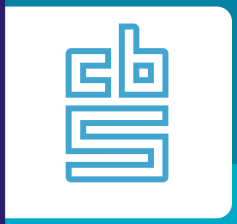


Center for
Big Data Statistics

Working paper

DeepSoLim

Detecting Solar Panels on Aerial Images of Limburg with Convolutional Neural Networks

Working paper no.: 07-20

Han van Leeuwen

February 2022

Contents

Abstract

Earth observation by planes and satellites provide imagery data that may complement survey and register data. Convolutional neural networks (CNNs) are a class of deep learning algorithms that can automatically learn features from image data. DeepSolaris (Jong et al., 2020) was a European project where CBS applied this technique for the first time to exploit aerial image data. The main aim of the project was to train CNNs to detect solar panels, as a first step towards the more general question of how much power is generated by solar panels in Europe. This paper is a follow-up study where we investigate how to improve model performance and show the sensitivity of the model to noise, sample size, class imbalance, class weights, data augmentation and weight initialization, offering provisional best practices for a quick start in exploiting earth observation.

1 Introduction

Statistics have traditionally been based on information retrieved by sample surveys. Sample surveys allow for making valid inference about the population using a relatively small sample. In addition, the uncertainty of estimators can be quantified. However, sample surveys come with limitations: they suffer from non-response, resulting in higher variance and potentially biased estimates; they require a tedious amount of planning and processing, resulting in high costs and slow dissemination; they inflict response burden, resulting in lower data quality. These form clear incentives to look into alternatives that are cheaper, faster and less burdensome. Earth observation by planes and satellites provide imagery data that may complement survey and register data. Convolutional neural networks (CNNs) are a class of deep learning (DL) algorithms that can automatically learn features from image data. DeepSolaris (Jong et al., 2020) was a European project where CBS applied this technique for the first time to exploit aerial image data. The main aim of the project was to train CNNs to detect solar panels, as a first step towards the more general question of how much power is generated by solar panels in Europe.

Some important lessons were learned: Images with 50-cm resolution proved to be of insufficient quality. Images were so blurry that solar panels could not be distinguished. Satellite data covering Europe could therefore not be used and the researchers used higher-resolution images taken by airplanes instead. Aerial images are not available for all regions in Europe, which meant that their initial scope of Europe proved to be too broad. Aerial images with 25-cm resolution of the Netherlands are freely accessible and can be used without cost, with several consecutive years of image datasets available. These datasets proved to be workable, however solar panels were still hard to distinguish from background noise but at least models could be trained even though their performance was poor. What remained was using 10-cm resolution aerial images that cover the southern part of Limburg, a province of the Netherlands.

Another observation was that most models trained on images from a different region than the classification task at hand, would perform considerably worse. For this reason, both a validation dataset of the same region and a test set with images from a different region should be used to measure a model's performance. For automatic solar detection on aerial images, the resulting models of the DeepSolaris project were promising, though they lacked the performance necessary. Their best performing model had a macro average f1-score of 95% on the

validation set and 88% on the test set, where both training, validation and test set came from the same dataset and region. Models were also tested on a different geographic area, where the best performing model had a macro average f1-score of 61%.

With an increasing emphasis on statistics based on aerial images, we also perform several scenarios to lay a foundation for future research, saving time and effort in search for optimal hyperparameters for high performance models. Each scenario looks at a different aspect of model building and assesses their impact. Perhaps the most important aspect of machine learning is data. The amount of data offered by aerial photo cameras can potentially be vast, but they also come with a cost. Knowing the effect of sample size on model performance, could offer insight in how much data is required for a certain goal. Additionally, these results may prove to be valuable when upscaling the scope of this project to include other regions such as the Netherlands or Europe.

Data augmentation reduces the degree of overfitting and improves the model's ability of generalization (Zhang et al., 2015). However data augmentation comes in many forms and it is uncertain which techniques offer benefits, do nothing or are counterproductive when using aerial images. Yosinski et al. (2014) shows that transfer learning can potentially improve the resulting models rather than using randomly initialized weights. One important difference from their study is that ImageNet images were used to train both the original 'transfer' weights and the new weights of the resulting model. Aerial images offer a top-down perspective of landscape, which is different from the object-oriented images in the ImageNet dataset. Weights learned from street view images may or may not help in detecting solar panels on aerial view images.

Solar panels have become increasingly popular, yet houses where they remain absent are larger in number than houses where solar panels are present. When using aerial photographs as source, this imbalance is also seen in the dataset. This class imbalance may have a profound influence on the model performance, which is why it might be fruitful to look into it. Class imbalance is somewhat counteracted through class weights.

In the experiments throughout this paper, models are both trained and evaluated on the city of Heerlen dataset. Meaning, during training the model gets evaluated based on how well it does on images from Heerlen. Is it more reasonable to evaluate the model based on images of the target area?

In this report we aim to answer the following questions:

1. With the current state of technology, how well can a model detect solar panels on aerial images in Limburg, in such capacity that it can be used as a statistical tool? The lower bound of pure chance should at least be achieved.
2. Scenarios:
    a. What is the effect of sample size on model performance?
    b. What are the effects of data augmentation on model performance?
    c. What is the effect of ImageNet weights on model performance?
    d. What is the effect of class imbalance on model performance?
    e. What is the effect of adjusted class weights on model performance?
    f. What is the effect of using the South Limburg region as evaluation dataset on model performance?
    g. What is the effect of noise on models performance?

The answer to these questions can further our understanding of the potential of aerial images as a data source for the production of official statistics.

2 Methodology

For the experiments supervised deep learning (section 2.1) was used to automate the detection of solar panels on aerial images (section 2.2). Supervised learning requires labeled data, therefore the dataset was annotated (section 2.3). Finally, the resulting models are validated and tested (section 2.4).

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), as a part of Deep Learning (DL), are the preferred method for image classification. CNNs are unique in that they keep the spatial information of images intact. Other neural network algorithms disregard the position of pixels within an image and do not take advantage of the information provided by nearby pixels. When taking two photos from a slightly different angle of the same object, a dense network will consider them to be completely different where a CNN network is better suited to recognizes the similarities, even though rotations remain difficult. Due to the image processing nature of CNN architectures, it is also more robust when recognizing similar images having different details. When performing image classification, the CNN algorithm 'learns' from an image label by adjusting the weights until the prediction error cannot be reduced. This process, iterating over all the images, is called training and eventually results in a model that can accurately predict the class of an image.

