Artificial Neural Networks Project

Self -Supervised Scene Classification

Author:
Karol Zioło

Introduction

The primary objective of this project was to compare the performance capabilities of different models that employed fully supervised and self-supervised techniques. The ultimate goal was to develop three distinct classification models. To facilitate this, we were provided with the 15-Scene Dataset, which consisted of images representing various scenes across 15 different classes, including office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, and suburb. The dataset was conveniently divided into separate train and test sets for training and evaluation purposes.

The first model we constructed followed a fully supervised technique, wherein we employed a fine-tuning approach on the pretrained EfficientNet-B0 architecture. This involved utilizing the pre-existing knowledge captured by the EfficientNet-B0 model and adapting it to our specific classification task. By fine-tuning the model, we aimed to optimize its performance on our target dataset.

The other two models were designed based on self-supervised learning strategies. The second model utilized Rotation Classification as a Pretext Task. In this approach, we subjected the images to rotations at four different angles: 0°, 90°, 180°, and 270°. The labels for these images were determined based on their respective rotations. The features learned through this pretext task were then employed in the subsequent classification stage, where we added new fully connected layers and trained them using the actual target data. Similarly, the third model also employed self-supervision, but this time we used Perturbation Classification as a Pretext Task. Here, we introduced black and white 10x10 squares as perturbations within the images. The model was trained to classify the color of these perturbations. By doing so, the model learned to extract meaningful features that could aid in the subsequent classification task.

After constructing these models, we evaluated their performance using various metrics. We compared the accuracies of the models on the validation set to assess their classification capabilities. Additionally, we examined the confusion matrices, which provided insights into the specific classes where the models faced challenges in accurately distinguishing between them. To gain further understanding of the models' decision-making processes, we applied the Score-Cam technique. This allowed us to visualize the regions of the image that were taken into account by specific layers of the models when determining the assigned class. This analysis helped us understand the models' attention mechanisms and the areas of importance within the input images. Furthermore, for the purpose of model interpretation, we introduced the concept of model inversion. By employing model inversion techniques, we could illustrate the features

that maximized the activation of the output neurons associated with the selected classes. This visualization aided in comprehending the specific visual patterns and characteristics that influenced the models' classification decisions.

Overall, this project encompassed a comprehensive analysis of models utilizing fully supervised and self-supervised techniques. Through rigorous evaluation and interpretation methodologies, we aimed to gain insights into their performance, strengths, and limitations in the context of scene classification.

1. Fully supervised learning

To identify the optimal model for a fully supervised task, we developed six different models, each exploring various aspects of architecture, data preprocessing, and learning techniques. Through thorough analysis, we were able to determine the most effective combinations for our dataset and components for training that would be applied in the final model.

Initially, we aimed to investigate whether our images would benefit from more advanced transformation techniques. Consequently, we created two data loader functions. The first function, named **simple_15SceneData**, incorporated basic transformations suitable for training purposes. These transformations included Resize, ToTensor, and Normalize. Since the EfficientNet-B0 model requires input images of size 224x224, we utilized the Resize transformation with these dimensions. Additionally, considering that our images were black and white, we opted for a mean of 0.5 and a standard deviation of 0.5.

The second function, **load_15SceneData**, involved more advanced augmentation techniques. These techniques are detailed in the table below. For the code implementation of both functions, please refer to **supervising_task.dataloader.py**.

Table 1 Transformers for Model 2

Augmentation technique	Parameters
Resize	(224,224)
RandomHorizontalFlip	p = 0.5
RandomResizedCrop	size = 224
RandomRotation	degrees = 20
ToTensor	-
Normalize	mean = (0.5,) std=(0.5,)

In order to determine the appropriate data loader for comparing different approaches, a simple architecture called **Model1** was created. This involved replacing the classifier part of the

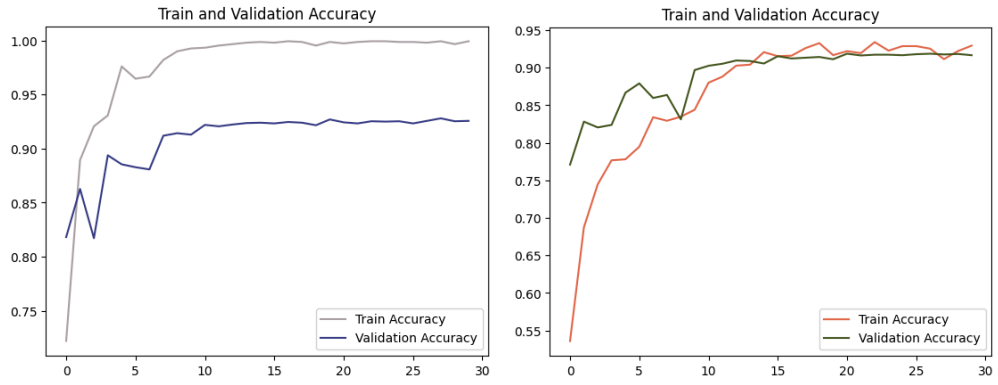
EfficientNet-B0 model with Dropout and Linear layer. Specifically, the dropout layer had a probability of 0.2, and the last layer consisted of 15 neurons to align with the number of classes in the dataset. The complete architecture can be found in the **supervising_task.models.py** file.

To evaluate these two approaches, the train function from **helper_functions.py** was employed. By default, this train function utilized **Adam** as the optimizer and **Cross-Entropy** as the loss function. Adam is a widely used optimization algorithm for multi-class classification tasks, thanks to its adaptive learning rate that enables efficient parameter updates and its capability to handle sparse gradients. Cross Entropy loss, on the other hand, is well-suited for multi-class classification due to its ability to measure dissimilarity between predicted class probabilities and the true class labels. It encourages the model to assign high probabilities to the correct class and offers a continuous and differentiable objective function for efficient gradient-based optimization during training. Furthermore, cross-entropy loss naturally accommodates the probability distribution nature of multi-class classification, making it a standard choice for such tasks. While the original formulation of Cross Entropy loss requires providing probabilities as input, PyTorch conveniently allows using the raw output from the network without explicitly computing the softmax probabilities beforehand. This simplifies the implementation without compromising the loss calculation.

In addition, a learning rate scheduler was applied to gradually decrease the learning rates during training, aiding in reaching the minimum loss. The train function also incorporated an early stopping method to prevent overfitting by monitoring the validation loss.

For both approaches, the models were trained for 30 epochs, with a fixed learning rate of 0.001. However, this time the early stopping method was not utilized. The figures below provide visual comparisons of the validation performance between the training and validation sets for both models.

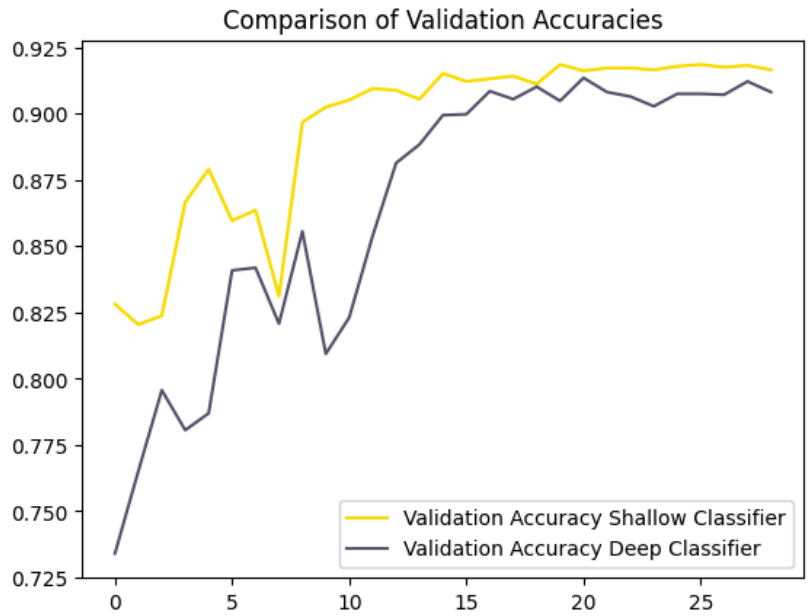
Figure 1 Train and validation accuracy comparisons in respectively first and second model.



The obtained results indicate that the model using basic transformations tends to overfit, with the training accuracy approaching 100% while the validation accuracy reaches a saturation point at around 92%. In contrast, the model utilizing more advanced augmentation techniques demonstrates a similar trend between training and validation accuracies, suggesting that the applied data augmentation successfully addresses the overfitting issue. Furthermore, the validation scores for both models are comparable.

Consequently, the decision was made to proceed with the data loader incorporating all data augmentation techniques for all further analysis. Subsequently, an exploration was conducted to assess whether the classification part of the model should be deeper. For this purpose, two blocks consisting of dropout, linear, and ReLU layers were introduced, followed by another block with dropout and linear layer. Each dropout layer had a dropout probability of 0.2. Additionally, the out-channels of the linear layers were set to 640, 320, and 15, respectively. The resulting architecture, known as **Model2**, can be found in the same script as the previous one. The model with deeper classifier was trained using the same parameters as the previous one (the training results can be found in Appendix 1). Ultimately, the validation accuracy, in comparison to the shallow network, showed a slight decline, as depicted in the figure below. Based on these findings, it was decided to maintain the shallow network architecture as deeper one doesn't improve the performance.

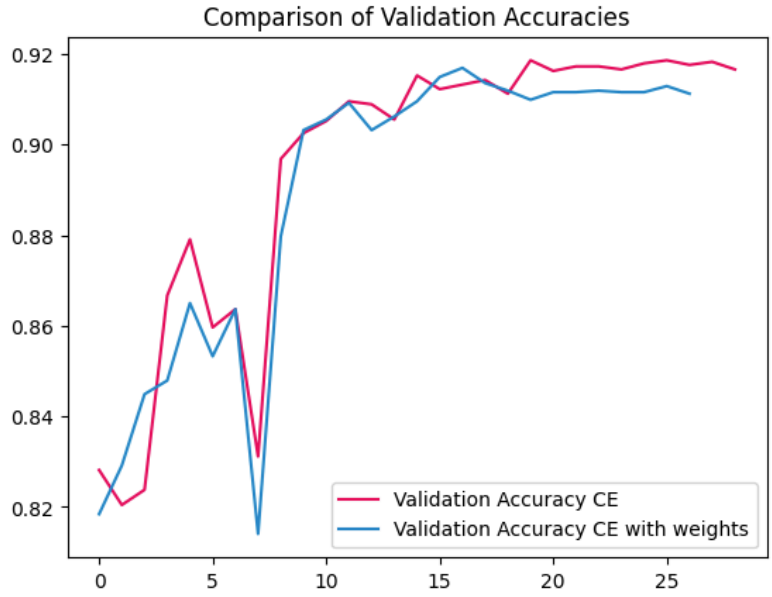
Figure 2 Validation Accuracy comparison Deep vs Shallow Classifier



In the subsequent phase of the analysis, it was decided to experiment with the choice of loss function. This decision was driven by the findings of the confusion matrix analysis conducted on the model with a shallow classifier and augmented data (refer to Appendix 14). The analysis revealed that certain classes, namely open country, coast, and mountain, were prone to misclassification. To address this issue, the weight parameter in the Cross Entropy Loss function was introduced. For the mentioned classes, the weight parameter was set to 1.5, while for the remaining classes, it was set to 1. In general, this weighting scheme aims to mitigate the impact of class imbalances and prioritize the correct classification of the challenging classes during the loss calculation. The rationale behind this approach was to encourage the model to focus more on learning these specific classes (training results can be found in the Appendix 2).

However, the results obtained after incorporating the weighted loss function did not show significant improvement, and in fact, the model's performance was slightly worse compared to the original configuration. This observation is visually evident in the figure provided below. As a result, it was determined that applying these weights did not yield substantial benefits, leading to the decision to retain the original model configuration without the additional weighting scheme.

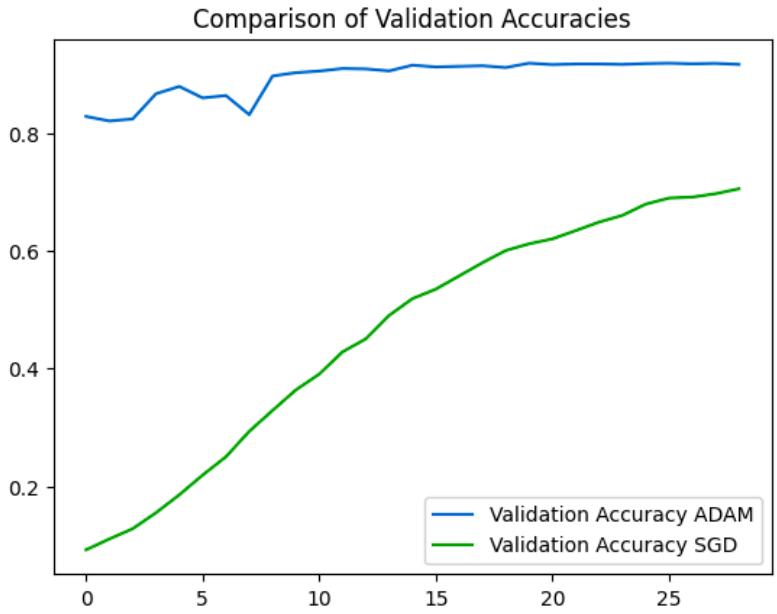
Figure 3 Validation Accuracy comparison between CE and CE with weights.



In the subsequent stage, I aimed to explore alternative optimizers, particularly SGD. Given the largeness of our dataset, it is generally advised to use Adam instead of SGD. Adam's

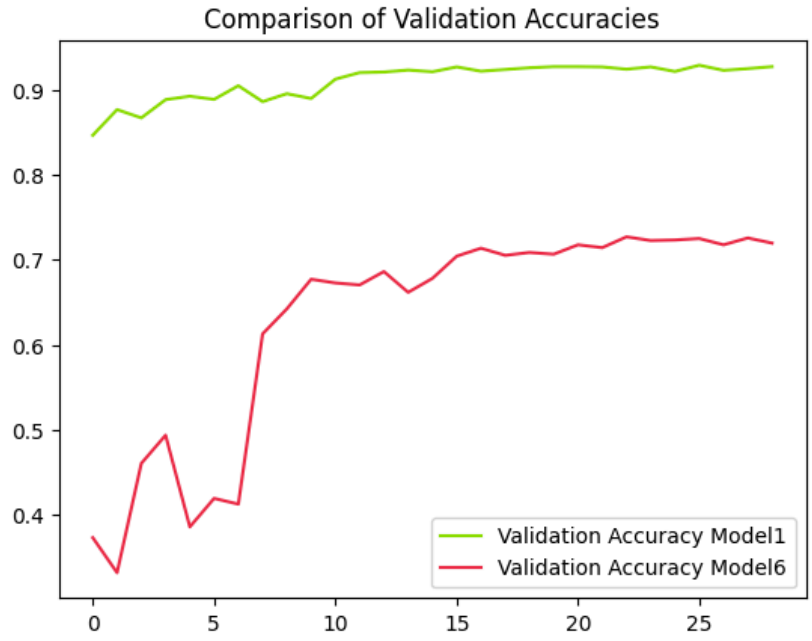
adaptive learning rate and momentum can accelerate convergence and effectively handle the intricacies of extensive datasets. Nonetheless, it is advantageous to conduct experiments with both optimization algorithms to ascertain the most suitable choice based on factors such as the neural network architecture and dataset characteristics (training results can be found in Appendix 3). From the observed outcomes presented below, it is evident that ADAM outperforms SGD significantly, as SGD requires a greater number of epochs to acquire effective learning.

Figure 4 Validation Accuracy comparison ADAM vs SGD



For the final experiment, the decision was made to test a higher learning rate in hopes of facilitating rapid learning in the early stages. The intention was to subsequently employ a learning scheduler to gradually reduce the learning rate and converge to a minimum. Accordingly, a learning rate of 0.01 was chosen (training results can be found in Appendix 4). However, contrary to expectations, the results, as indicated in the figure below, were not favorable. The model became stuck at the 10th epoch, with the validation accuracy plateauing at around 70%. It was speculated that the model had become trapped in a local minimum.

Figure 5 Validation Accuracy comparison high LR vs low LR



In summary of this analysis, the validation accuracies for the ultimate version of each model were recorded and are displayed in the table provided.

Table 2 Model comparisons

Model	Validation Accuracy
Unaugmented data loader	0.9289
Base model	0.9185
Deep Classifier	0.9135
Huber Loss	0.9219
SGD optimizer	0.7051
High LR	0.7269

After a comprehensive analysis of the results, the necessary components for the final model were determined. Specifically, it was decided to train the model using augmented data, while excluding the RandomResizedCrop transformation due to its negative impact on the Validation Accuracy. The classifier adopted for the final model was a shallow network, as the deeper architecture did not yield any improvements in the results.

The training parameters were carefully chosen, with ADAM selected as the optimizer due to its efficiency compared to SGD. To account for the exclusion of the Crop transformer, the Cross Entropy loss function was employed with weighted values of 1.5 assigned to the open country, coast, and mountain classes. This weighting scheme was expected to compensate for

the absence of the Crop transformer and enhance performance. Lastly, the learning rate was set to 0.001 to facilitate effective model training and optimization (training results can be found in Appendix 5).

