11 for example.

5.2. Self-supervised Learning Results

We implemented two self-supervised learning pipelines, each with a distinct pretext task, followed by the downstream task of scene classification.

5.2.1. Pipeline 1: Gaussian Blurring Pretext Task. In table 2, we see that in general the accuracy is very high and around 0.99. The same phenomenon, where Adam performs way worse for higher learning rates, occurs here, but only for more than one linear layer in the classifier part. We have figure 3 for the plots for model 7. Here, we should stop at epoch 3 since the validation performance doesn't improve anymore after this.

In table 3, we have the results concerning the scene classification task for pipeline 1. For comparison reasons, if we discuss two models from the fully supervised task and here, we will compare for the same epoch. If we compare the overall accuracy, it is much lower than the fully-supervised approach. If we compare the best models for epoch 4 (since the fully supervised model 7 took epoch 4), the fully supervised performs a lot better.

TABLE 1: Results for fully supervised scene classification, and the different parameter settings.

Model Nr.	Learning Rate	Batch Size	Epochs	Linear Layers	Optimizer	Best Epoch Accuracy	Last Epoch Accuracy
0	0,001	16	10	[1280]	adam	0,9	0,87
1	0,001	16	10	[1280]	sgd	0,84	0,84
2	0,001	16	10	[1280, 640]	adam	0,91	0,86
3	0,001	16	10	[1280, 640]	sgd	0,79	0,79
4	0,001	16	10	[1280, 640, 320]	adam	0,9	0,88
5	0,001	16	10	[1280, 640, 320]	sgd	0,62	0,62
6	0,01	16	10	[1280]	adam	0,78	0,75
7	0,01	16	10	[1280]	sgd	0,94	0,94
8	0,01	16	10	[1280, 640]	adam	0,77	0,68
9	0,01	16	10	[1280, 640]	sgd	0,94	0,91
10	0,01	16	10	[1280, 640, 320]	adam	0,68	0,59
11	0,01	16	10	[1280, 640, 320]	sgd	0,94	0,93
12	0,1	16	10	[1280]	adam	0,52	0,52
13	0,1	16	10	[1280]	sgd	0,93	0,92
14	0,1	16	10	[1280, 640]	adam	0,3	0,27
15	0,1	16	10	[1280, 640]	sgd	0,92	0,88
16	0,1	16	10	[1280, 640, 320]	adam	0,15	0,15
17	0,1	16	10	[1280, 640, 320]	sgd	0,91	0,84
18	0,001	32	10	[1280]	adam	0,91	0,91
19	0,001	32	10	[1280]	sgd	0,75	0,74
20	0,001	32	10	[1280, 640]	adam	0,91	0,9
21	0,001	32	10	[1280, 640]	sgd	0,69	0,69
22	0,001	32	10	[1280, 640, 320]	adam	0,91	0,9
23	0,001	32	10	[1280, 640, 320]	sgd	0,53	0,53
24	0,01	32	10	[1280]	adam	0,81	0,76
25	0,01	32	10	[1280]	sgd	0,94	0,94
26	0,01	32	10	[1280, 640]	adam	0,78	0,73
27	0,01	32	10	[1280, 640]	sgd	0,92	0,92
28	0,01	32	10	[1280, 640, 320]	adam	0,7	0,69
29	0,01	32	10	[1280, 640, 320]	sgd	0,92	0,92
30	0,1	32	10	[1280]	adam	0,46	0,32
31	0,1	32	10	[1280]	sgd	0,93	0,86
32	0,1	32	10	[1280, 640]	adam	0,26	0,26
33	0,1	32	10	[1280, 640]	sgd	0,91	0,91
34	0,1	32	10	[1280, 640, 320]	adam	0,11	0,05
35	0,1	32	10	[1280, 640, 320]	sgd	0,93	0,93

5.2.2. Pipeline 2: Black and White Perturbation Pretext Task. In table 4, we can see that the overall accuracy is very high and pretty much 100%. This is reasonable as this is a pretty simple task to determine whether there is a black or white perturbation square in the images.

In table 5, we have the results concerning the scene classification task for pipeline 2. The perturbation gives worse results than the gaussian blurring task, and much worse than the fully supervised approach. we expect the gaussian blurring task to perform better in terms of accuracy than the perturbation task, since the perturbation task simply focuses on predicting whether the square in the picture is black or white and tries to learn non-general features for this. The gaussian blurring task needs to predict the amount of blurring and should learn the features like contours and such in the image.