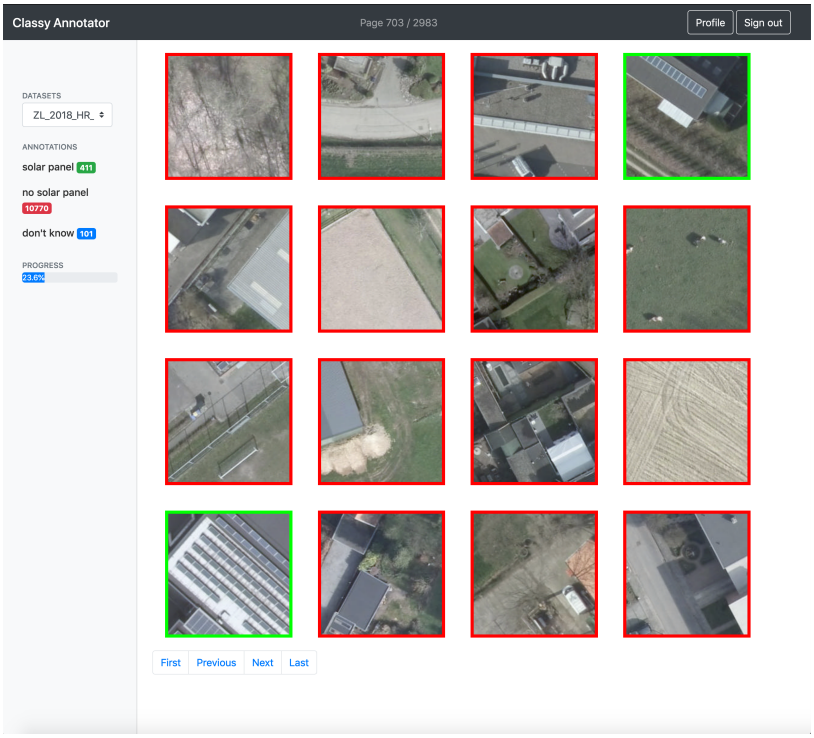
2.2 Images

The image datasets consists of photos taken in autumn 2018 by airplane[1)] , covering the province Limburg of the Netherlands. The photos have a 24-bit RGB color depth and each pixel represents 10x10 cm of detail (DOP10). The infrared (CIR) color spectrum was not available. The lower resolution DOP25 images are freely accessible on the PDOK website[2)] for reference. These photos were then cut into 200x200 pixel images, to make them suitable for deep learning algorithms, which was done during the DeepSolaris project (Jong et al., 2020). One 200x200 pixel image represents 20x20 meter.

2.3 Annotation

Before a network can learn, the dataset has to be labeled. Image labels are made by annotating the images by hand. Each image needs to be viewed and decided upon whether it contains solar panels or not, resulting in a Present or Absent label. Deep networks require quantity, for which perfect quality is not realistic. Due to the labor-intensive nature of this procedure and the limited resources at hand, it is not feasible to double or triple check all the tens of thousands of labels on their accuracy. 23,559 Images from the city of Heerlen and 39,343 from the southern part of Limburg were annotated. These are the same images and annotations used in the DeepSolaris project, complemented with additional images from the same source (Jong et al., 2020). Figure 2.1 shows a screenshot of the web app developed for the image

1) http://www.beeldmateriaal.nl/index.html
2) https://www.pdok.nl/



2.1 Annotation tool as was used in DeepSolaris. Images with a red border are labeled as Absent, Green borders labeled as Present.

classification. By clicking on an image, the border color changes from red (no solar panels) to green (solar panels) to blue (do not know), changing the corresponding label. Images in the 'do not know' class were not used. For faster processing purposes, the images were pre-annotated by a neural network, which was trained on a limited dataset.

2.4 Architecure

Creating a performing architecture is a field of its own. Keras, a high-level neural networks API, comes with pre-built CNN architectures. These architectures were once built by teams to beat other teams in challenges. Their best architectures are accessible within Keras. Using these results on a different image classification is called transfer learning. Transfer learning consists of two parts: firstly the CNN architecture that outlines the type, shape and order of layers, and secondly the weights of the nodes that reside within these layers. These weights are the result of training on 1.2 million images divided in 1000 classes for weeks. The annotated dataset used for training these networks come from ImageNet[3)] , a large visual database designed for use in visual object recognition software research. Hence the weights are referred to as ImageNet weights.

2.4.1 Convolutional Neural Network Architecture

With a pre-built CNN architecture and their respective ImageNet weights loaded, images can be classified based on the 1000 classes the network was trained on. To predict something different, whether solar panels are visible, the architecture needs to be altered. Each layer

3) http://image-net.org/

in a network can be considered as a processing step that prepares a result upon which the final layer will make a verdict: to which class does the image belong? It is for this reason that we must remove this final layer and add our own. Between the processing steps and the final layer, the image gets 'flattened': the spatial information of an image is removed, and what is left are the individual pixels. Flattening the image also concludes the processing of the image, as without spatial information, there is no image left to process. All layers thereafter, including the flatten layer, are considered the 'top' of a network. The top consists of the flatten layer, sometimes some processing layers like dense layers and the final layer that classifies the images. Alternatively a global average/max pooling layer can be used. In that case, all the spatial dimensions are aggregated to 1 value using the average or maximum. This has the benefit that the network also becomes resolution-invariant.

In all experiments where transfer learning was performed, the top was removed, and replaced by a global max pooling layer together with a binary classifier with softmax activation. The softmax function, also known as normalized exponential function, makes the two resulting numbers add up to 1. This way the network outputs a probability for both Present (solar panel detected) and Absent (no solar panel detected).

Each network comes with a different amount of layers. Trainable layers are those layers in a network that contain nodes with weights. Some older networks such as VGG16 and ResNet50 state these layers in their name, to distinguish between networks with a similar layout but a different depth, such as VGG19 or ResNet152. Do note in the graphs that layers not containing node weights, such as flattening or pooling layers, are still considered a layer. This results in higher layer count than when only counting layers with node weights.

2.4.2 ImageNet weights

When training a network, one changes the random noise of the initial node weights into weights that reflect patters in the image data. This is how a network recognizes similarities in images and is able to perform predictions. The reflections of the ImageNet weights have a closer resemblance to other worldly images, such as aerial images with solar panels, than random noise does. This gives ImageNet weights two advantages. First, it speeds up training time as it takes fewer iterations for a model to resemble the new image data than it would when it has to start with noise. In our experiments it was about two times faster. Second, less data is required, resulting in a performance boost when there is insufficient training data. An important drawback would be the reliance on pre-made models and the inability to alter the image processing part of the architectures. Another minor potential drawback is that the transferred weights introduce bias when applied to different images (aerial instead of horizontal) and different classification problems. For example, the dataset used to train ImageNet weights does not include solar panels or aerial images in general.

2.4.3 Training, validating, testing

The models were trained based on image classification. As laid out in section 2.4, transfer learning was used to base the models on pre-built architectures and use their respective ImageNet weights. The models were then trained on 75% of the annotated images of the city of Heerlen and its surrounding area, where the remaining 25% of the data was held apart for model validation during training. Models that performed well were then tested on a different geographical region, the southern part of Limburg with a more dominant rural area to better represent the landscape of Limburg and on North Rhine-Westphalia to test against overfitting.

2.4.4 Class imbalance

With binary classification, ideally the dataset contains 50% Present and 50% Absent training material. The model learns evenly what belongs to each class, be it True or False. However, just 7.4% of the images in the southern part of Limburg dataset contain solar panels and for Heerlen this is 20%. This class imbalance is countered with class weights, meaning a 'Present' labeled image has several times the impact as an 'Absent' labeled image on the learning process. Class weights are applied with the class_weight function of sklearn with the formula:

$$w_j = \frac{n}{Mn_j}, \tag{1}$$

where n is the number of images in the training set (n = 17655), M is the number of classes (M = 2), and n_j is the number of images in the training set labeled j. Based on the Heerlen prevalence, this results in a class weight of 0.625 for Absent labeled images and a weight of 2.5 for Present labeled images. However even with class weights, a preliminary model trained on South Limburg performed poorly. The Heerlen dataset was used for training the models discussed in this paper and the southern Limburg dataset for testing. Using the southern Limburg dataset for testing models, with images containing a more rural setting, has the benefit that it better, albeit not perfectly, represents Limburg.

2.4.5 Training and validation

During training, the model 'learns' by adjusting the weights that link two nodes, and it does so with a cost or a loss function. Initially the cost is high, but as more similar images contribute less new information, the cost lowers. However, a lower cost only indirectly translates into a better performing model, as the training procedure could be stuck in a local minimum or could overfit on training data which will make the model not generalize well. Therefore, a validation dataset is used during training in an attempt to measure the real performance of a model and it does so by using several metrics: accuracy, precision, recall, f1-score.

Accuracy: the percentage of images that were correctly classified, whether they were classified as Present or Absent.