The top-performing model achieved an accuracy of 93% during training. However, when the Crop transformer was removed, the model became more vulnerable to overfitting. Interestingly, there was no observed decrease in the validation curve, and in fact, it achieved the highest validation score compared to other models.

The analysis of the supervising task could be found in the **supervising_task.report.ipynb** script.

2. Self-supervised learning

In this section, we will outline the design of two distinct pipelines for a self-supervised learning scheme. Each pipeline consists of a two-step learning process, beginning with a pretext task and followed by the main task of scene classification. The first pipeline focuses on rotation classification as the pretext task, while the second pipeline incorporates perturbation classification as the pretext task. In both pipelines, the model trained during the pretext task will undergo further fine-tuning to tackle the primary objective of scene classification.

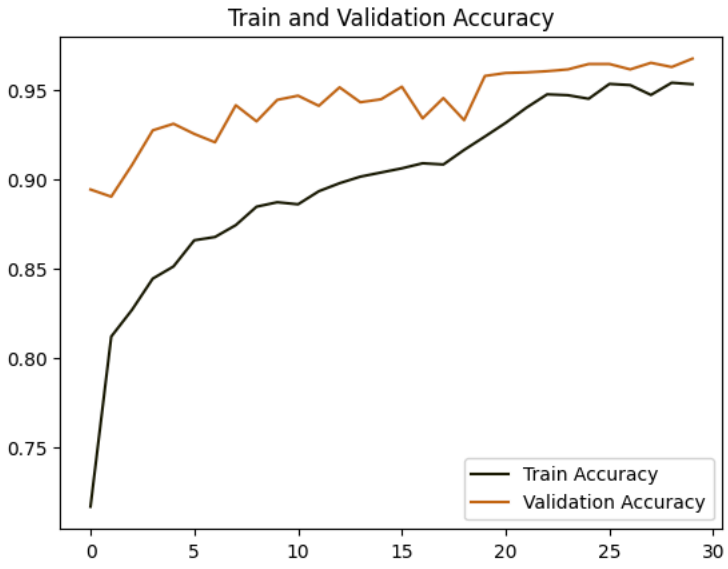
2.1 Pipeline 1 - Rotation

At the outset of this task, our objective was to develop a model capable of classifying rotations applied to images. This pretext task aimed to facilitate the model's learning of relevant features. Subsequently, the learned feature parameters were frozen and utilized for classifying the original 15scene dataset. However, prior to that, proper dataset preparation was necessary. To achieve this, the **RotatedDataset** class was created in the **self_supervising_task.dataloader.py** script. This class enabled the application of four different rotations to the images. The **train_rotation_loader** and **valid_rotation_loader** functions were responsible for loading the data, concatenating the sub-datasets for each rotation, and shuffling the data to avoid correlation. Additionally, several basic transformations were applied, including Resize, ToTensor, RandomResizedCrop, and Normalize. The Resize transformation ensured that images were resized to dimensions of 224x224, while the Normalize transformation utilized a mean of 0.5 and a standard deviation of 0.5.

Given that the pretext task's objective was feature detection, a default architecture with a shallow classifier was deemed appropriate. It was called **Rotation1** and could be found in **self_supervising_task.models.py** script. This architecture consisted of a single block comprising a Dropout layer with a probability of 0.2 and a Linear layer with four output neurons corresponding to the number of applied rotations. Bassed on the experience from the previous task it was decided that during model training, the ADAM optimizer was employed, along with the Cross Entropy loss function and a learning rate of 0.001. A learning rate scheduler was also utilized.

The training results, depicted in the accompanying figure, indicate that the model did not exhibit signs of overfitting. In fact, a slight degree of underfitting was observed. The achieved highest score of 0.9678 is relatively high. Based on these compelling results and tests conducted in previous section, it was determined that no further modifications were necessary, as the model's performance was already acceptable.

Figure 6 Train vs Validation Accuracy of rotation pretext model.



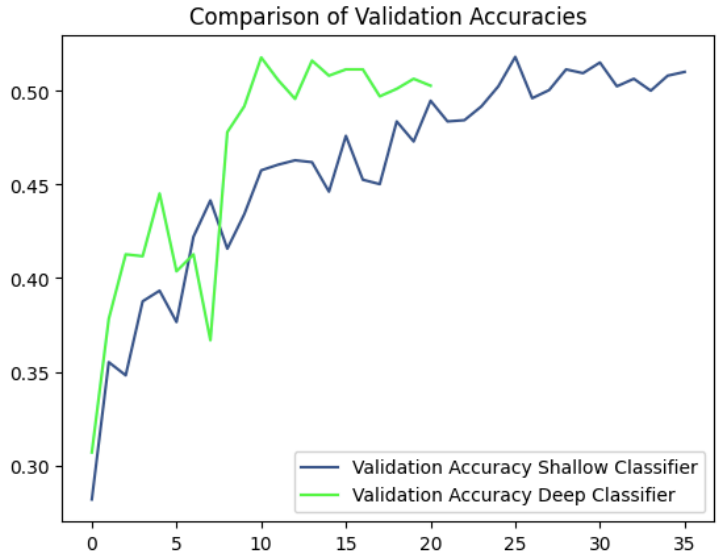
The subsequent phase involved the classification task, where all the feature parameters were frozen, and three different models were trained. The first model utilized fully augmented data with a shallow classifier, employing the same training parameters as before, but with an increased number of epochs to 50 and an early stopping patience of 10 (training results can be found in Appendix 6).

To explore the potential performance improvement with a deeper classifier, the second model employed the same classifier as in the supervising task. The rationale behind this was

that the learned features might exhibit more intricate relationships as they were trained for different purposes. Therefore, complex classifier could better capture these nuances. Again, augmented data and the same parameters were used. To train this model the same parameters were used as in the previous one (training results can be found in Appendix 7).

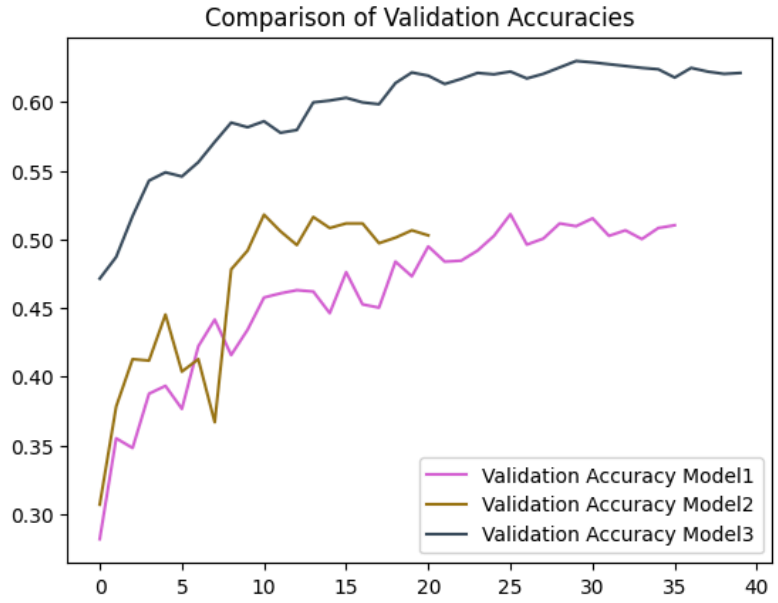
A comparison of the performance between the two approaches was conducted, as depicted in the figure provided. Despite the use of an early stopping criterion, resulting in different stopping epochs for each model, both approaches eventually achieved similar accuracies. Consequently, the decision was made to proceed with the shallower classifier since the deeper one did not yield a substantial improvement in performance.

Figure 7 Shallow vs Deep Classifier



Subsequently, it was observed that training on augmented data might be unnecessary in this case since the feature parameters were already learned, and only the classifier needed to be trained. It was believed that the transformations applied during augmentation might interfere with the classifier's ability to effectively utilize the learned features, rendering them unnecessary. Additionally, the training results of these models also demonstrate clear signs of significant underfitting. Consequently, the last model was trained on data loaded by **simple_15SceneData** (training results can be found in Appendix 8). As expected, this decision resulted in an improvement in performance of approximately 15%. Moreover, that change also solved the issue with underfitting. The comparison of all classification models is depicted in the figure below. Based on that it was decided to select the last model as a final one for Pipeline 1.

Figure 8 Comparison of all models.



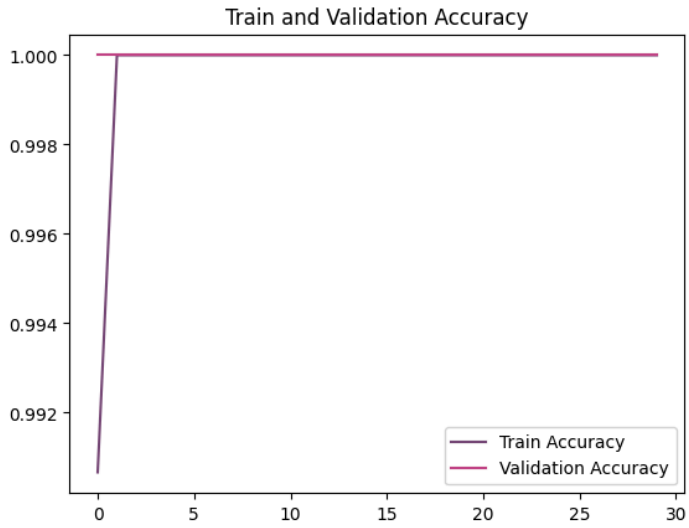
2.2 Pipeline 2 - Perturbation

The objective of this project phase was similar to the previous one, but with a different approach. Instead of training the model based on rotations, this time the focus was on perturbing the dataset. The concept involved applying 10x10 black or white squares to random parts of the images. To implement this, the **PerturbationDataset** class was created in the self_supervising_task.dataloader.py script. This class allowed for the application of two different perturbation techniques to the images. The **train_perturbation_loader** and **valid_perturbation_loader** functions utilized this class to create two sub-datasets with black and white perturbation techniques, which were then concatenated and shuffled. For the training loader, several transformations were applied, including resizing the images to 224x224, random horizontal flipping with a 50% probability, random rotation by 20 degrees, conversion to tensor format (ToTensor), and normalization with a mean of 0.5 and standard deviation of 0.5. However, the validation loader only applied basic transformations: resizing to 224x224, conversion to tensor format, and normalization with a mean of 0.5 and standard deviation of 0.5.

To train the model for this pretext task, the **Perturbation1** class from **self_supervising_task.models.py** was utilized. Similarly, to the previous phase, a shallow classifier was employed, consisting of a Dropout layer with a 0.2 probability and a Linear layer with two output neurons. The training process employed the ADAM optimizer and Cross

Entropy loss. A learning rate of 0.001 was set, along with a learning scheduler. The training results are depicted in the graph provided below.

Figure 9 Train and validation accuracy of pretext model.

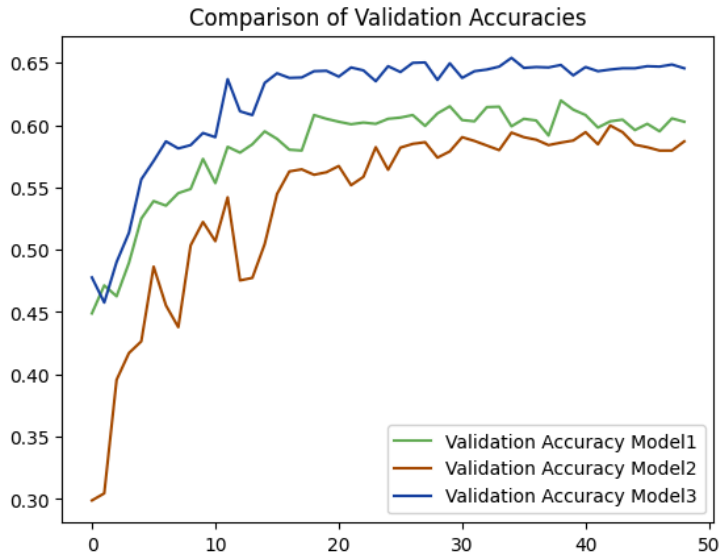


Based on the obtained results, it is evident that the model quickly learns to distinguish between labels, as the validation accuracy reaches 100% after the first epoch. As a result, the validation accuracy curve in the graph remains relatively constant at around 100%. These promising results led to the decision of selecting this model for the subsequent phase of the task.

In the classification part, a similar approach to the previous pipeline was adopted. Two different classifiers and two different data loaders were explored. Firstly, a model with a shallow classifier was trained using the **load_15SceneData** data loader, which included advanced transformations. The second model also utilized the **load_15SceneData** data loader, but it's architecture consisted of a deeper classifier. For the third model, the **simple_15SceneData** data loader with basic data augmentation was used, along with a shallow classifier. The training results for each model are provided in Appendix 9-11.

Analyzing these results, it is apparent that the advanced data augmentation led to underfitting, which hindered the training process. Additionally, the deep classifier exhibited a slight decrease in performance compared to the other models. To facilitate a comprehensive comparison of these models, a plot has been included below.

Figure 9 Comparison of all models.



The plot provides compelling evidence that the model utilizing a shallow classifier and trained with basic transformations data achieved the highest performance, reaching a validation accuracy level of 65%. The other models also demonstrated good performance, with accuracy levels around 60%. Consequently, the decision was made to choose the last model as the final one, given its superior performance.

The detailed analysis of the self-supervising task can be found in the **self_supervising_task.report.ipynb** script.

3. Evaluation

The evaluation part of a neural network project is crucial for assessing the performance, reliability, and generalizability of the trained model. It involves various metrics, techniques, and methodologies to measure how well the model performs on unseen data. The evaluation phase helps to validate the model's accuracy, identify areas for improvement, compare different models, and make informed decisions about deploying the model in real-world scenarios. In this part I was required to provide training settings for each model. Additionally, I needed to create confusion matrices and compare model's accuracy. At the end the of this part the evaluation and interpretation methods were applied. More specifically the score-cam method was used for evaluation purposes and model inversion for interpretation one.

The table in Appendix 12. presents the settings for each tested model. The highlighted examples are the final ones. Additionally, I collected validation accuracies for each tested

model. They are presented in the table below. Moreover, the confusion matrices were added in the Appendix 13-27 (final supervised: Appendix 19, final rotation scene classification: Appendix 23, final perturbation scene classification: Appendix 27).

Table 3 Accuracies for all model

Model	Accuracy
Supervised 1	0.9289
Supervised 2	0.9185
Supervised 3	0.9135
Supervised 4	0.9219
Supervised 5	0.7051
Supervised 6	0.7269
Supervised Final	0.9300
Rotation 1	0.9678
Rotation–classifier 1	0.5182
Rotation–classifier 2	0.5179
Rotation–classifier 3	0.6298
Perturbation 1	1.0000
Perturbation-classifier 1	0.6197
Perturbation-classifier 2	0.5996
Perturbation-classifier 3	0.6539

Based on the provided table, it can be inferred that supervised models, in general, exhibited superior performance compared to self-supervised models. This disparity can be attributed to the nature of fully supervised learning, which enables models to acquire specific and accurate features essential for data scene classification. Conversely, self-supervised methods focus on different feature extraction approaches, as they aim to address a variety of classification tasks. While self-supervised models have their own merits, such as the ability to learn representations without explicit labels, their focus on diverse tasks may lead to a relatively lower performance in certain scenarios compared to fully supervised models. Taking a closer look, it can be speculated that rotation models primarily concentrate on smaller objects, utilizing their positions to infer the image's rotation. Conversely, perturbation models prioritize the detection of black or white squares, potentially leading to less attention being given to complete objects within the scene. On the other hand, supervised models have the advantage of learning about more complex objects, as they are guided by explicit labels during training. This focus on comprehensive object understanding contributes to the superior performance of supervised models in capturing the intricacies of the data.

The significance of emphasizing the detection of specific features becomes evident when examining the confusion matrices of these models. A closer analysis reveals that self-supervised methods encountered challenges with similar types of scenes, particularly those related to natural environments such as open country, coast, forest, and mountain. Similarly, scenes associated with indoor spaces like bedrooms, kitchens, living rooms, and offices were prone to misclassification. These observations highlight the need for the improvement in capturing the distinctive characteristics of these specific scenes within the self-supervised models.

Furthermore, it is important to note that my primary focus from the beginning was to prevent overfitting, which led to the implementation of several techniques. Firstly, Dropout layers were incorporated when creating classifiers. This technique reduces dependence on specific neurons, promoting learning from diverse neuron combinations, enhancing generalization to new data, and mitigating overfitting. Additionally, various types of transformations were employed, but they did not yield significant changes in terms of overfitting. Only the Crop transformation resulted in a decrease in training accuracy without gaining additional training abilities. In addition to these two techniques, the early stopping criterion was utilized, halting the learning process when the validation accuracy started to decline.

In the subsequent phase, we employed the score-cam evaluation method. This involved generating a distinct set of eight images, all of which were correctly predicted by each model. The results of this analysis have been included in the Appendix. The score-cam was applied to these eight images for the first, second, and final blocks of the architecture. Let's first concentrate on the output of the supervised model (Appendix X). We can observe that in the initial two layers, the model predominantly emphasizes low-level features. Notably, it highlights the borders of objects, indicating its focus on capturing these foundational visual elements that play a crucial role in detecting more intricate patterns in subsequent layers.