As per assignment, to train a scene-classification model on top of the perturbation pretext model, we also consider the pretext models pre-trained at early epochs, for example the third epoch, as well as the last epoch. I have ran a few scene classification models with different parameters, each time using a pretext model trained for 3 and 10 epochs.

Throughout my tries for different parameter combinations, since we also want to take the variance into account and not only look at one example run, I noticed that sometimes the earlier stopping model gave better results, and sometimes the 10 epoch model gave better results. Overall, the 10 epoch model performs better if we look at all the runs, probably because the earlier stopping model is underfitted a bit and didn't always converge to the optimum. Some exact

TABLE 2: Results for self-supervised pipeline 1 pretext, and the different parameter settings.

Model Nr.	Learning Rate	Batch Size	Epochs	Linear Layers	Optimizer	Best Epoch Accuracy	Last Epoch Accuracy
0	0,001	16	10	[1280]	adam	1	1
1	0,001	16	10	[1280]	sgd	0,98	0,98
2	0,001	16	10	[1280, 640]	adam	1	1
3	0,001	16	10	[1280, 640]	sgd	0,98	0,98
4	0,001	16	10	[1280, 640, 320]	adam	1	1
5	0,001	16	10	[1280, 640, 320]	sgd	0,98	0,98
6	0,01	16	10	[1280]	adam	0,99	0,98
7	0,01	16	10	[1280]	sgd	1	1
8	0,01	16	10	[1280, 640]	adam	0,99	0,99
9	0,01	16	10	[1280, 640]	sgd	1	1
10	0,01	16	10	[1280, 640, 320]	adam	0,97	0,41
11	0,01	16	10	[1280, 640, 320]	sgd	1	1
12	0,1	16	10	[1280]	adam	1	1
13	0,1	16	10	[1280]	sgd	1	1
14	0,1	16	10	[1280, 640]	adam	0,2	0,2
15	0,1	16	10	[1280, 640]	sgd	1	1
16	0,1	16	10	[1280, 640, 320]	adam	0,6	0,2
17	0,1	16	10	[1280, 640, 320]	sgd	1	1
18	0,001	32	10	[1280]	adam	1	1
19	0,001	32	10	[1280]	sgd	0,98	0,98
20	0,001	32	10	[1280, 640]	adam	1	1
21	0,001	32	10	[1280, 640]	sgd	0,98	0,98
22	0,001	32	10	[1280, 640, 320]	adam	1	1
23	0,001	32	10	[1280, 640, 320]	sgd	0,98	0,98
24	0,01	32	10	[1280]	adam	1	1
25	0,01	32	10	[1280]	sgd	1	1
26	0,01	32	10	[1280, 640]	adam	0,99	0,98
27	0,01	32	10	[1280, 640]	sgd	1	1
28	0,01	32	10	[1280, 640, 320]	adam	0,9	0,86
29	0,01	32	10	[1280, 640, 320]	sgd	1	1
30	0,1	32	10	[1280]	adam	0,99	0,44
31	0,1	32	10	[1280]	sgd	1	1
32	0,1	32	10	[1280, 640]	adam	0,2	0,2
33	0,1	32	10	[1280, 640]	sgd	1	1
34	0,1	32	10	[1280, 640, 320]	adam	0,41	0,2
35	0,1	32	10	[1280, 640, 320]	sgd	1	1

TABLE 3: Results for self-supervised pipeline 1 scene classification, and the different parameter settings.