Recall: the percentage of all images **with** solar panels, that were correctly classified.

Precision: of all the images that were classified as 'contains solar panels', that indeed did contain solar panels.

F1 score: the harmonic mean of the precision and recall and is therefore a number that represents both numbers in one. It is more sensitive to a low score on either precision or recall than the arithmetic mean.

Both precision, recall and f1-score can be calculated for True and for False labeled images. When calculating the outcome per class, the best possible scores for the above metrics would be 1 and the worst possible score would be 0.

2.4.6 Accuracy versus f1-score

Since the datasets are suffering from class imbalance, accuracy alone would be a poor indicator of whether a model is performing or not. In the case of the Heerlen dataset with 20% 'Present' labeled images, a model could achieve 80% accuracy by simply classifying all images as 'Absent'. In this case, one would expect the models to perform poorly on precision with images containing solar panels. The f1-score offers solace and is a common metric in the case of class imbalance during development and evaluation, although its lower bound is sensitive to the relative frequency of the class. The f1 is to a lesser extent sensitive to class imbalance. If 20% of the images has solar panels, one gets an accuracy of 80% by always predicting Absent. Similarly, one would get an f1 of 33% by always predicting Absent (2 / (1 + 1 / 0.2) = 0.33). If we were to correct for this, the norm for accuracy would be 0.8 + 0.95 * (1 – 0.8) = 0.99, or 99% and the norm for f1 would be 0.33 + 0.95 * 0.66 = 0.967, or 96.7%.

Heerlen (20% Present)	Baseline	Original norm	Adjusted norm
Accuracy	80%	95%	99%
F1	33%	95%	96.7%

Limburg (7.4% Present)	Baseline	Original norm	Adjusted norm
Accuracy	92.6%	95%	99.6%
F1	13.8%	90%	91.4%

An accuracy of 99.6% or even 99% is extremely high and not likely achievable. Images that contain roof tiles or parallel lined windows that look convincingly like solar panels have a hard Absent label, where images with real solar panels on them, but due to sun glare, odd orientation and other distortions can look nothing like them yet are labeled as Present. At the intersection between the two labels, the model would not be able to distinguish the two as it has no concept of what a solar panel is or what logical applications, locations and orientations are. It bases its predictions on reflections of images that are stored within the model during training that look 'similar enough'. However, with this in mind, we have a better understanding of the true performance of our resulting models.

2.5 Default hyperparameters

As we build upon the DeepSolaris project's results, the hyperparameters used for their best model were taken as our default values. Specifically these data augmentations[4)] were used:

- Height shift range: 0.1
- Horizontal flip: True
- Vertical flip: False
- Rotation range: 30
- Shear range: 0.2
- Shuffle data: True
- Width shift range: 0.1
- Zoom range: 0.2
- Fill mode: reflect
- Seed: 42
- Early Stopping: 2

4) https://towardsdatascience.com/exploring-image-data-augmentation-with-keras-and-tensorflow-a8162d89b844

Furthermore, we made use of Early Stopping, where after two consecutive epochs with worse loss values, the model training procedure stops. When hyperparameters are not named, their default values within Keras v2.3.0 [5] were taken. For example, when the Adam optimizer is used, only the differentiating learning rates are mentioned and the beta values are left by their defaults. There are dozens of hyperparameters to consider, in fact too many, therefore we focus on those that promise the highest impact and will result in the greatest reduction of further hyperparameter tweaks. The network's learning rate, the amount of layers that are trainable and optimizer used are such hyperparameters. Since learning is likely to be influenced the most by tweaking other hyperparameters, the optimal learning rate is recalculated for every experiment. Initially three different networks were used, ResNet50, InceptionResnet and VGG16. To save computational time, ResNet50 was dropped after the first experiment. For the scenarios our only InceptionResnet was used, as it was our best performing architecture.

2.6 Optimizers

During training, when an image is being processed by the network, the model will estimate whether the image does include solar panels or not. The loss function calculates how far off the estimation was, by giving the average 'loss' of all parameters. An optimization algorithm, or optimizer, deals with in which direction and at what pace changes should happen. At a basic level, it will move towards the direction that results in the least amount of loss and it will move with a fixed fraction of the gradient, known as the learning rate[6]. This is what Stochastic Gradient Descent, or SGD, does. SGD can be extended with Momentum. Momentum adapts the descent in a way like a ball gains momentum when it rolls down hill. It causes the ball to go slow in its initial descent, not to overshoot a valley. On the other hand it prevents the ball getting stuck in the first valley it comes across, as the momentum helps the ball in its ascent when it faces a hill afterwards. This helps the ball reach a depth that hopefully is close to the global minimum. Newer extensions of SGD come with per-parameter learning rates, such as Adaptive Gradient Algorithm, or AdaGrad. Or it can adapt the learning rate based on how quickly the gradients change like 'Root Mean Square Propagation', or RMSProp. Adaptive Moment Estimation, or Adam, is an optimizer that incorporates both the benefits of AdaGrad and RMSProp[7] . The Nadam optimizer incorporates Nesterov momentum into the Adam algorithm. Both Adam and Nadam can be considered state of the art optimizers[8] .

[5] https://keras.io/api/
[6] https://dominikschmidt.xyz/nesterov-momentum/
[7] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
[8] https://github.com/yanni-georghiades/Nadam-v.-Adamhttps://github.com/yanni-georghiades/Nadam-v.-Adam

2.7 Scenarios

During the training process a lot can be controlled, though there is always noise coming from random occurrences that cannot be controlled. Whenever a model's performance approaches 100% accuracy, any improvement or decline can be hard to distinguish from this noise. Alternatively, hyperparameters can influence each other in unexpected ways that may go unnoticed. By taking the hyperparameters used to create the model that performed the best, we can start tweaking the hyperparameters back and forth, one at a time, to get a better understanding of their influence on the final result.

2.7.1 Data Augmentation

Data augmentation in Keras comes with various possibilities. Here we look into some of the ones used by the DeepSolaris project, and as such also used in the experiments of DeepSoLim. For a full list of data augmentation and their settings used, see section 2.6. For the data augmentations experiments, all augmentations were disabled, except for the augmentation in question. Without doing this, we would be unable to track down augmentations that have a negative impact on the result. First a model is made without any augmentations for comparison. Then each augmentation is turned on individually. One important limit is that the data was validated on the same validation set as all other experiments were and not on the South Limburg dataset. Section 5.5 goes into greater detail about this. Combinations and resulting interactions between data augmentation techniques are beyond the scope of this study.

3 Results Classification

The following experiments aim to develop a high performing model to automatically detect solar panels on aerial images in Heerlen, South Limburg and NRW. The output of the model validations were used to describe the results.

3.1 Trainable layers

With the top removed, VGG16 has 13 convolutional layers, that can be marked as either trainable or not trainable. The layer index indicates the point at which the layers were made trainable. Layer index 0 means that all layers of the network were made trainable. Layer index 7 means that the network had layer 0 to 6 set as not trainable and layer 7 to 13 set as trainable. With just a few layers trained, figure 3.1 shows VGG16 performs reasonably well. This is unsurprising as VGG16 has a much shallower depth than the other architectures, containing less detail of the training data, causing it to generalize better. The first 6 layers perform similarly, and although layer 6 offers the best result, the differences are so small that they cannot be distinguished from noise. It's therefore important to look at the trend line, not just the individual results.