This emphasis on low-level features is particularly apparent in the visualization of the final block of layers. Upon closer inspection, we can determine which regions of the image indicate belonging to specific classes. In the street image, the model focuses on the buildings, pavement, and partially on the car. However, when examining the tall building image, the model accurately detects all three skyscrapers. In the case of the forest image, the model identifies all the trees. There is also heatmap for the coast scene and model focuses on the waves and parts of the beach. For the inside city image, the model's attention is drawn to the banner. Similarly, in the mountain picture, it detects the summits. In contrast, for the living

room image, the model's focus extends to a larger region, encompassing the sofa, windows, and the picture on the wall. Lastly, in the open country picture, the heatmap indicates that the model places importance on the field, forest, and even portions of the sky. These observations shed light on the model's ability to selectively highlight and identify key elements corresponding to different scenes or objects within the images.

Moving on to the rotation model (Appendix 28), upon closer examination, it becomes evident that the model captures different parts of the low-level features for various images. For instance, if we take a closer look at the forest heatmap, we can notice that the model focuses on the trunks, branches, and ground, which are crucial components of the forest scene. In contrast, in the previous case, the model emphasized the leaves. Similarly, this observation holds true for the mountain heatmap. Previously, the model focused on the summit and sky, emphasizing their importance in the scene. However, this time, the model directs its attention to the bottom of the mountains, possibly capturing different visual elements that contribute to the understanding of the mountain landscape.

Analyzing the outputs of the last block of layers for this rotation model (Appendix 29) unveils its distinct focus on different regions compared to the supervised model. In general, it appears to be more attuned to smaller objects or finer details within the scene. For example, when examining the tall building heatmap, we can notice that the model zooms in on the skyscraper and parts of the sky, possibly capturing architectural features that are indicative of tall buildings. Similarly, in the case of the forest heatmap, the model highlights the sky regions between trees, possibly paying attention to the interplay between the vegetation and the sky, which is characteristic of forest scenes. Furthermore, a notable distinction can be observed in the living room scenario, where the model directs its attention to the railing of the stairs and partially to the sofa. Similarly, in the mountain heatmap, the model highlights certain sections of the summits as well as a portion of the bottom of the mountain, further exemplifying its capability to discern different regions of interest within the image.

The perturbation model (Appendix 30) also directs its focus to different regions of the image. The distinction of low-level features is noticeable in a similar manner to the supervised model. However, in terms of the last layer block, the perturbation model, in contrast to the rotation model, appears to place more emphasis on broader spatial regions rather than specific objects or details. Nonetheless, the highlighted regions are not as specific as those captured by the supervised model. For instance, upon closer inspection of the street heatmap, we can notice that the model focuses on the top portion of the image, indicating a general perception of the street scene. The same trend can be observed in the inside city and forest heatmaps, where the

model highlights broader spatial regions without zooming in on specific objects. However, when examining the living room heatmap, we observe more specific regions, such as the fireplace or coffee table, are highlighted, suggesting a level of object recognition. Similarly, in the mountain heatmap, the entire mountain range is emphasized, signifying a broader understanding of the mountainous landscape.

In the interpretation part of the project, we delve into the process of understanding and interpreting the features learned by different models. For this analysis, we select five classes of interest from a total of fifteen. Our goal is to generate synthesized images that showcase the features maximizing the activation of the output neuron associated with each selected class. The results for each model are presented in the Appendix 31.

Let's commence our analysis with the supervised model. Upon examining the visualizations, several intriguing observations can be made. In the case of the bedroom class, we can discern a distinct, floating shape resembling a bed, which aligns with our expectations. Moving on to the coast class, horizontal wavy lines are prominently visible, suggesting the presence of waves, a characteristic feature of coastal scenes. Similarly, for the forest class, we can observe multiple vertical shapes, reminiscent of trees, further confirming the model's ability to capture relevant features. An interesting scenario arises with the highway class. Towards the bottom of the image, there are discernible shapes resembling a road with pavements on either side. In addition, a rectangular shape in the background implies the presence of buildings in the distance, enhancing the realism of the visualization. Interpreting the industrial picture proves to be more challenging. Wavy vertical lines with a circular center are observed, these might suggest the presence of structures or machinery. However, the exact interpretation of this image remains more ambiguous compared to the other classes.

When examining the visualizations generated by the rotation model, we encounter a greater challenge in their interpretation. The ability to discern distinct features becomes less obvious compared to the supervised model. In the case of the bedroom class, some geometric shapes are noticeable, particularly a structure in the top right corner that resembles night table. The coast image exhibits similar shapes as observed in the previous model, with horizontal wavy lines indicating the presence of waves. However, for the remaining three classes, the visualizations become less visible. The forest image reveals some circular shapes, but their exact representation remains unclear. Similarly, the highway image exhibits zigzag patterns, but they do not strongly resemble elements associated with a highway. These indistinct patterns persist in the visualization of the industrial class as well. Overall, the interpretation of the visualizations generated by the rotation model proves to be more challenging, with less obvious

connections to the expected features of the respective classes. One possible explanation for this observation, as highlighted in the evaluation section, is that the rotation model tends to prioritize specific small regions. This emphasis on localized features could contribute to the less obvious and less easily interpretable visualizations.

The perturbation model yielded the least satisfactory visualizations among the three models. While each image displayed discernible shapes, none of them indicated any objects directly associated with the corresponding class. To understand the potential reasons for this, I referred back to the evaluation section. It became evident that the perturbation model prioritized wide spatial regions that were not necessarily linked to class-specific objects. Additionally, since the model was trained to detect perturbations, its focus was not geared towards capturing meaningful objects of interest.

Conclusion

To summarize, the analysis of the fully supervised task involved exploring various models, data preprocessing techniques, and learning parameters to find the best configuration. After a systematic evaluation, it was determined that a model using basic transformations performed better than one with advanced augmentation techniques, indicating that simpler transformations were sufficient for the dataset. Additionally, a shallow classifier architecture outperformed a deeper classifier, leading to the selection of the shallow network for the final model. Further experiments with loss function weighting, optimizer choice, and learning rate adjustment did not significantly improve performance. As a result, the original configuration with Cross Entropy loss with weights, ADAM optimizer, and a learning rate of 0.001 was retained.

In the self-supervised learning phase, two pipelines were developed: one focused on rotation classification as the pretext task and the other incorporated perturbation classification. The pretext models trained on rotation and perturbation achieved high accuracy, indicating successful feature learning. For the classification task, it was found that a shallow classifier with basic data augmentation yielded the best performance in both pipelines. The deeper classifier and advanced data augmentation did not significantly improve the results.

The evaluation revealed that fully supervised models outperformed the self-supervised counterparts, demonstrating a superior ability to learn precise features for accurate data scene classification. The visualizations generated during the model inversion process provided

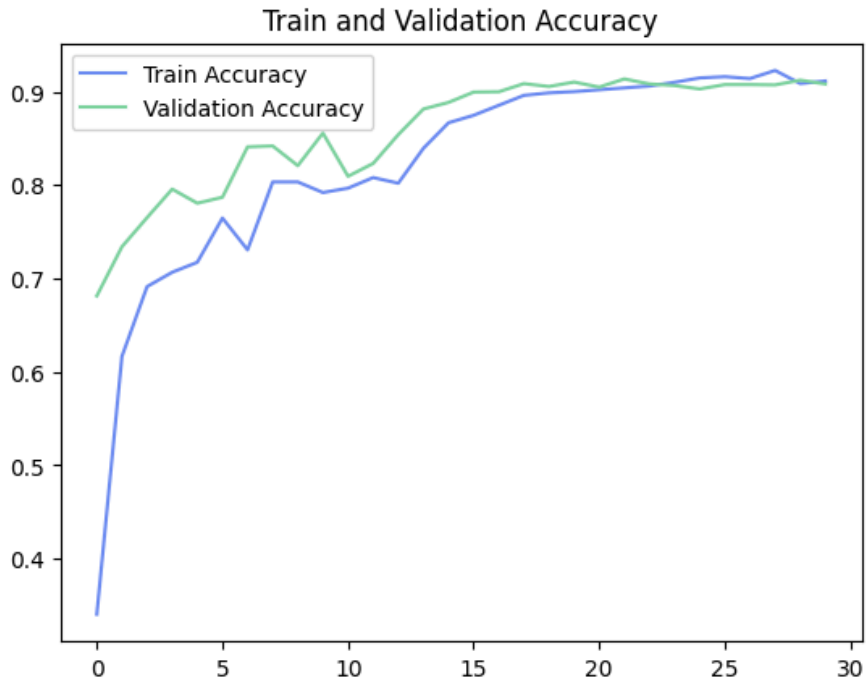
insights into the encoded features of each model. The supervised model exhibited visualizations that aligned well with the expected features for the selected classes, while the rotation and perturbation models presented less obvious and challenging visualizations. The rotation model showed a focus on small, localized regions, while the perturbation model prioritized wide spatial areas, resulting in less interpretable visualizations.

These findings highlight the importance of supervision and specific task training in achieving superior performance and meaningful feature representation. The project underscores the advantages of fully supervised models in learning precise and relevant features for classification tasks, as well as the limitations and challenges associated with self-supervised approaches, which can lead to lower performance and less interpretable visualizations due to the focus on diverse tasks.

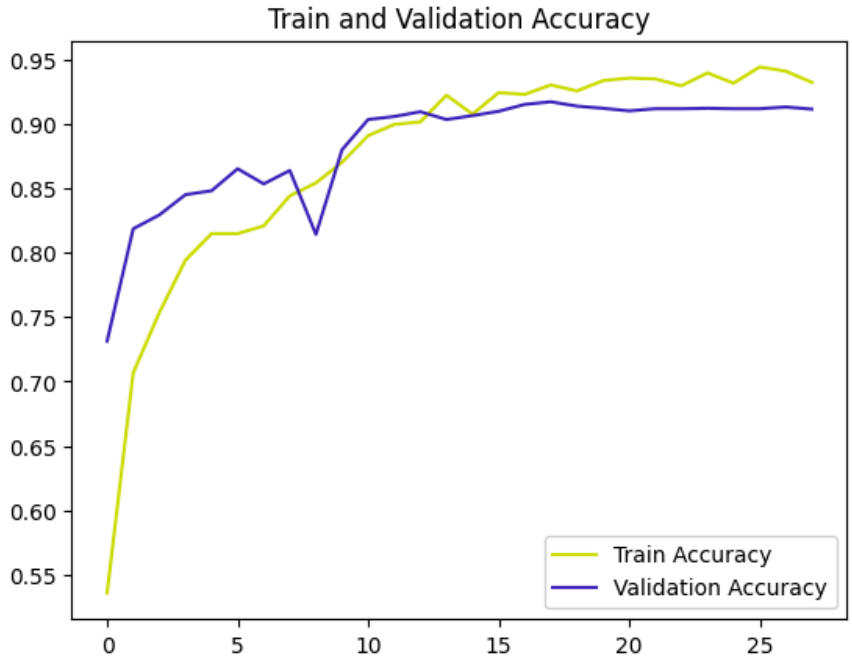
Overall, this project contributes to the understanding of model performance, feature extraction, and interpretation in neural networks. The insights gained can inform future research and applications in the field of computer vision and deep learning. The detailed findings and analysis of the fully supervised task and self-supervised learning pipelines can be found in the Jupyter Notebook scripts described in the Appendix 32.

Appendix

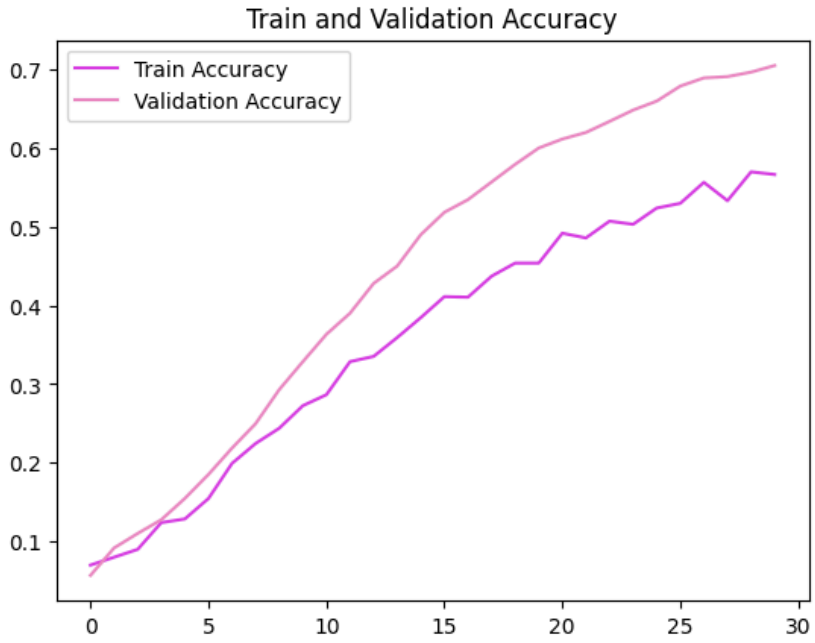
1. Training results of the supervised model with deep classifier.



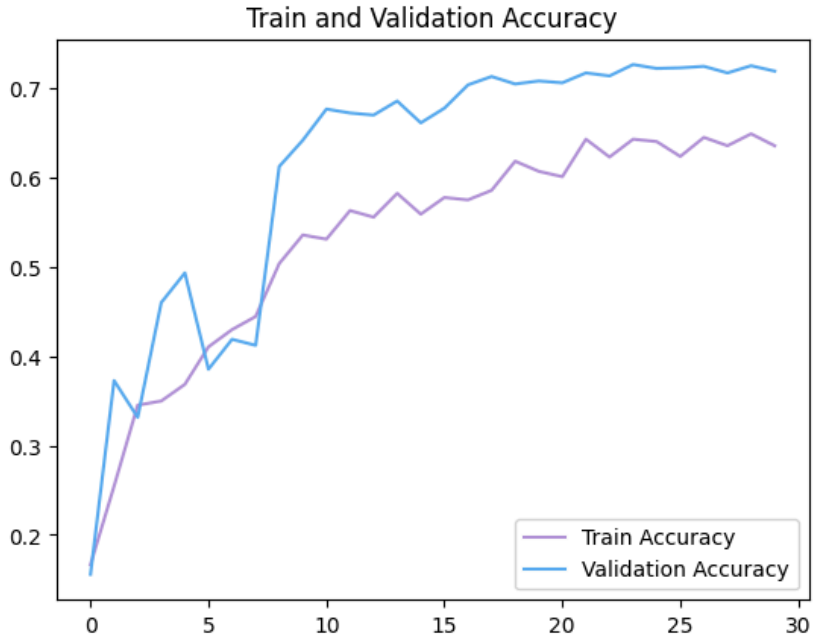
2. Training results of the supervised model with Cross Entropy with weights.



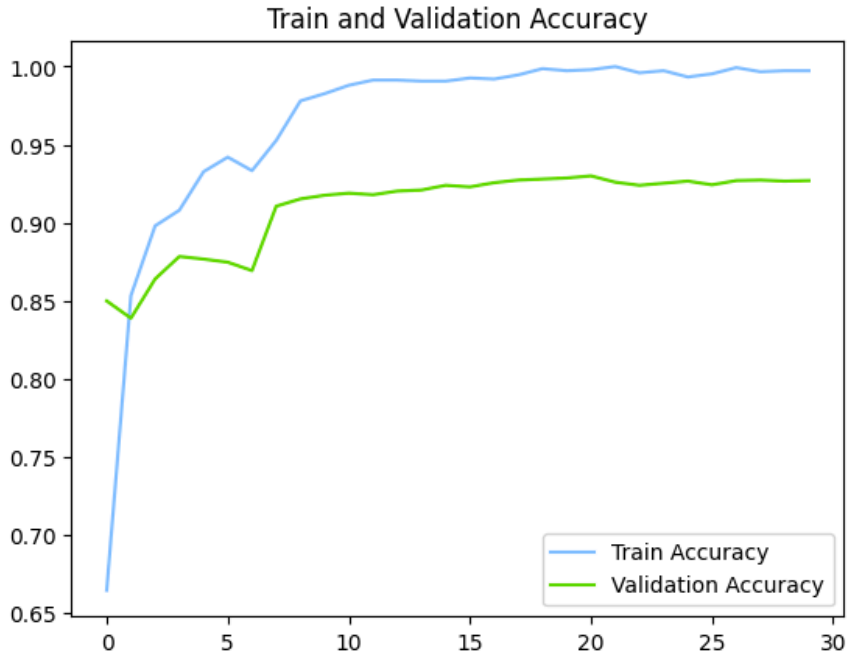
3. Training results of the supervised model using SGD optimizer.