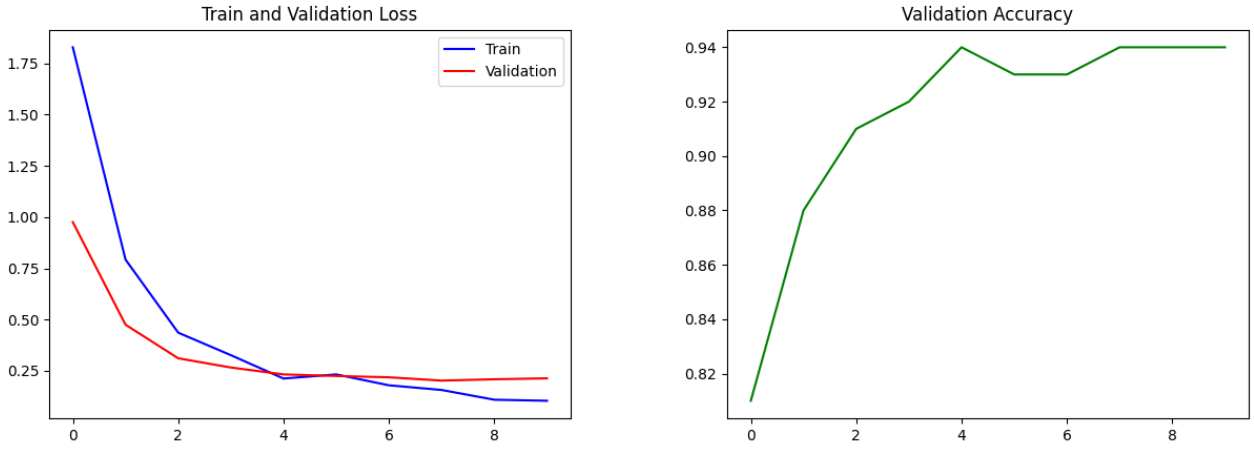
Model Nr.	Learning Rate	Batch Size	Epochs	Linear Layers	Optimizer	Best Epoch Accuracy	Last Epoch Accuracy
0	0,001	16	10	[1280]	adam	0,4	0,38
1	0,001	16	10	[1280]	sgd	0,32	0,3
2	0,01	16	10	[1280]	adam	0,38	0,36
3	0,01	16	10	[1280]	sgd	0,35	0,33
4	0,1	16	10	[1280]	adam	0,34	0,25
5	0,1	16	10	[1280]	sgd	0,38	0,38
6	0,001	32	10	[1280]	adam	0,4	0,4
7	0,001	32	10	[1280]	sgd	0,31	0,31
8	0,01	32	10	[1280]	adam	0,39	0,35
9	0,01	32	10	[1280]	sgd	0,34	0,34
10	0,1	32	10	[1280]	adam	0,32	0,3
11	0,1	32	10	[1280]	sgd	0,38	0,36

TABLE 4: Results for self-supervised pipeline 2 pretext, and the different parameter settings.

Model Nr.	Learning Rate	Batch Size	Epochs	Linear Layers	Optimizer	Best Epoch Accuracy	Last Epoch Accuracy
0	0,001	16	10	[1280]	adam	1	1
5	0,001	16	10	[1280, 640, 320]	sgd	1	1
6	0,01	16	10	[1280]	adam	1	1
11	0,01	16	10	[1280, 640, 320]	sgd	1	1
12	0,1	16	10	[1280]	adam	1	1
17	0,1	16	10	[1280, 640, 320]	sgd	1	1
18	0,001	32	10	[1280]	adam	1	1
23	0,001	32	10	[1280, 640, 320]	sgd	1	1
24	0,01	32	10	[1280]	adam	1	1
29	0,01	32	10	[1280, 640, 320]	sgd	1	1
30	0,1	32	10	[1280]	adam	1	1
35	0,1	32	10	[1280, 640, 320]	sgd	1	1

TABLE 5: Results for self-supervised pipeline 2 scene classification, and the different parameter settings.

Model Nr.	Learning Rate	Batch Size	Epochs	Linear Layers	Optimizer	Best Epoch Accuracy	Last Epoch Accuracy
0	0,001	16	10	[1280]	adam	0,26	0,24
1	0,001	16	10	[1280]	sgd	0,17	0,16
2	0,01	16	10	[1280]	adam	0,25	0,21
3	0,01	16	10	[1280]	sgd	0,2	0,19
4	0,1	16	10	[1280]	adam	0,21	0,19
5	0,1	16	10	[1280]	sgd	0,25	0,18
6	0,001	32	10	[1280]	adam	0,26	0,25
7	0,001	32	10	[1280]	sgd	0,16	0,13
8	0,01	32	10	[1280]	adam	0,27	0,22
9	0,01	32	10	[1280]	sgd	0,18	0,14
10	0,1	32	10	[1280]	adam	0,27	0,27
11	0,1	32	10	[1280]	sgd	0,18	0,18