Resnet50 and InceptionResnet are networks made with repeating groups of layers, or 'blocks', causing them to be extremely deep. As it would be nonsensical to measure the performance of trainable layers at each individual layer, the performance was measured after each such block. This results in 36 models for the ResNet50 architecture and 44 models for the InceptionResnet architecture. Resnet50 (figure 3.2) starts to perform well when more than half the network layers are made trainable with a sharp decline after layer 141. It seems that the more layers that are made trainable, the better the result. With 776 layers, InceptionResnet (figure 3.3) is the deepest network used in the experiments, and therefore contains more details about the data it was trained on. The ImageNet dataset does not contain solar panels, resulting in a poor initial performance. Independent of the architecture, we can conclude that a model performs better

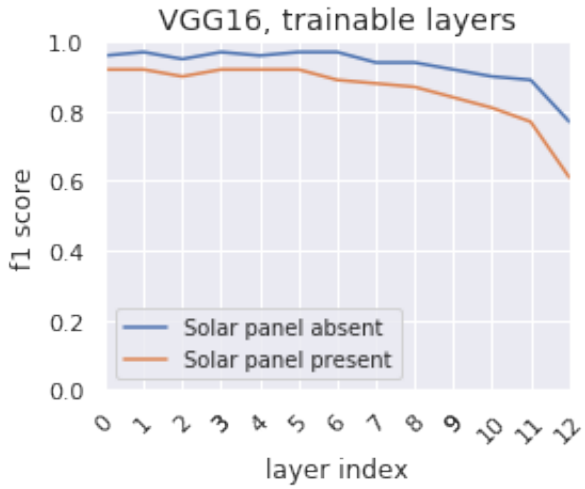
Architecture	F1 'Absent'	F1 'Present'	Validation Accuracy	At layer	Max depth
VGG16	98%	94%	97.7%	6	13
ResNet50	97%	93%	97.8%	27	255
InceptionResnet	97%	90%	96.4%	0	776

3.1 Trainable layers validation results

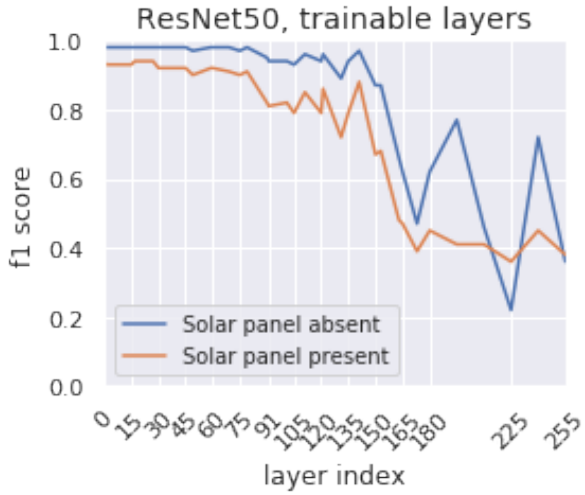
when it is trained with all layers set as trainable, reducing the amount of hyperparameters we have to test for.

3.2 Optimizers

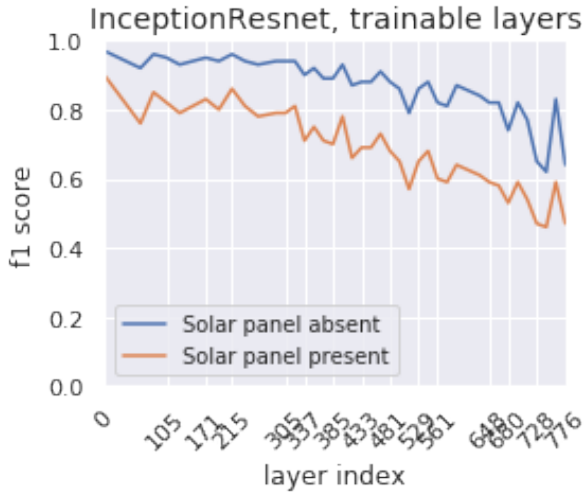
VGG16 with SGD+Momentum (figure 3.4) shows a clear cutoff at 5e-4 learning rate, which is also its best performing model. SGD+Momentum offers the worst performance, which is not entirely surprising given that the other optimizers are improvements upon SGD. VGG16 with the Nadam optimizer performs much better (figure 3.5). Learning rates above 1e-4 show erratic behavior as can be seen in the graph and offer poor results. Here Nadam goes over its intended mark, which is a logical consequence for a too high learning rate. At and below 1e-4 Nadam



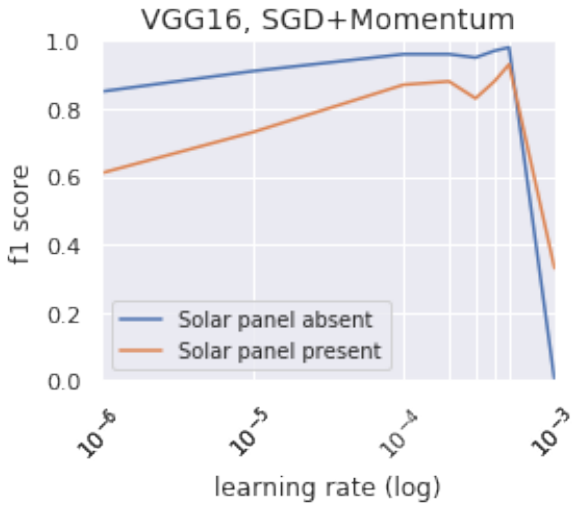
3.1 VGG16 performance with layers trained from the layer index onward. A lower layer index means more layers were trained



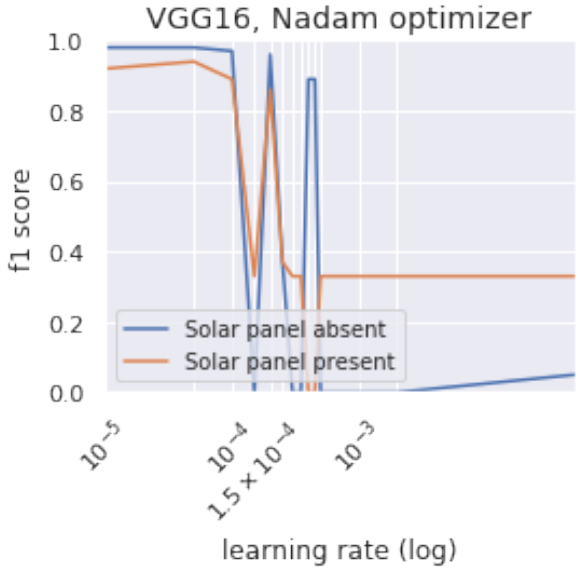
3.2 ResNet50 performance with layers trained from the layer index onward



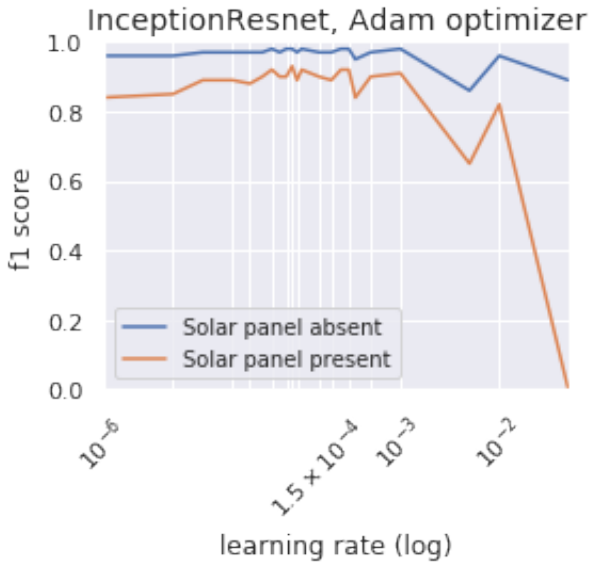
3.3 InceptionResnet performance with layers trained from the layer index onward