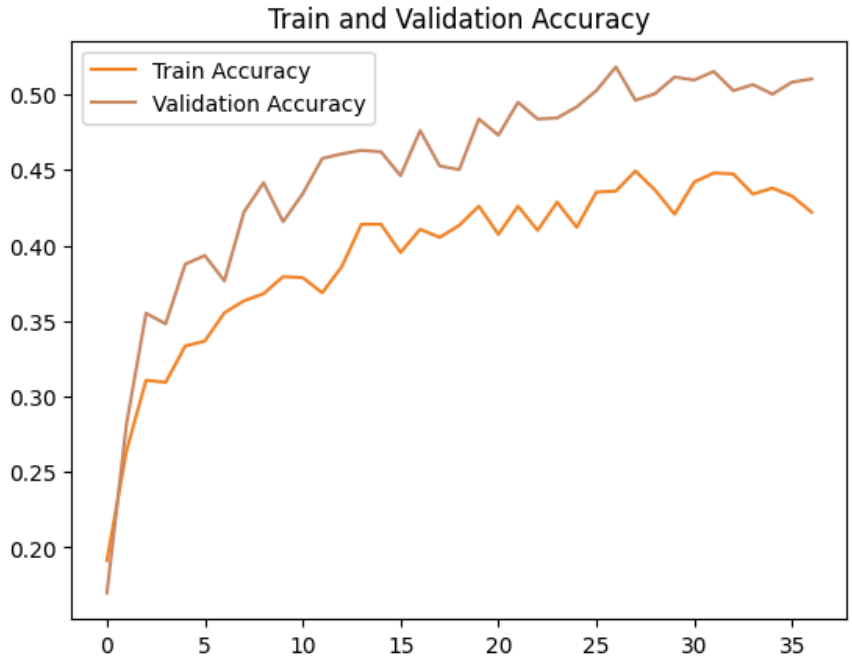
4. Training results of the supervised model using LR of 0.01.



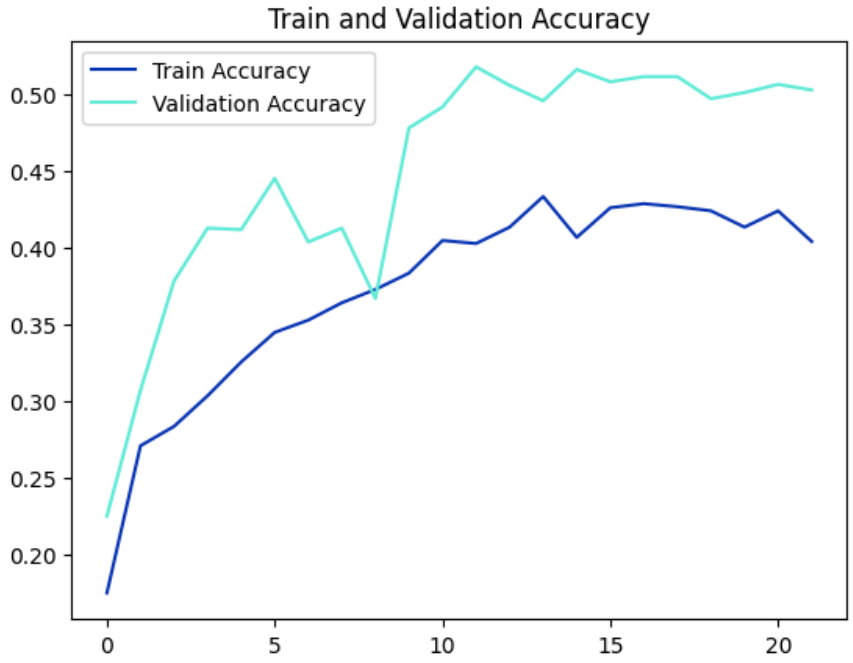
5. Training results of the final supervised model.



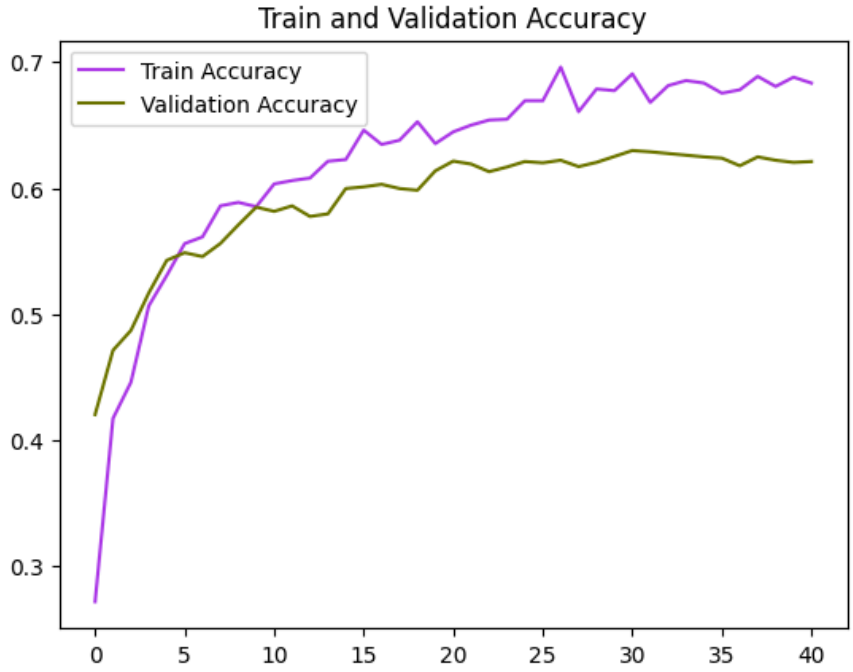
6. Training results of the first classification model from the pipeline 1.



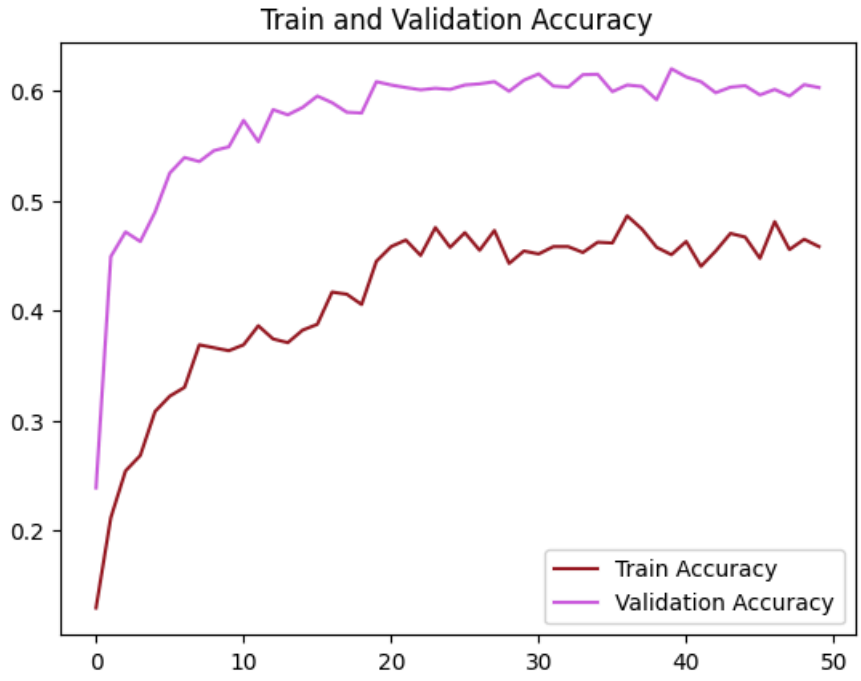
7. Training results of the second classification model from the pipeline 1.



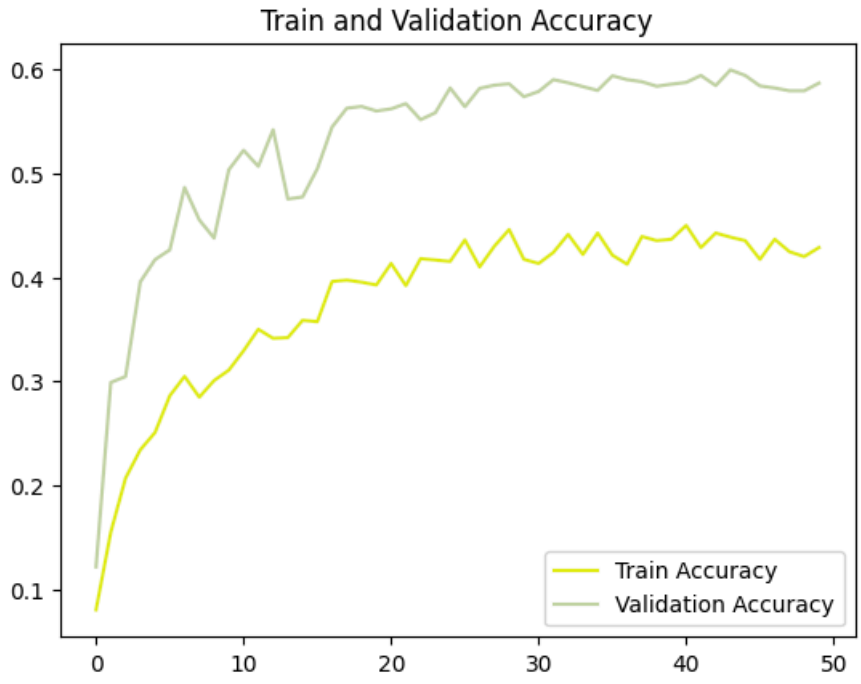
8. Training results of the third classification model from the pipeline 1.



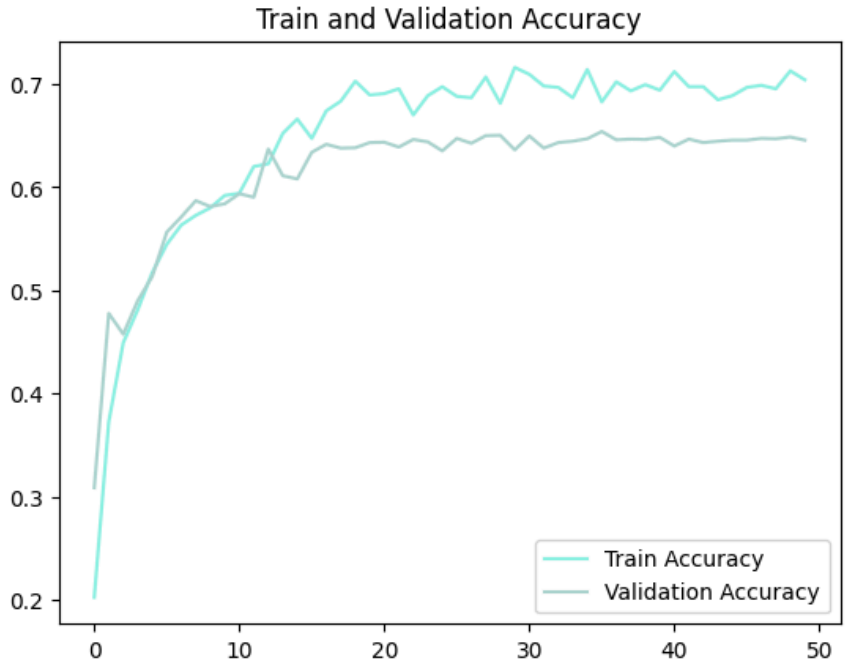
9. Training results of the first classification model from the pipeline 2.



10. Training results of the second classification model from the pipeline 2.



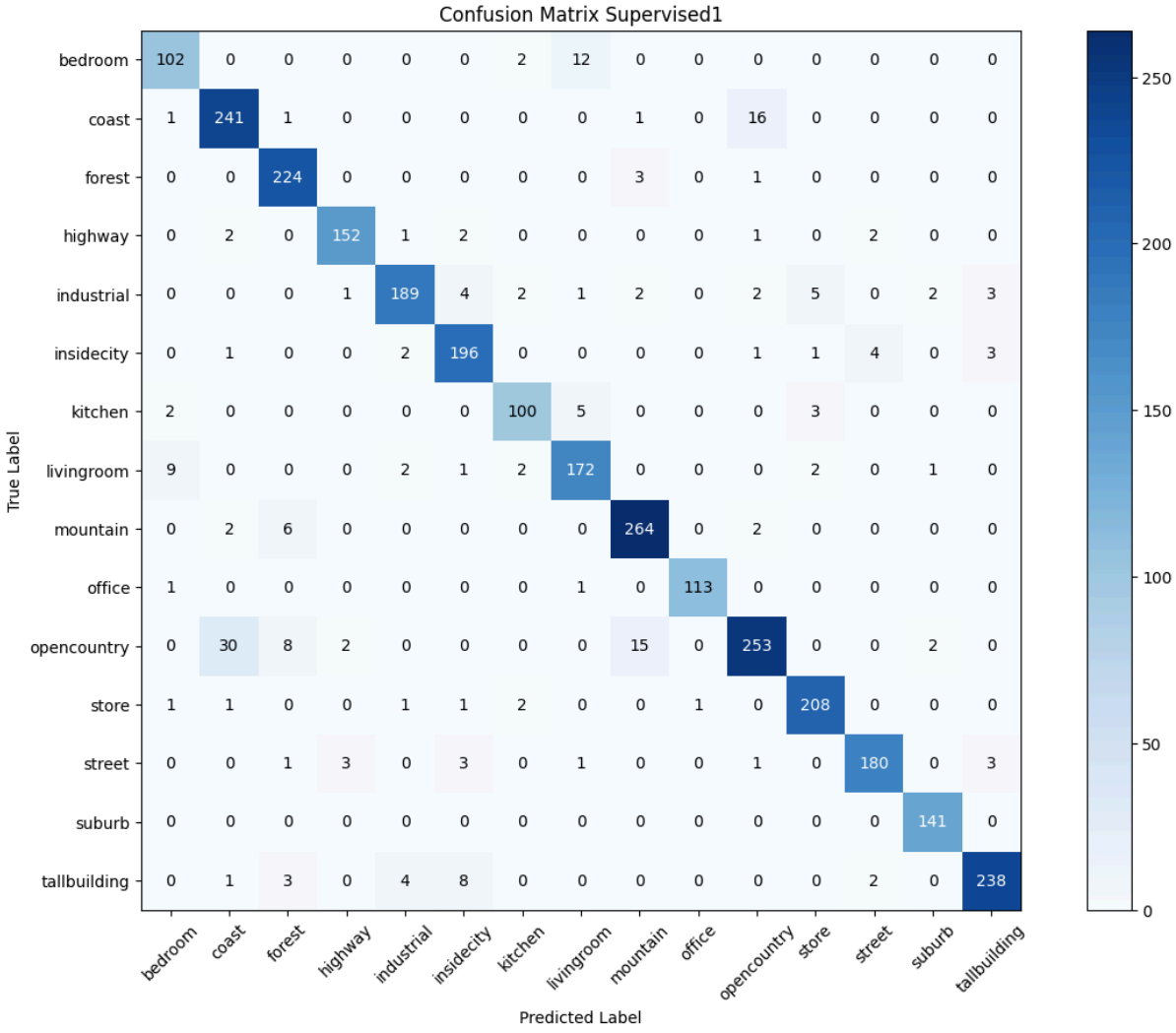
11. Training results of the third classification model from the pipeline 2.



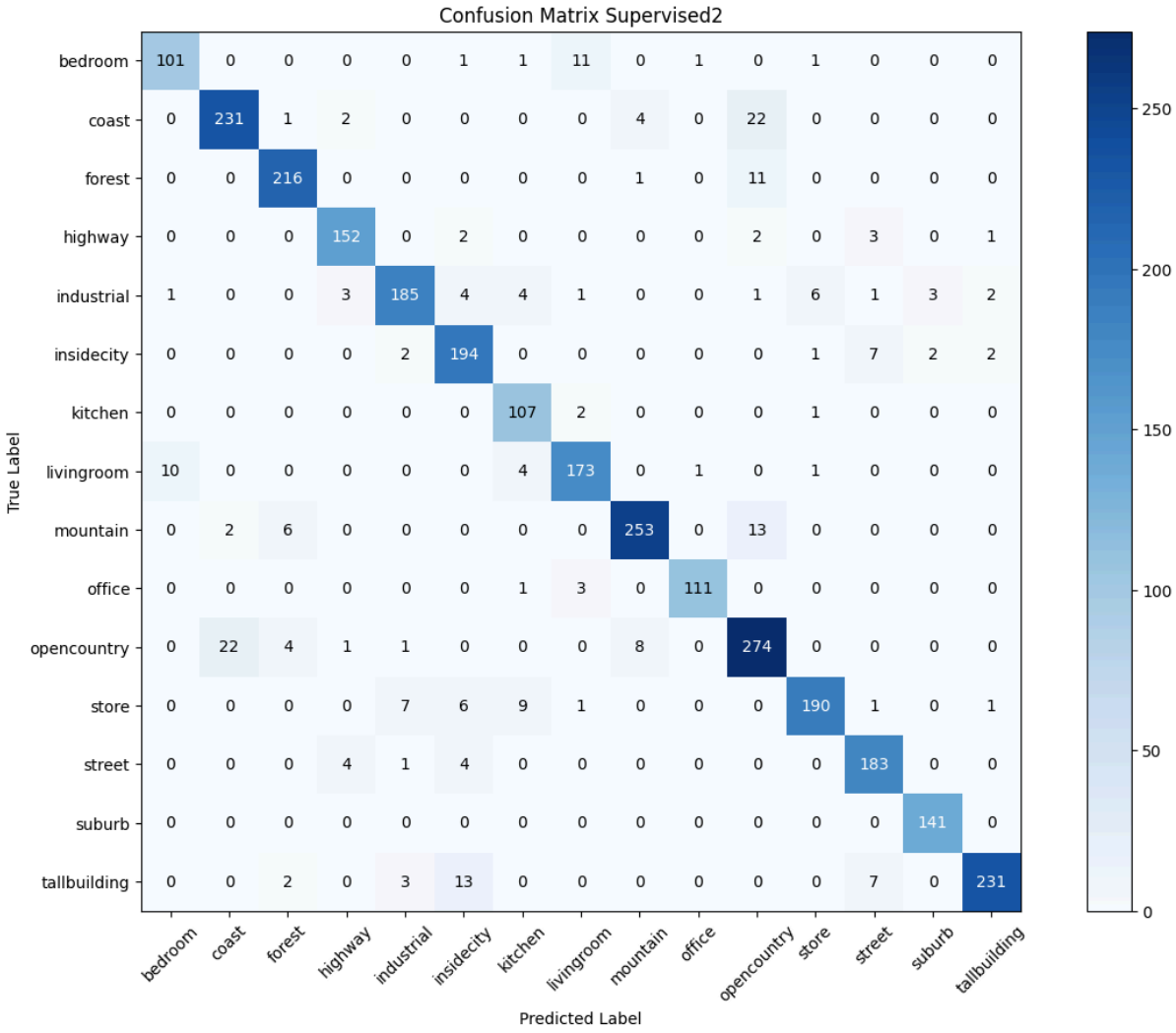
12. Settings table.