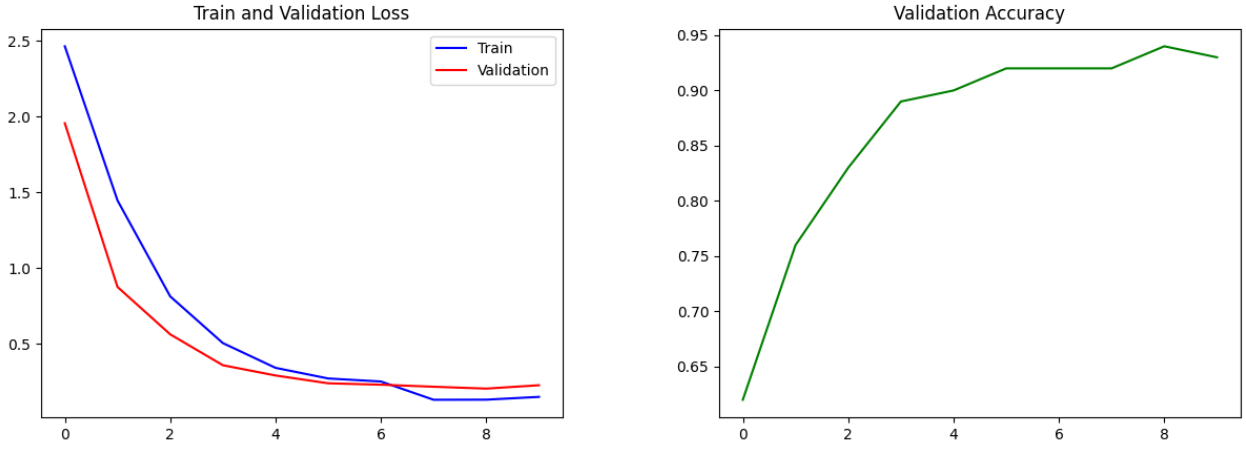
(a) (b)

Figure 1: (Left) Train and validation loss plot for fully supervised model 7. (Right) Validation accuracy plot for fully supervised model 7.

values include: 3-epoch (early stopping): [0,69 ; 0,74 ; 0.11]; 10-epoch (last epoch stopping): [0.2 ; 0.7 ; 0.09].

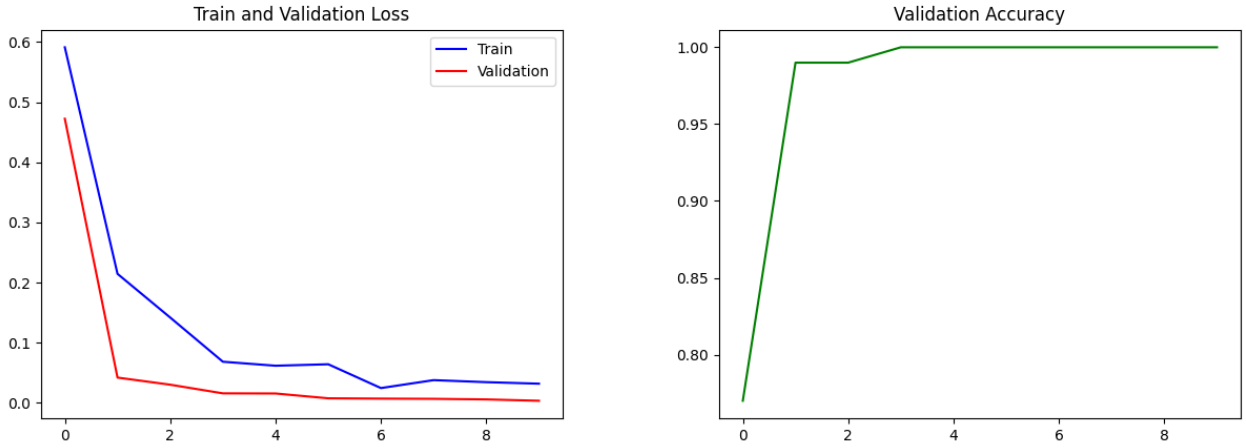
6. Conclusion

In general, the better models from the fully supervised approach perform better than the best models of both the



(a) (b)

Figure 2: (Left) Train and validation loss plot for fully supervised model 11.
(Right) Validation accuracy plot for fully supervised model 11.



(a) (b)

Figure 3: (Left) Train and validation loss plot for self-supervised 1 pretext, model 7.
(Right) Validation accuracy plot for self-supervised 1 pretext, model 7.

self-supervised pipelines. The self-supervised scheme can be useful since we don't need pre-annotated data, in order to get a decent feature extractor network.

I did notice that when I trained the scene classification for the pipeline tasks for a specific parameter setting, the score would be way higher (around 70%) than when I used the gridsearch, which may mean there is something going wrong in the gridsearch function, but I didn't see anything wrong. When running for the parameter setting separately, the gaussian blurring task scene classification comes a bit close to the fully supervised task with a slightly worse accuracy. The perturbation task performed a bit worse than the gaussian blurring.

7. Notes

The code for this project can be found at https://github.com/MaksimKarnaukh/ANN_project. The assignment instructions file can also be found there.