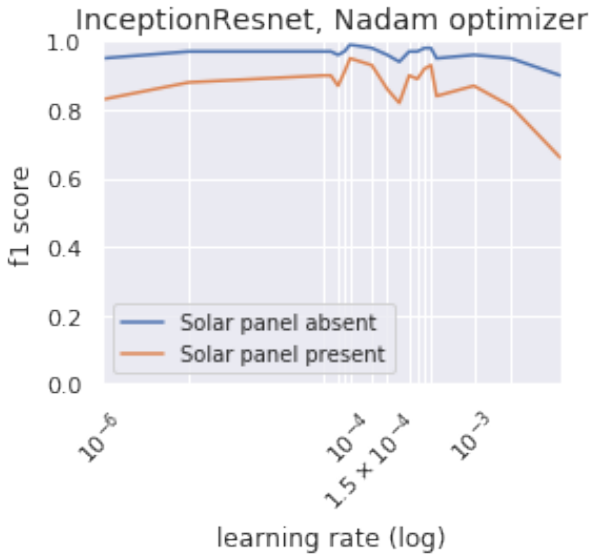
3.4 VGG16, SGD with Momentum at different learning rates



3.5 VGG16, Nadam at different learning rates



3.6 InceptionResnet, Adam at different learning rates



3.7 InceptionResnet, Nadam at different learning rates

gives consistent results. InceptionResnet performs well with both optimizers, where Nadam (figure 3.7) outperforms Adam (figure 3.6), especially in the 'present' f1-score. As such we can conclude that Nadam is the preferred optimizer which provides the best results with very low learning rates. A minor observation is that SGD+Momentum comes with a narrow bandwidth

Architecture	Optimizer	F1 'Absent'	F1 'Present'	Validation Acc.	Learning Rate
VGG16	SGD+Momentum	97%	93%	97.5%	5e-4
VGG16	Nadam	98%	94%	98.1%	5e-5
InceptionResnet	Adam	98%	92%	98.0%	3e-5
InceptionResnet	Nadam	99%	95%	98.1%	1e-4

3.2 The effectiveness of different optimizers

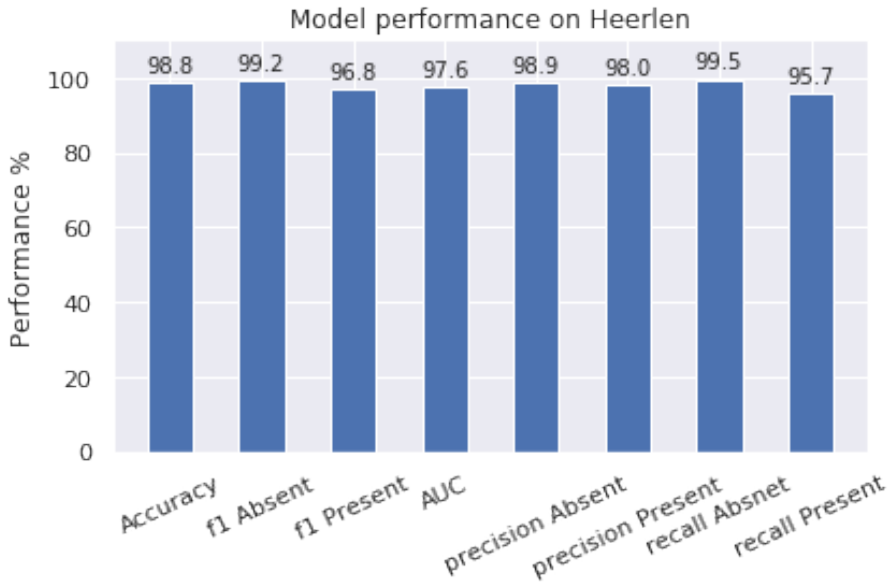
of learning rates at which it operates well. This makes sense as Adam and Nadam have more control over their learning rate. InceptionResnet architecture is capable of producing models with an accuracy above 90% with a learning rate set as low as 0.01 and as high as 1e-7, making it more robust than VGG16.

Architecture	Optimizer	Minimum learning rate	Maximum learning rate
VGG16	SGD+Momentum	0.0001	0.0005 (1e-4)
VGG16	Nadam	0.0004	0.00001 (1e-6)
InceptionResnet	Adam	0.01	0.000001 (1e-7)
InceptionResnet	Nadam	0.002	0.000001 (1e-7)

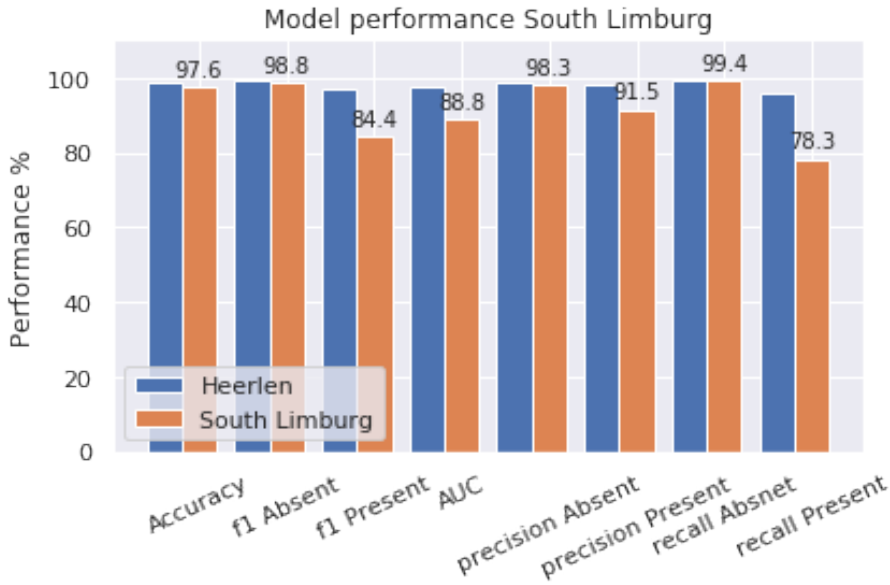
3.3 The range of learning rates in which all models score above 90% validation accuracy

4 Model performance testing

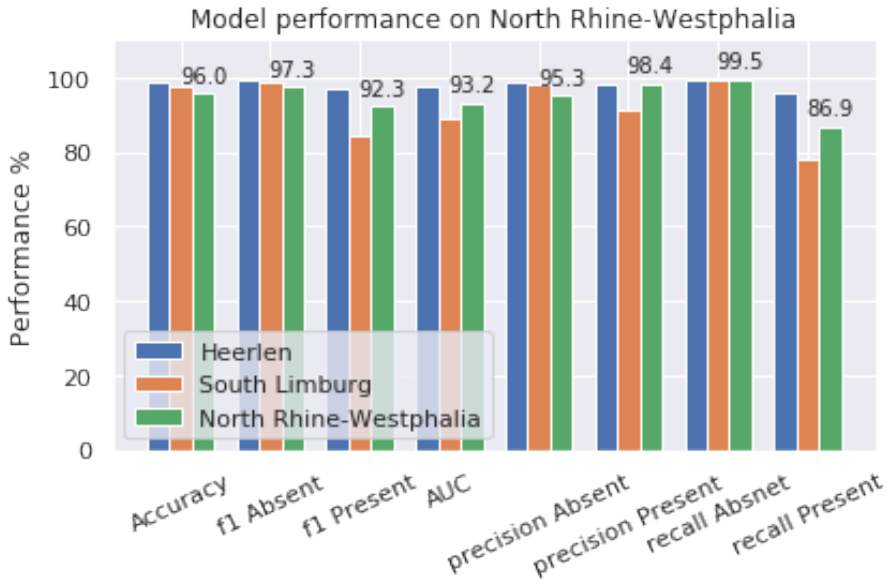
We have found our best model by using the InceptionResnet architecture with the Nadam optimizer and a learning rate of 1e-4. When we test the model on the Heerlen dataset, the one which was partially used to train with, we get the following results (figure 4.1). 95.7% of images with solar panels were recognized (recall), 2% of the images classified as having solar panels did not contain them (1 - precision). When testing on the South Limburg dataset (figure 4.2), 78.3% of images with solar panels were recognized, and 8.5% of the images classified as having solar panels did not contain them. When we compare the model with the dataset from North Rhine-Westphalia (NRW), Germany (the same used in DeepSolaris) we notice that the model performs much better than on South Limburg (figure 4.3), and that is odd. 86.9% of the solar panels were recognized, where just 1.6% of the images that were classified as having solar panels, did not contain them. 98.4% of the images that were classified as having solar panels, did indeed contain solar panels. It is plausible that the South Limburg dataset contains certain data with unique characteristics not found in the Heerlen nor the North Rhine-Westphalia dataset, explaining the drop in recall. What the model doesn't know, it will not recognize. Further analysis of these images could provide answers.