Model	Image size	Learning rate	Optimizer	Loss Function	Batch size	Nr of epochs	Nr of FC-layers	Transformers
Supervised 1	224x224	0.001	ADAM	Cross Entropy	32	30	1	Basic
Supervised 2	224x224	0.001	ADAM	Cross Entropy	32	30	1	Advanced
Supervised 3	224x224	0.001	ADAM	Cross Entropy	32	30	3	Advanced
Supervised 4	224x224	0.001	ADAM	Cross Entropy (weights)	32	28	1	Advanced
Supervised 5	224x224	0.001	SGD	Cross Entropy	32	30	1	Advanced
Supervised 6	224x224	0.01	ADAM	Cross Entropy	32	30	1	Advanced
Supervised Final	224x224	0.001	ADAM	Cross Entropy (weights)	32	30	1	Advanced - without Cropping
Rotation 1	224x224	0.001	ADAM	Cross Entropy	32	30	1	Basic
Rotation–classifier 1	224x224	0.001	ADAM	Cross Entropy	32	37	1	Advanced
Rotation–classifier 2	224x224	0.001	ADAM	Cross Entropy	32	22	3	Advanced
Rotation–classifier 3	224x224	0.001	ADAM	Cross Entropy	32	41	1	Basic
Perturbation 1	224x224	0.001	ADAM	Cross Entropy	32	30	1	Basic
Perturbation-classifier 1	224x224	0.001	ADAM	Cross Entropy	32	50	1	Advanced
Perturbation-classifier 2	224x224	0.001	ADAM	Cross Entropy	32	50	3	Advanced
Perturbation-classifier 3	224x224	0.001	ADAM	Cross Entropy	32	50	1	Basic

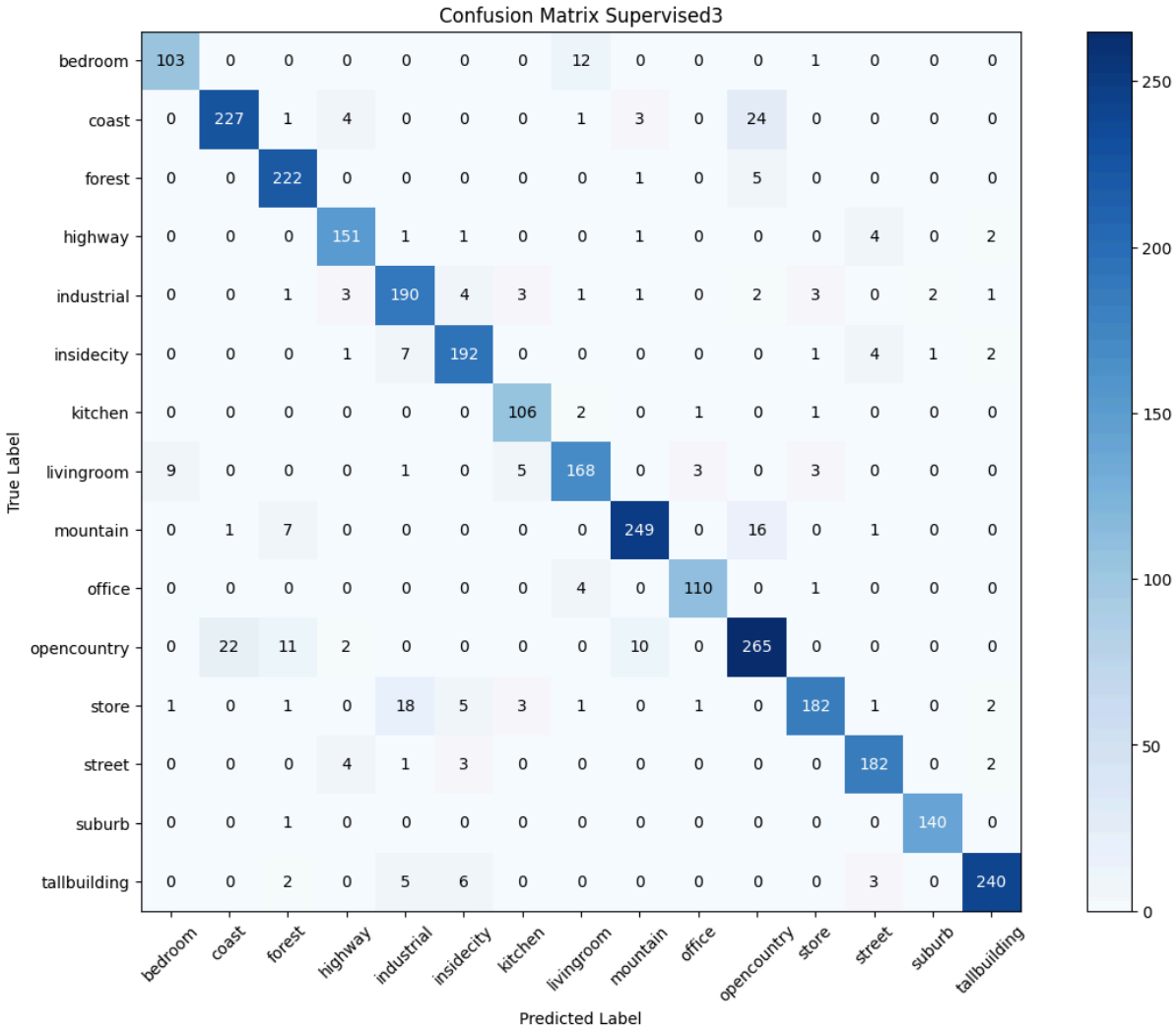
13. Confusion matrix of the supervised 1 model.



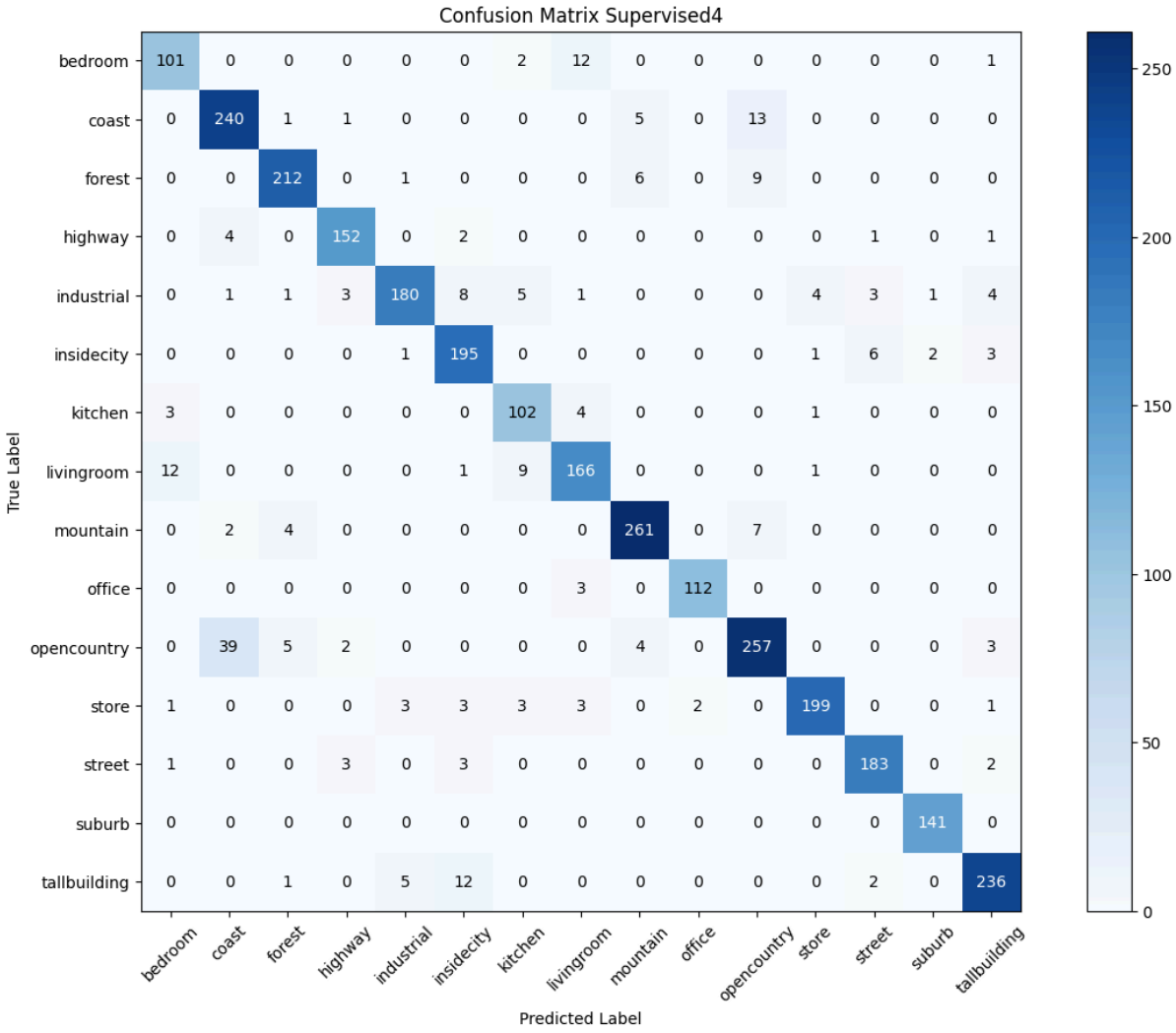
14. Confusion matrix of the supervised 2 model.



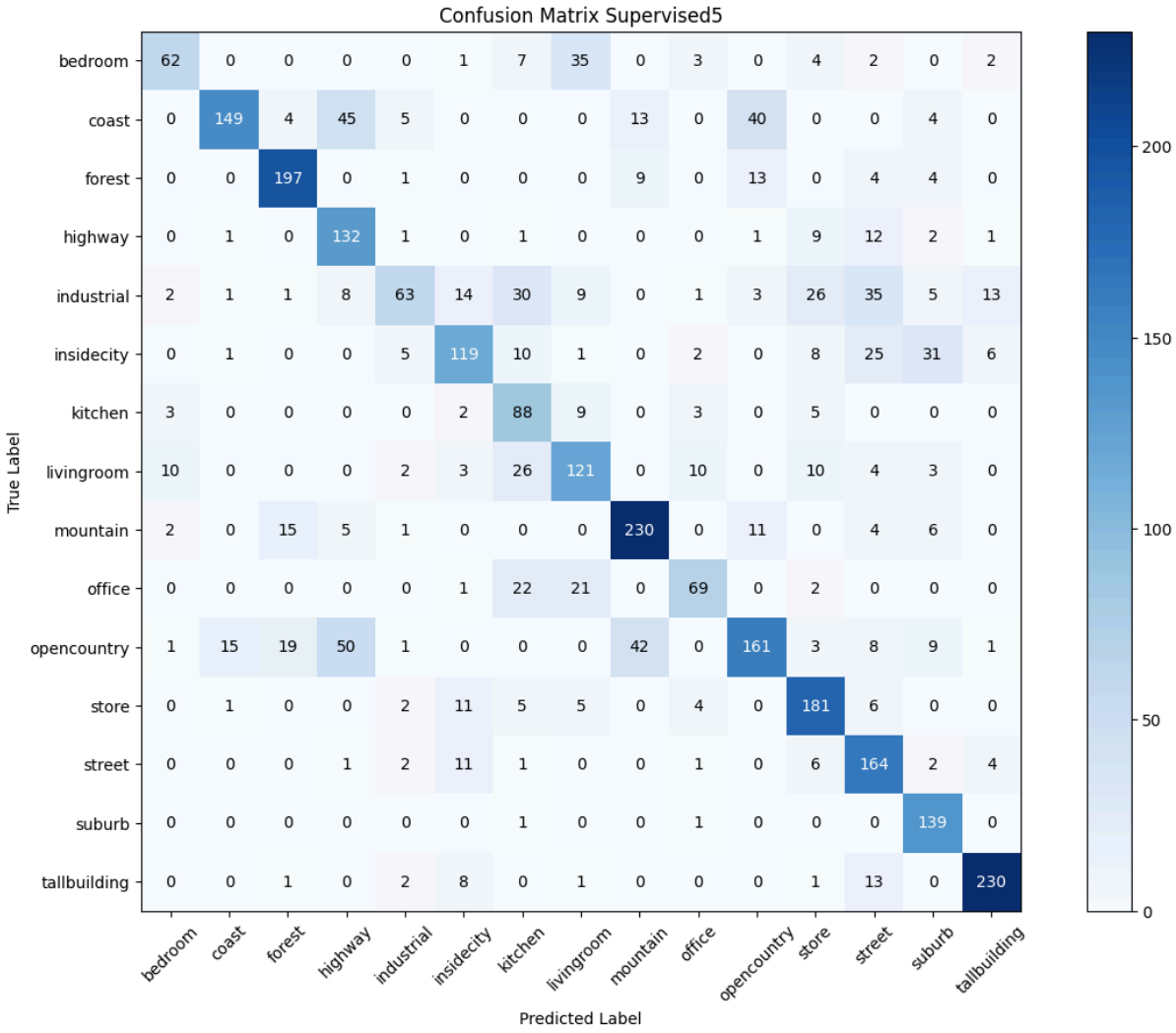
15. Confusion matrix of the supervised 3 model.



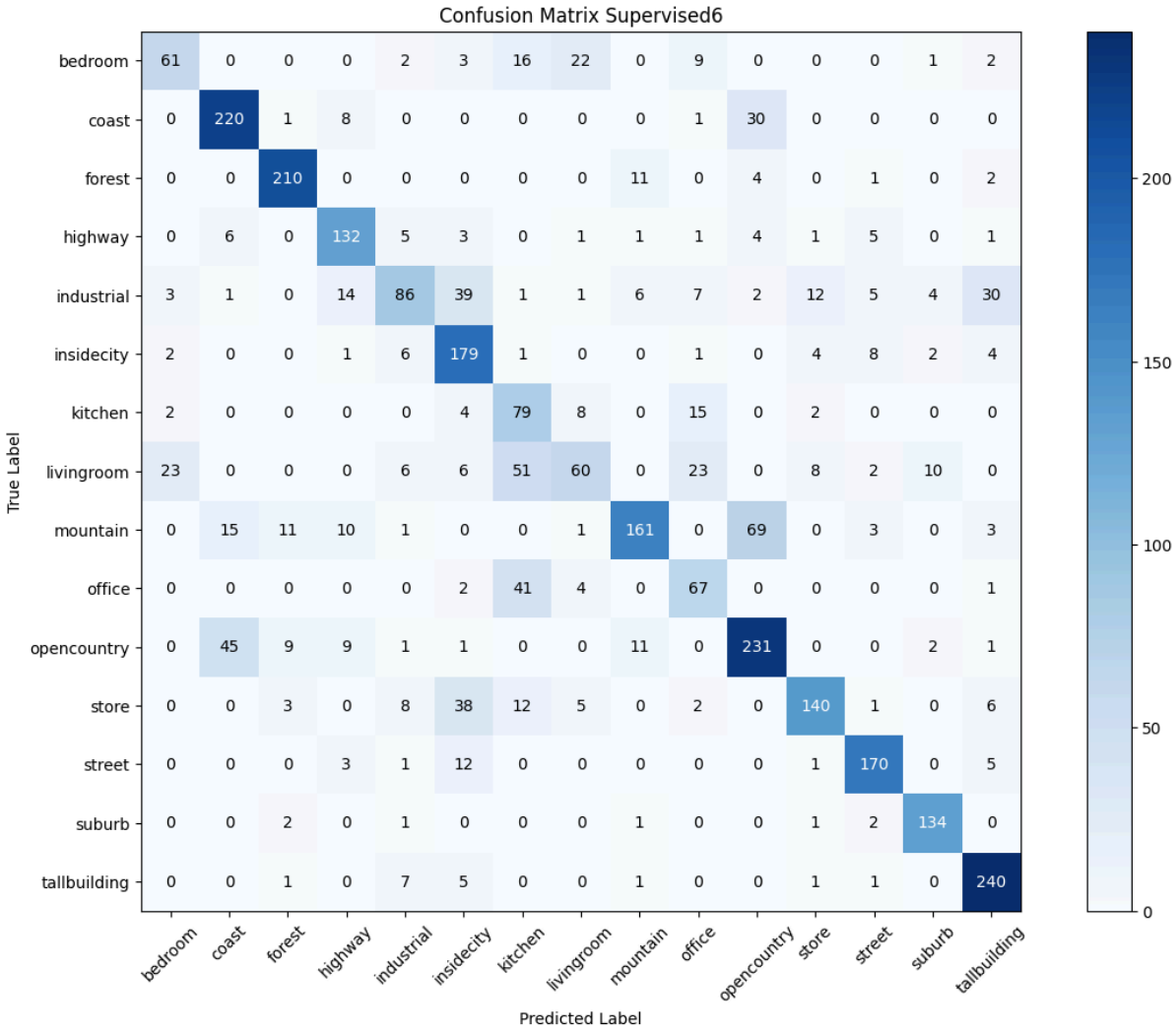
16. Confusion matrix of the supervised 4 model.



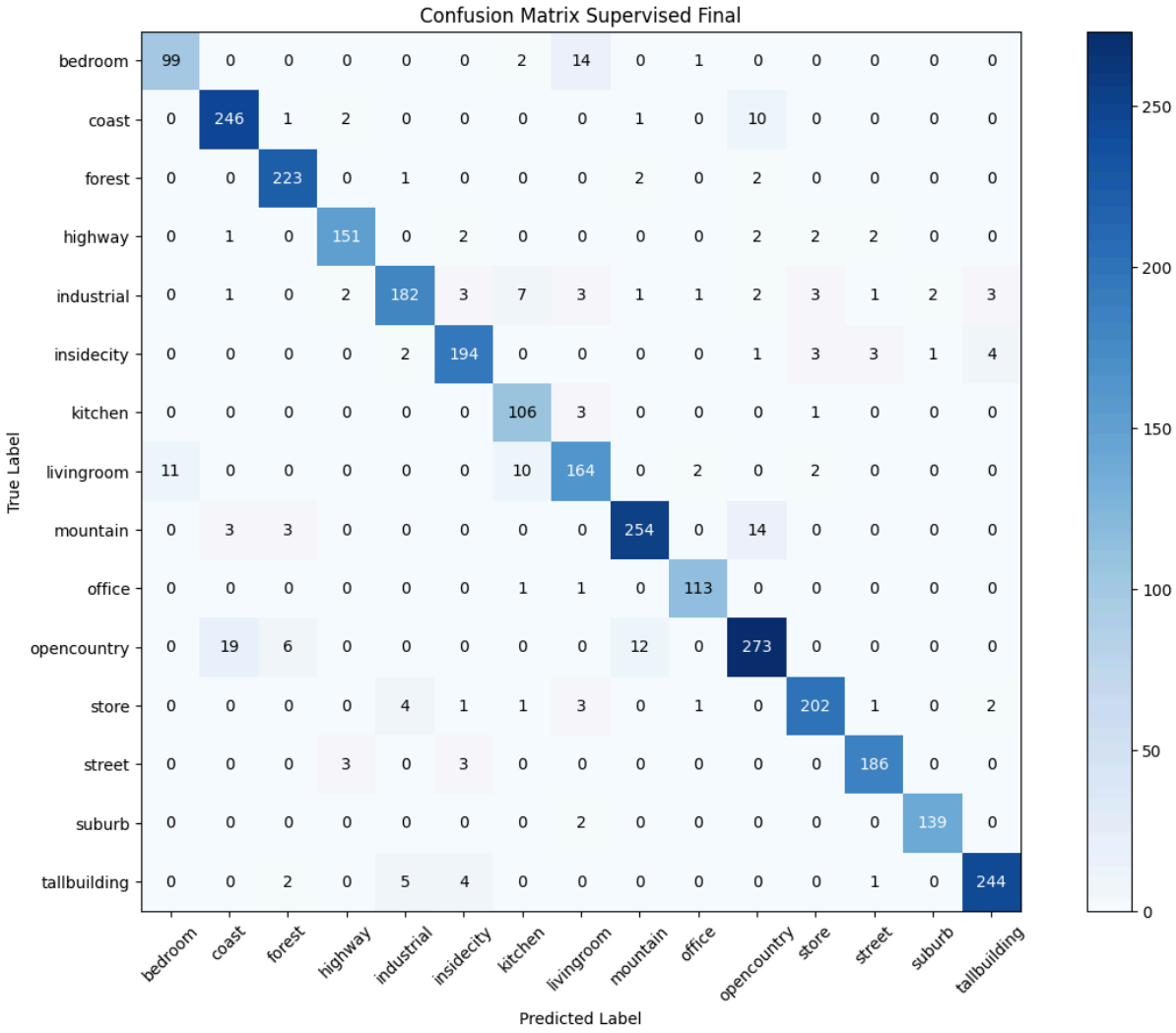
17. Confusion matrix of the supervised 5 model.



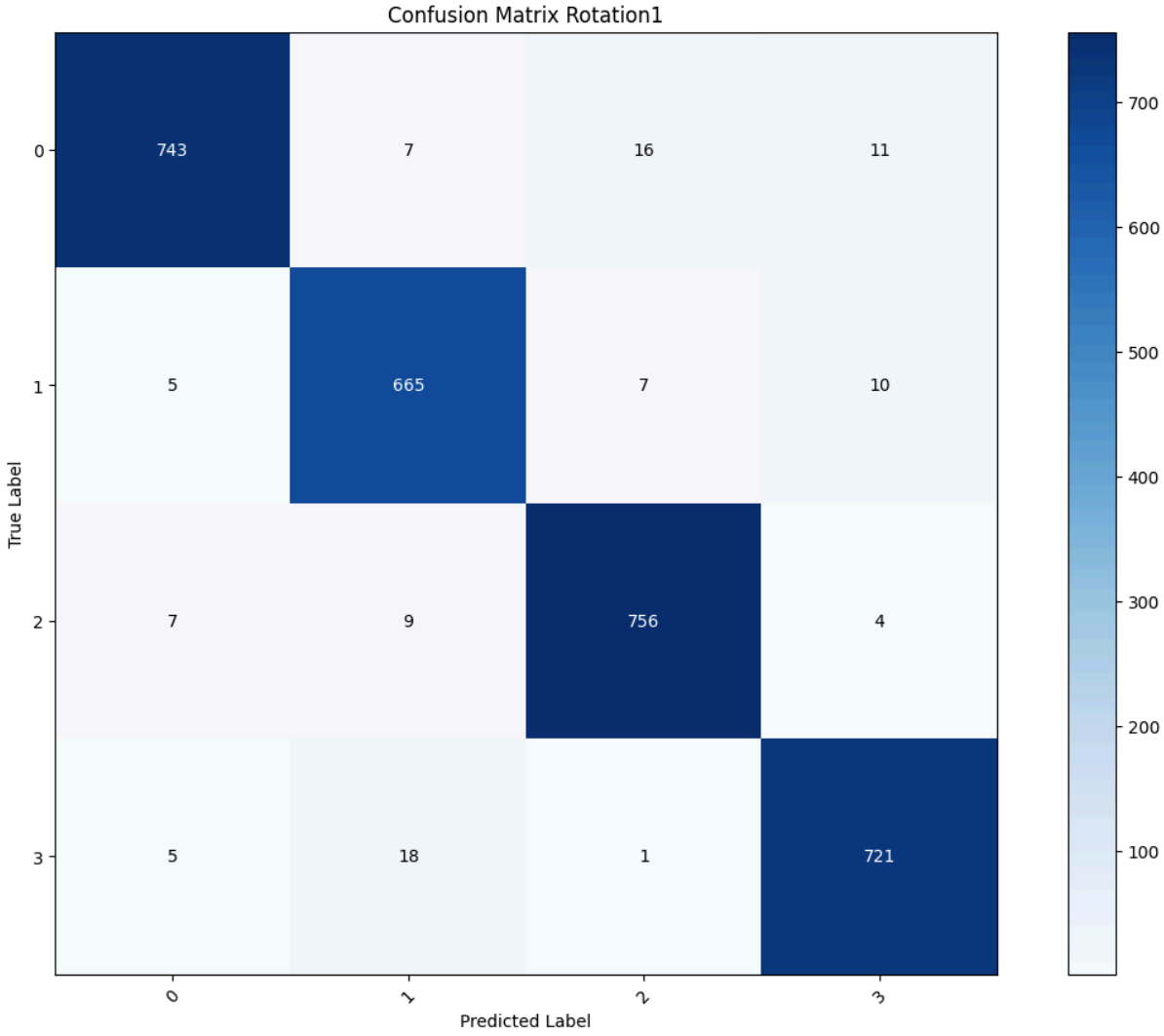
18. Confusion matrix of the supervised 6 model.



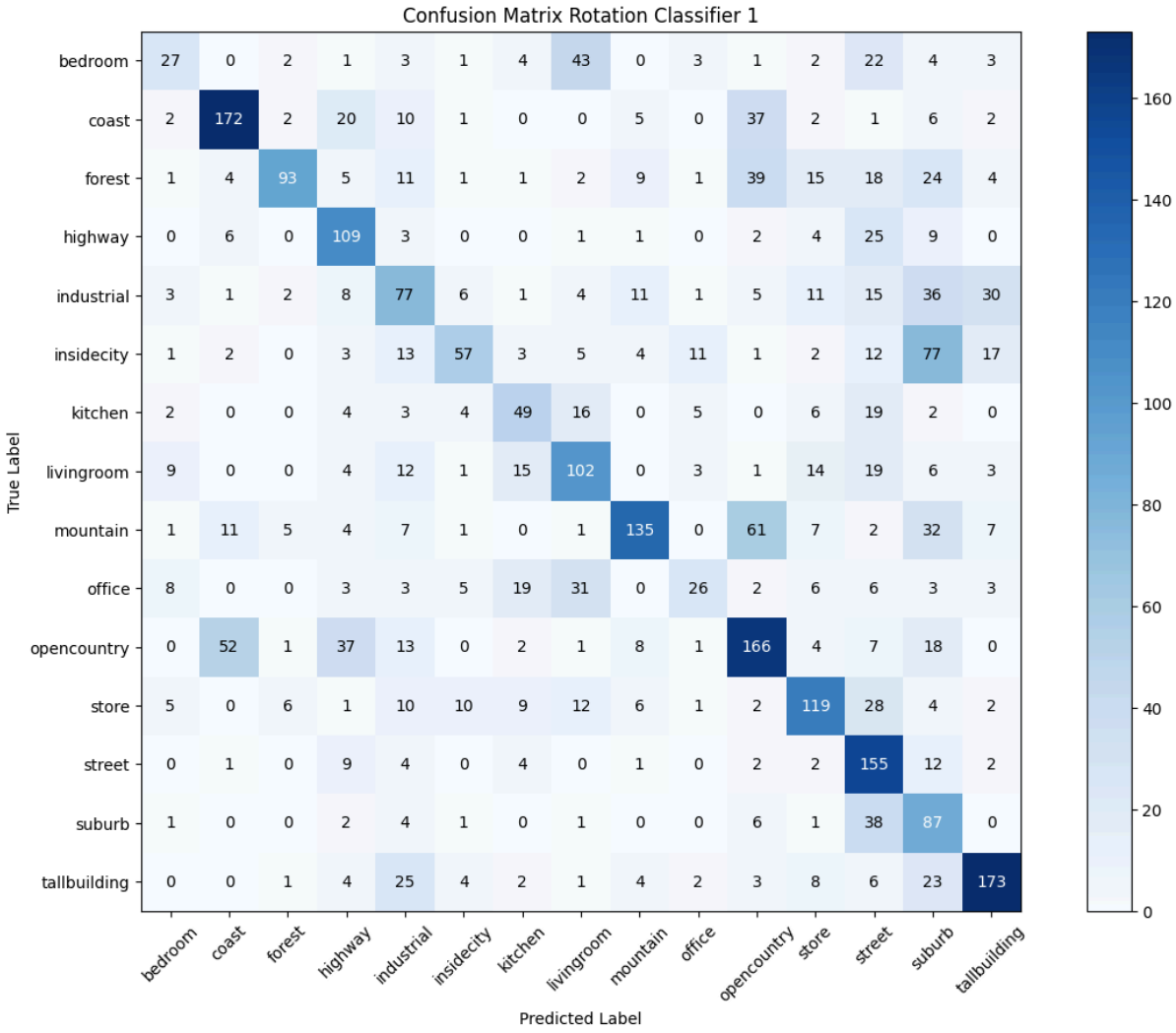
19. Confusion matrix of the final supervised model.



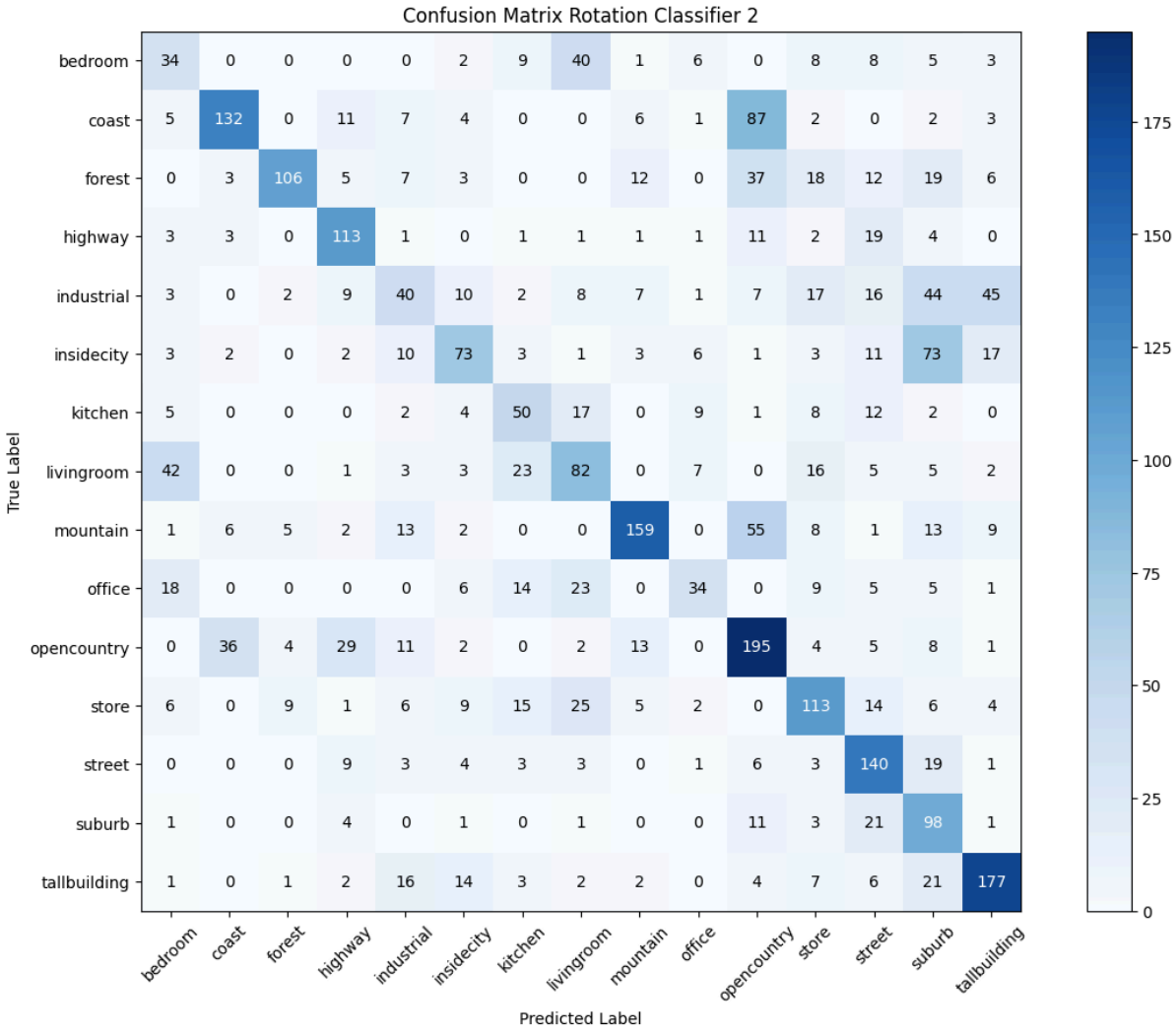
20. Confusion matrix of the rotation classification.