4.1 Best model performance tested on Heerlen



4.2 Best model performance tested on South Limburg



4.3 Best model performance tested on North Rhine-Westphalia

5 Scenarios

The following scenarios aim to answer some of the questions revolving the following topics and their impact on model performance:

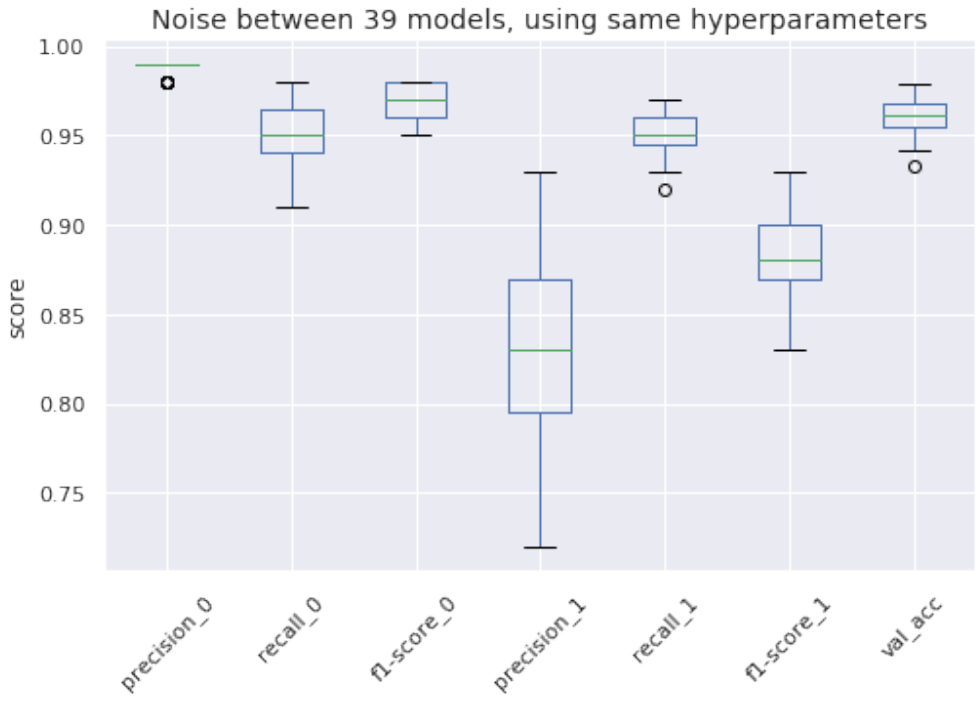
- noise
- sample size
- class imbalance
- class weights
- validation dataset
- data augmentation
- ImageNet weights

As before, the numbers reflect the validation output.

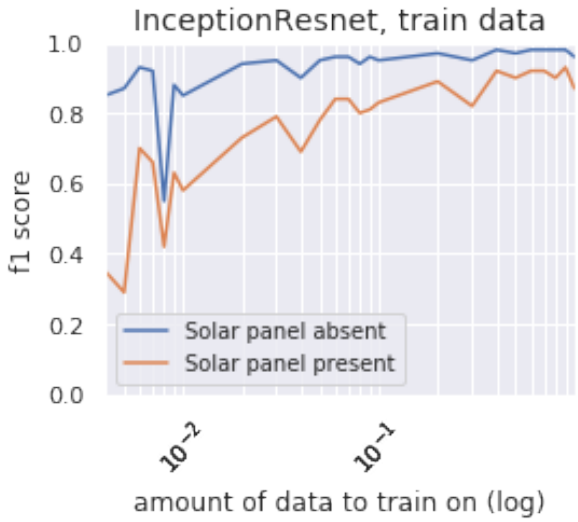
5.1 What is the impact of noise on model performance?

Noise can have a profound influence on the performance of a model. To deal with noise, we have looked for trends within graphs to find an approximation of the optimal hyperparameter we are after. When looking at figure 5.1, a dashboard representing 39 models in each graph which were trained with the exact same hyperparameters, we see the extent to which noise influences a model's performance. Ideally we should see horizontal lines, indicating no difference in performance between models. Understanding per metric how much noise is involved, gives us an idea of how accurate our results are and how much further improvement is possible when retraining a model with the same hyperparameters. Quite a bit of noise is generated by using shuffled data, which can be considered a double edged sword. Shuffled data leaves room for trying again, hoping to get an improved model where with no shuffling there are no second chances. When aiming to perfect a model, the higher the performance of a model, the harder

5.1 Graphs showing performance(y) versus noise levels(x)



5.2 Bandwidth of noise per metric for labels Absent (_0) and Present (_1)



5.3 Performance versus relative sample size

it gets to distinguish improvements over noise. With better hardware, cross validation could offer a solution.

5.2 What is the effect of sample size on model performance?

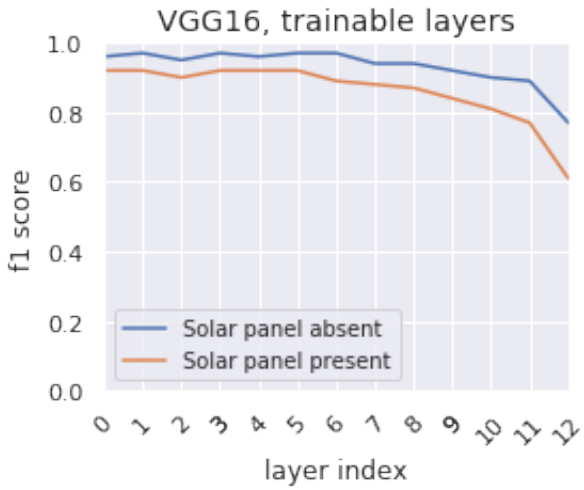
The best model performed well, but could it be better with more data? What is the performance impact of the amount of data on a model? When looking at figure 5.3, we can safely conclude that enough data was used to make our model. More data could improve the model further, though it would require exponentially more annotated data from the region of Heerlen. The graph also shows that with just a little bit of data, a reasonably well model can be build. With as little as 2% of the training data (a stratified sample), a model with 90% accuracy can be made. That's just 70 images with solar panels used during training. Likewise with 8% of the data, consisting 280 Present labeled images, an accuracy of 95.3% was achieved. Such improvement may seem small, however when considering the lower bound of 80% (achievable by always predicting the 'not present' category), the effective increase is 25%.