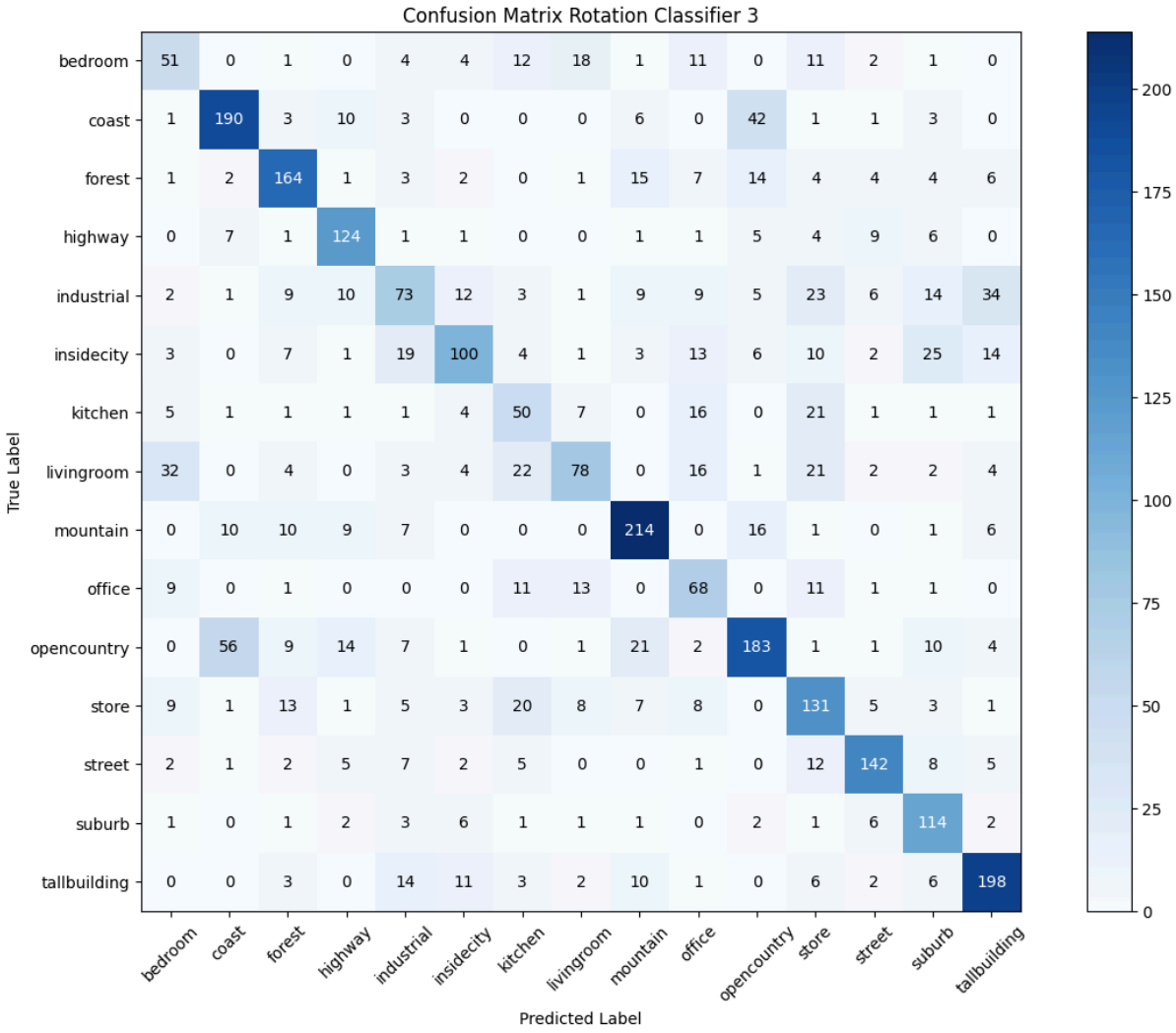
21. Confusion matrix of the first model for rotation scene classification.



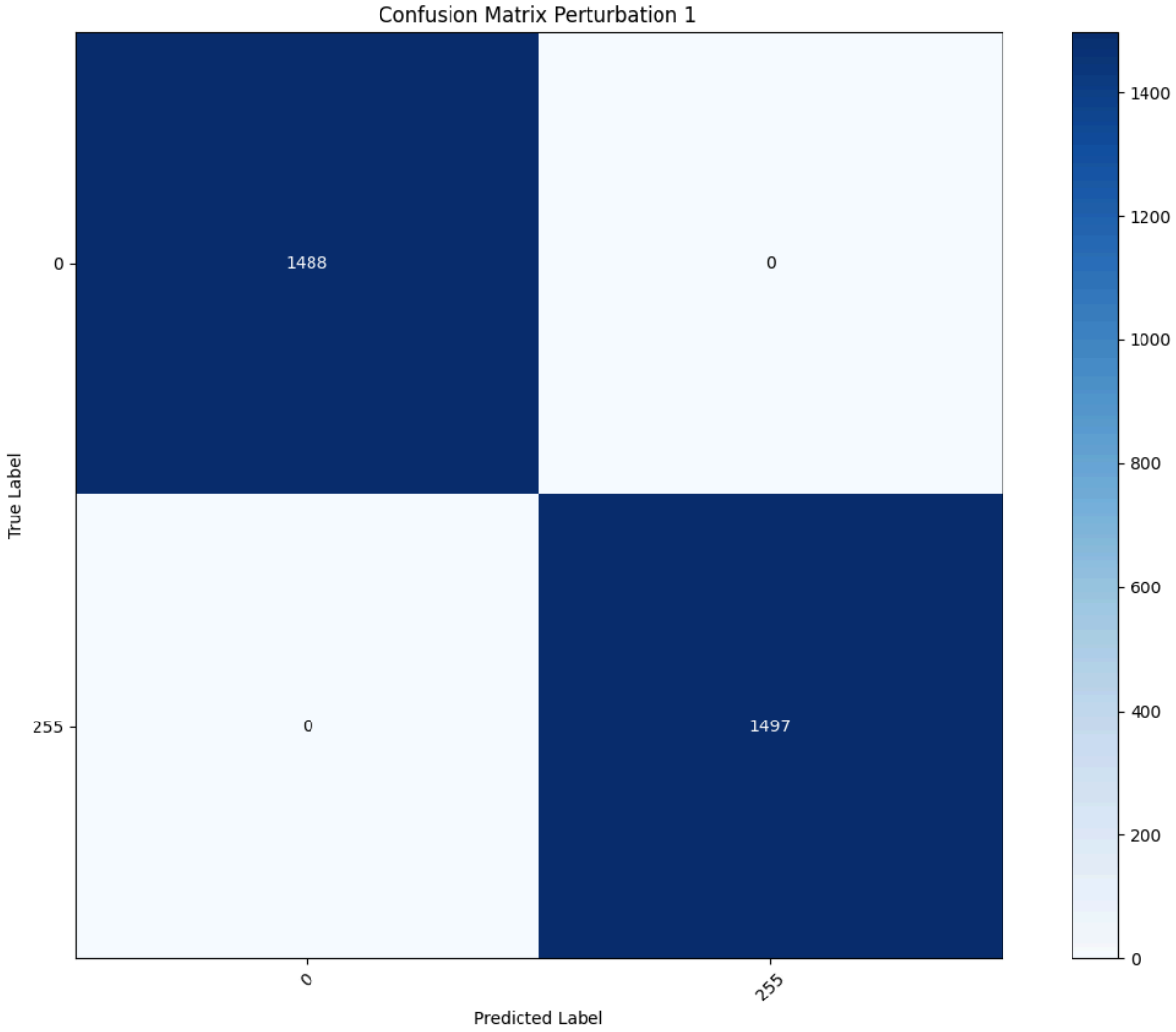
22. Confusion matrix of the second model for rotation scene classification.



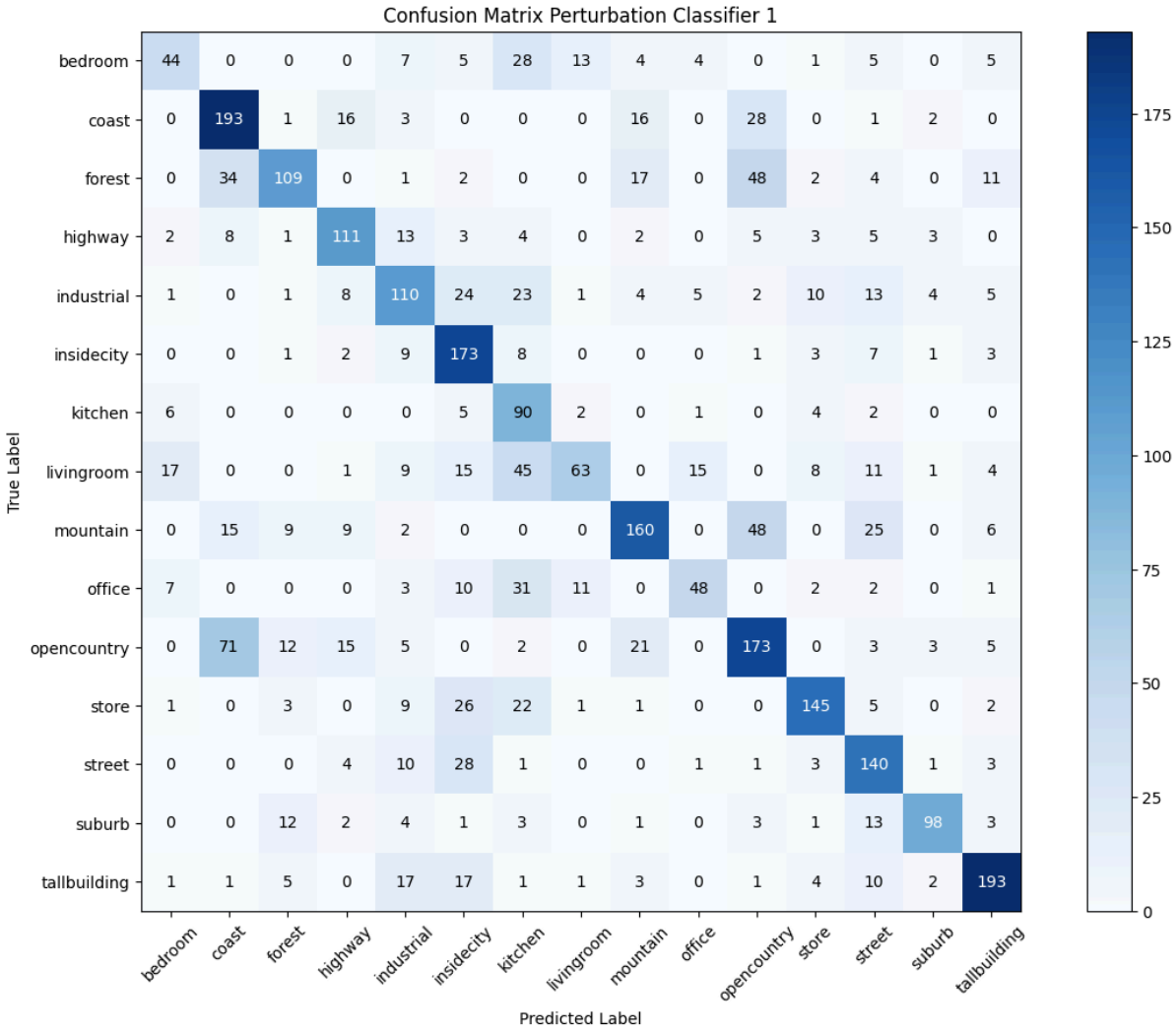
23. Confusion matrix of the third model for rotation scene classification.



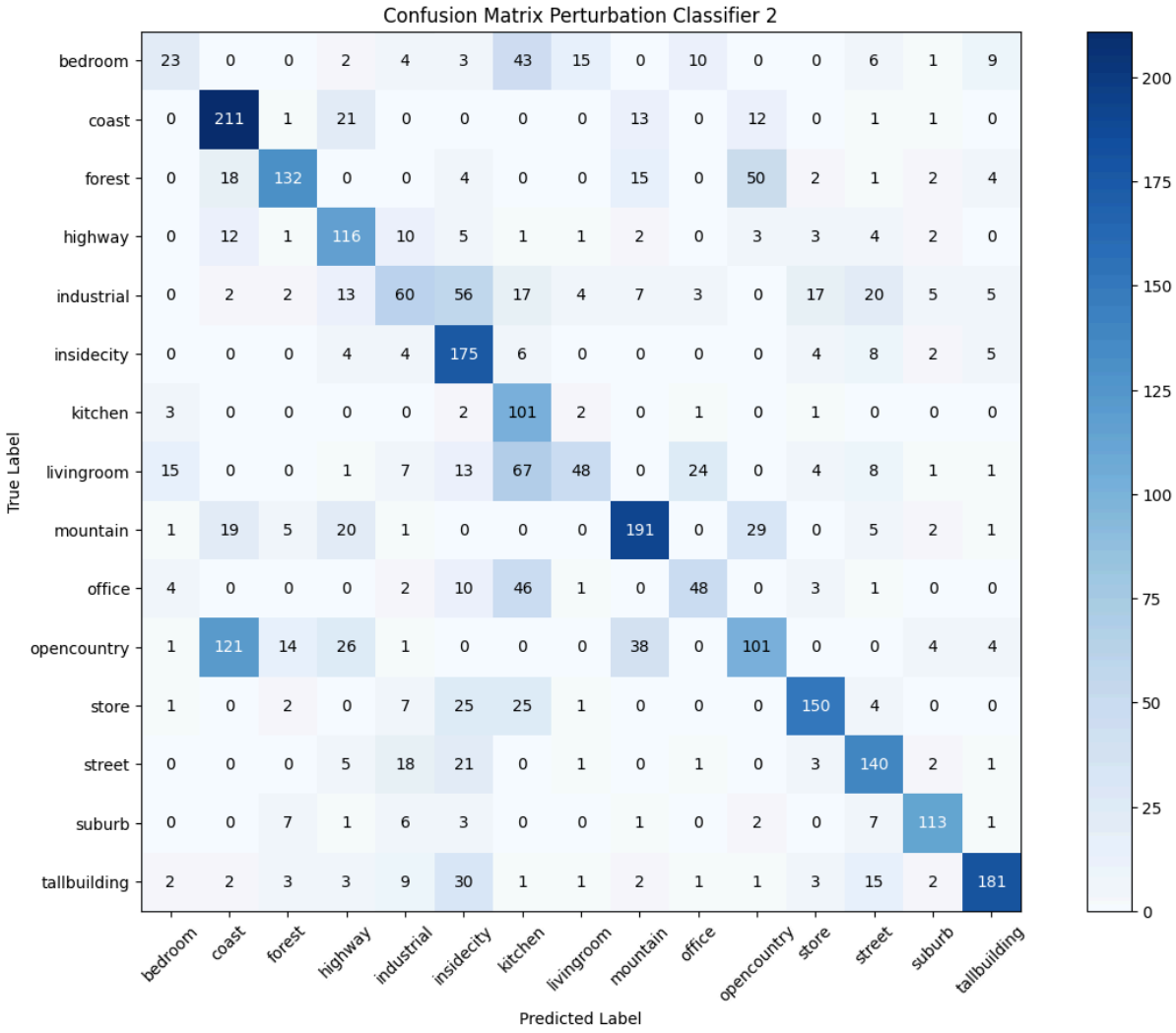
24. Confusion matrix of the perturbation classification.



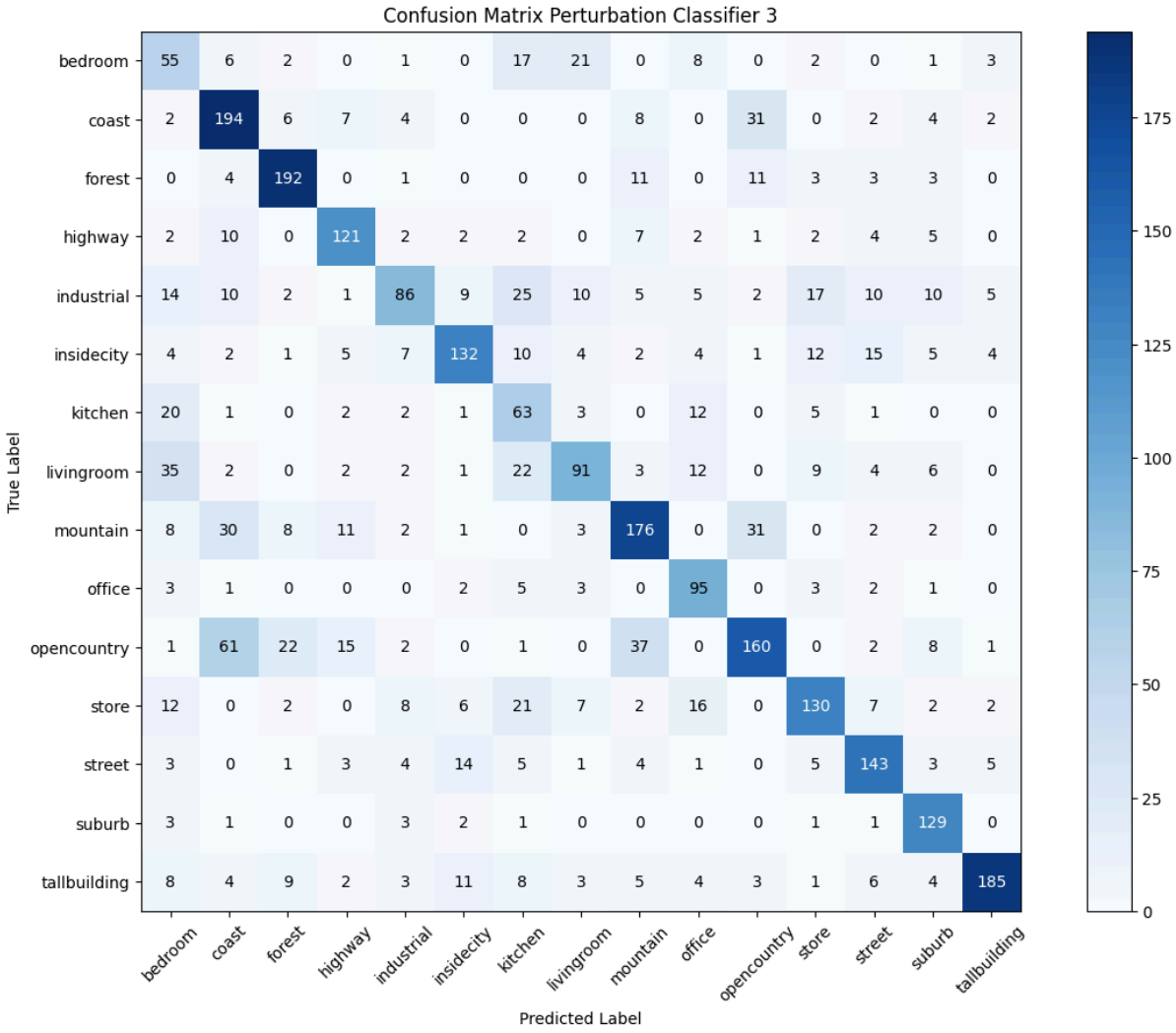
25. Confusion matrix of the first model for perturbation scene classification.



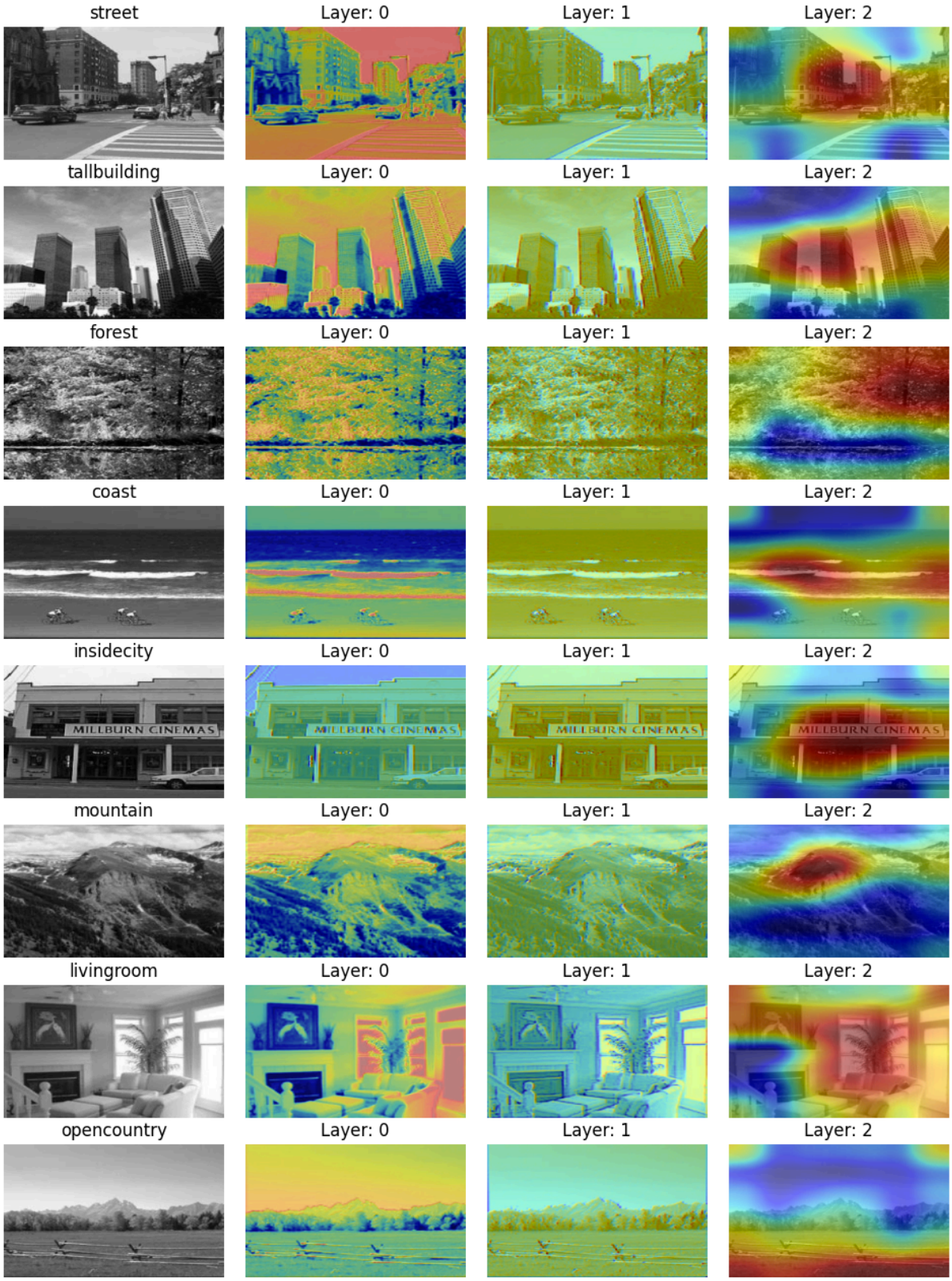
26. Confusion matrix of the second model for perturbation scene classification.



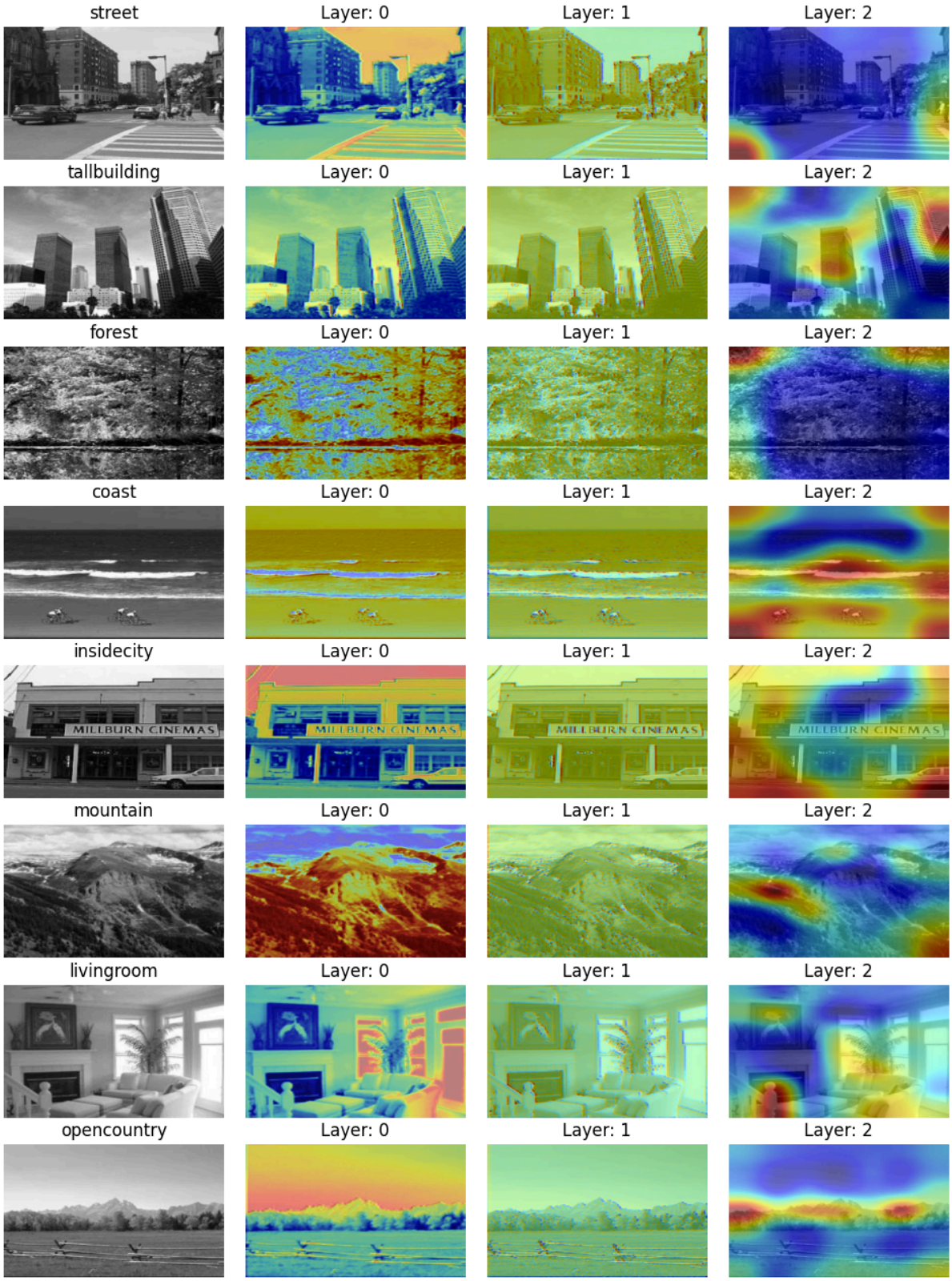
27. Confusion matrix of the third model for perturbation scene classification.



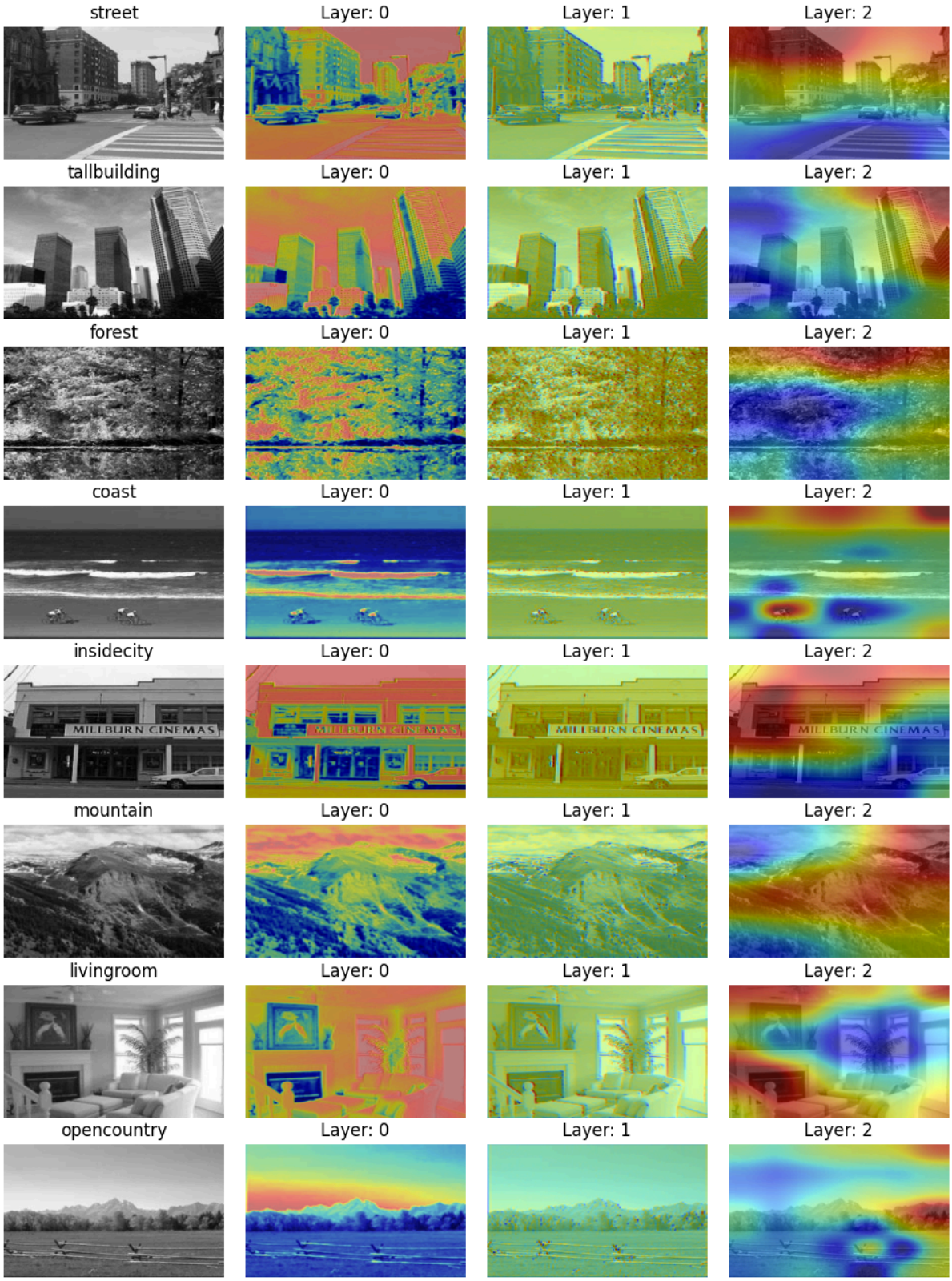
28. Score-Cam results for final supervised model.



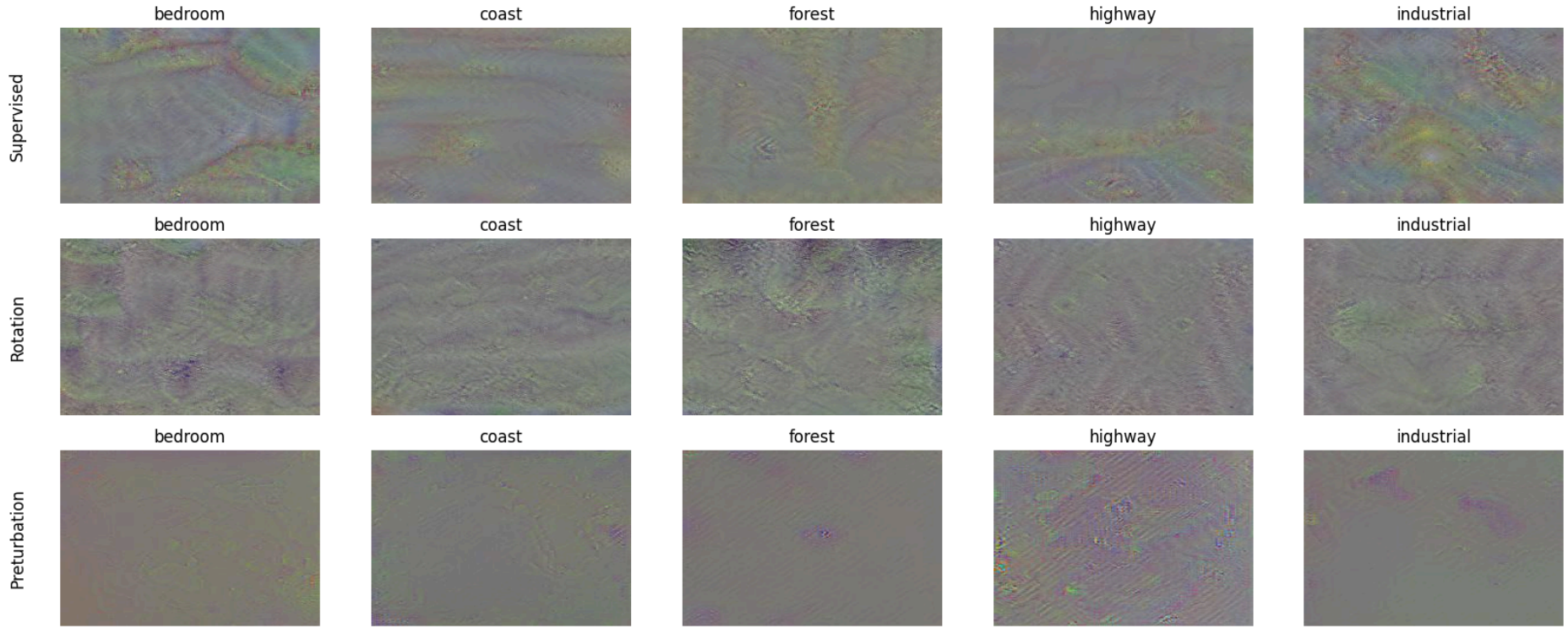
29. Score-Cam results for final rotation model.



30. Score-Cam results for final perturbation model.



31. Model inversion results.



32. Project Structure:

```
├── data                                          - working data
├── evaluation
│   ├── RegularizedUnitSpecificImageGeneration.py - script for model interpretation
│   ├── evaluation.ipynb                          - main script for evaluation part
│   ├── generated                                 - images generated for model interpretation
│   ├── misc_functions.py                         - functions used by model inversion script
│   ├── scorecam.py                               - score-cam script
│   └── scorecam_images                           - images for score-cam
├── helper_functions.py                           - helper functions for training
├── models                                        - all models
├── results                                       - validation scores for all models
├── self_supervising_task
│   ├── dataloader.py                             - functions to load data (self-supervised)
│   ├── models.py                                 - self-supervised models
│   ├── report.ipynb                              - main script for training (self-supervised)
│   └── settings.py                               - settings self-supervised training
└── supervising_task
    ├── dataloader.py                             - functions to load data (supervised)
    ├── models.py                                 - supervised models
    ├── report.ipynb                              - main script for training (supervised)
    └── settings.py                               - settings supervised training
```