Images	Sample %	F1 Absent	F1 Present	Accuracy
353	2	94	73	90.5%
1414	8	94	80	95.4%
7067	40	98	92	97.0%
15902	90	98	93	97.8%
17669	100	99	95	98.1%

5.3 What is the effect of class imbalance on model performance?

From the experiments discussed in section 5.1 we know that sample size has a profound influence on the performance of models. This should have a similar effect when dealing with class imbalance as it is a lack of data within a certain class. However class weights make this a more complex matter, as they cause the network to learn more aggressively when it concerns images with solar panels on them. In the section 2.5.1 we chose the Heerlen dataset for training

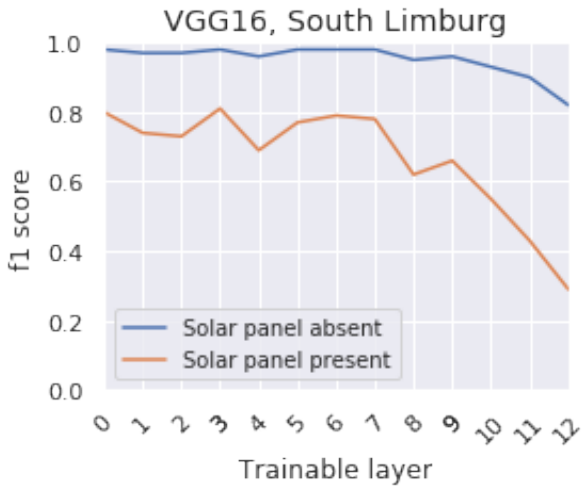
Class imbalance	Present label
Heerlen dataset	20%
South Limburg dataset	7.4%



5.4 InceptionResnet using ImageNet weights

and evaluation purposes, as it has a higher address density resulting in a smaller class imbalance. What would happen to the performance of the models if we had used the South Limburg dataset? As before, the numbers reflect the validation output.

The original graph of the trainable layers experiment (figure 5.4) shows that the network has a harder time recalling images that contain solar panels. This could have several causes, such as the class imbalance within the Heerlen dataset, or the fact that the area that solar panels cover within an image is rather small. It is this small area that the network has to learn to recognize solar panels from. When we repeat the same experiment while training on the South Limburg dataset, we see a dramatic decline in performance (figure 5.5). Although the South Limburg dataset is larger, it contains three times less Present labeled images, and interestingly the the distance between the f1 Absent curve and the f1 Present curve has become about three times as large as well. Would the opposite be true, if the Heerlen dataset would have no



5.5 InceptionResnet using randomly initialized weights

class imbalance? Important to note is that class imbalance is confounded with area and weight initialization method.

5.4 What are the effects of class weights on model performance?

Ratio class weight	Weight Absent	Weight Present	F1 Absent	F1 Present	Accuracy
70/30	0.7143	1.6667	95%	82%	95.7%
80/20	0.625	2.5	98%	95%	98.1%
90/10	0.5556	5	96%	87%	94.6%
95/5	0.5263	10	95%	84%	94.7%

To counteract class imbalance, class weights are used. These weights make the network learn more aggressively from images from the smaller class than from the bigger class, giving each class an equal weight during training. While the density of pixels that describe solar panels is tiny compared to the pixels that do not, further tweaking the class weights unfortunately does not result in a better performance. The current class imbalance of 80/20 causes the Present labeled images to count 5 times as much during training already. Make them count 10 times does not make up for the lack of data.

5.5 What is the effect on model performance when using the South Limburg dataset to evaluate performance during training?

Evaluation results	F1 'Absent'	F1 'Present'	Evaluation accuracy
Original model	99%	95	98.1%
New model	97%	88%	97.5%

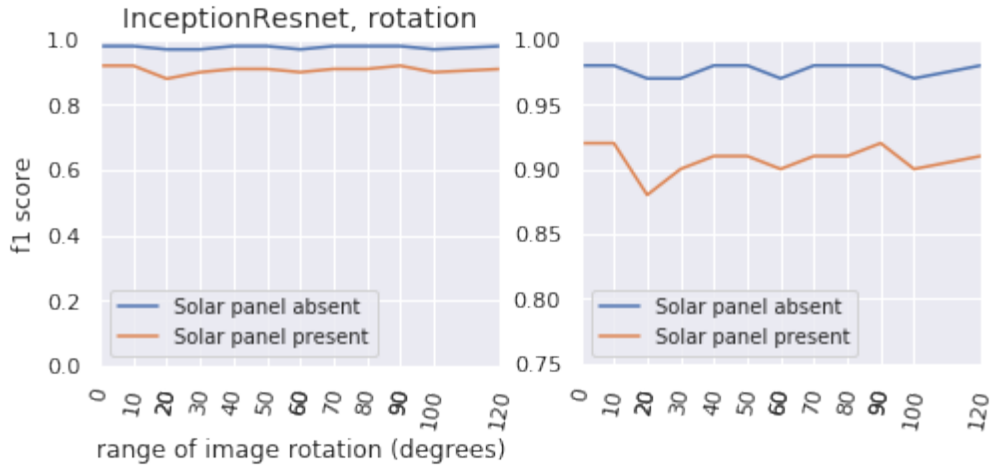
Test results	F1 'Absent'	F1 'Present'	Test accuracy
Original model	99.2%	96.8%	98.8%
New model	97.0%	56.4%	94.4%

Evaluating the model on South Limburg worsened the performance. This is probably due to the extreme class imbalance of the South Limburg dataset, where only 7.4% of the data contain images with solar panels.

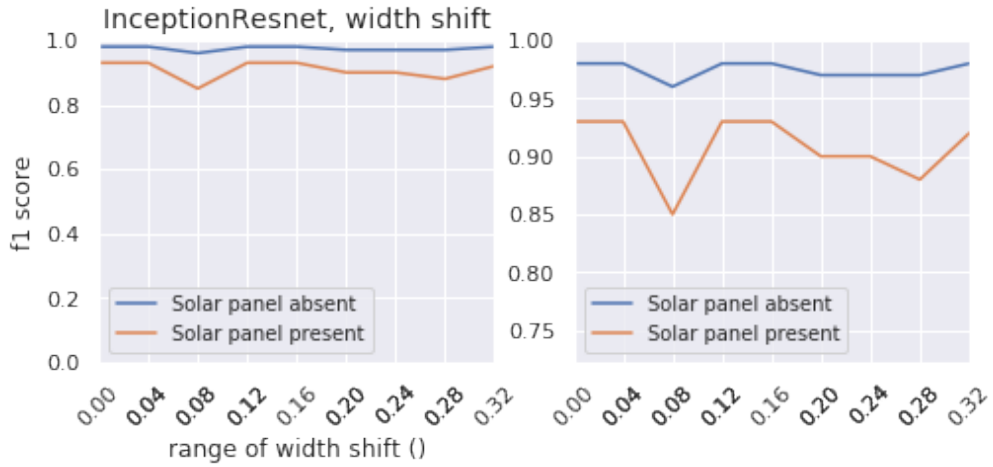
5.6 What are the effects of data augmentation?

Images	F1 Absent	F1 Present	Accuracy
All augmentations	99	95	98.1%
No augmentations	98	92	96.7%
With horizontal flip	98	93	97.7%
With vertical flip	98	91	93.7%

Data augmentations are not all harmless as vertical flip points out. Solar panels are almost all faced south with a certain angle facing the sky. Flipping them to face the north would not correspond with real data, and the poor results seem to support this notion. Horizontal flipping, although similar in nature, is rewarded with a substantial positive influence, which does makes sense as the solar panels keep facing south this way. Image rotation (figure 5.6) shows

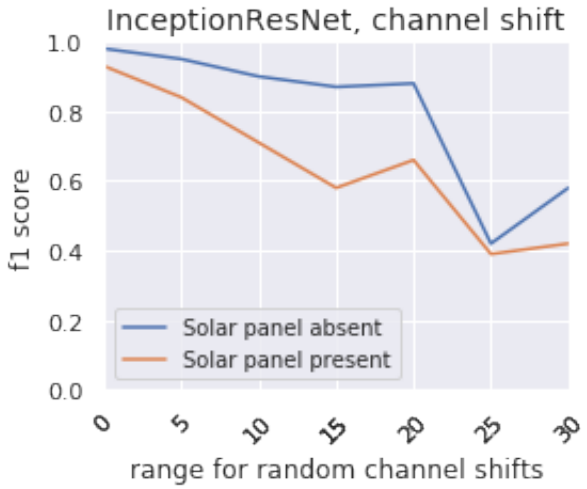


5.6 Model performance with image rotation augmentation (left: absolute, right: scaled)

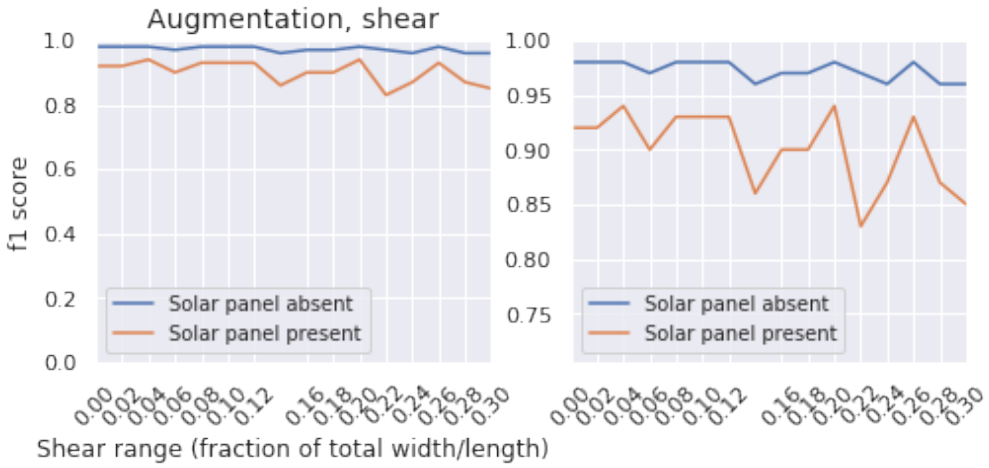


5.7 Model performance with image width shift augmentation

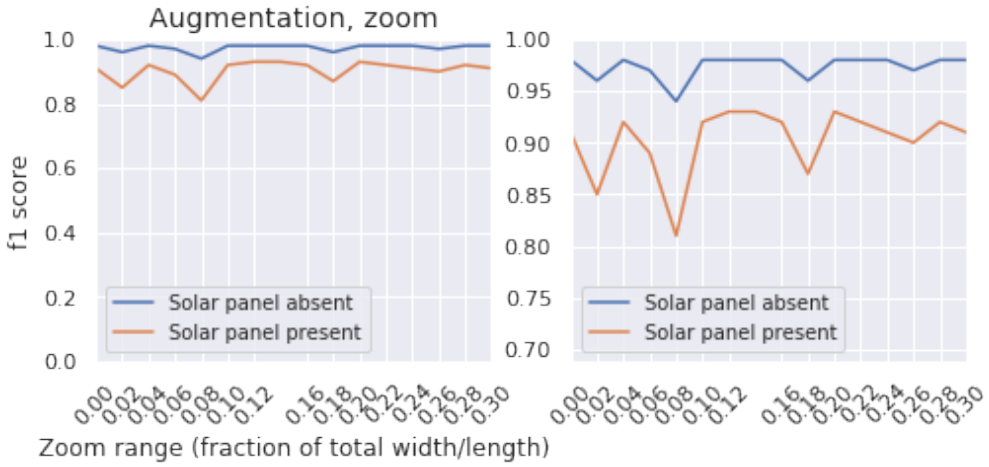
no improvement at best and a negative impact on the end result at worst. At 0 rotation range, in essence the same as having no data augmentation performed, we get our best result. In our earlier experiments we used a 30 degree image rotation, which could have had a negative impact on the end result. The same is true for width shift (figure 5.7). It should be used with caution, if at all. Channel shift is shifting the colors on each channel randomly within the specified range and has a dramatic negative impact in conjunction with aerial images (figure 5.8). Shear range (figure 5.9) is similar in nature. Here the image is fixed on one axis and then shifted or stretched on the other axis. The graph shows a downward trend, however with possible improvements at lower ranges. Zoom range (figure 5.10) is not exactly what one would expect it is. A better name would be 'stretch range', as the image is independently stretched on each axis. Apart from a few bad performing models, the range of the zoom does not seem to impact the models much.



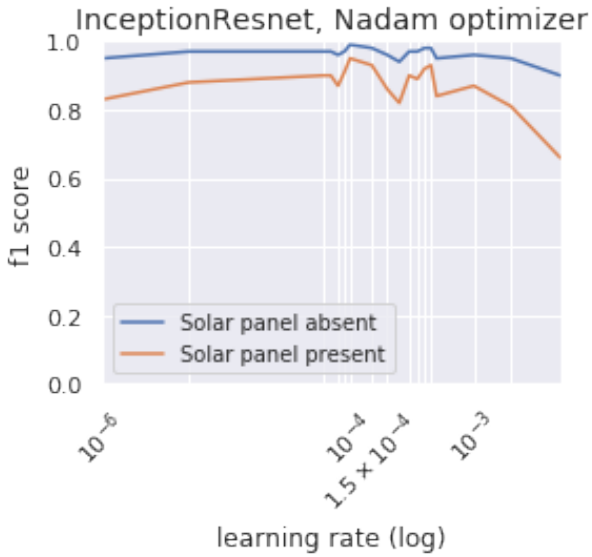
5.8 Model performance with image channel shift augmentation



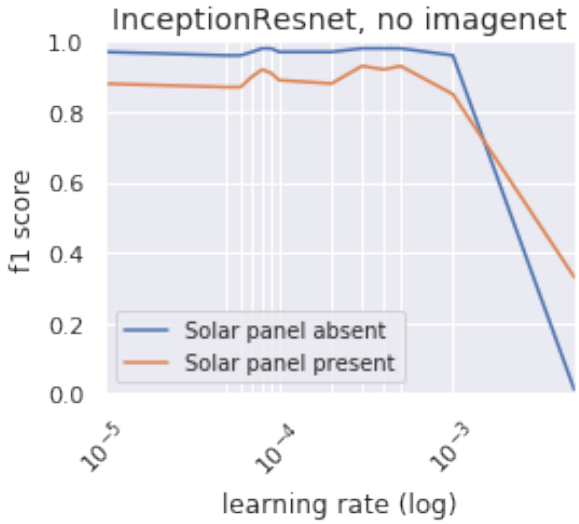
5.9 Model performance with image shear augmentation



5.10 Model performance with image zoom augmentation



5.11 The original graph using ImageNet weights



5.12 Using no ImageNet weights

5.7 What are the effects of ImageNet weights?

We have already seen some of the effects of ImageNet weights during the experiments about trainable layers in section 4.1. We concluded that the best performance was gained when all layers were set as trainable. When we set all layers as trainable, it means all of the ImageNet weights get overwritten. That raises the question of why use them at all? They come with the hindrance that one is forced to make use of pre-built architectures that contain these weights, and that these architectures cannot be adjusted or extended to one's needs. One expectation was that it would take longer to train from randomly initialized weights. This turns out not to be the case, if anything it is slightly faster. However, the resulting models perform worse (figure 5.12). Although an argument could be made that the model with ImageNet weights had twice the amount of training (6 epochs versus 3 epochs), resulting in a better performing model, it is not an uncommon amount of epochs for the Nadam optimizer to find its optimal result. A more obvious explanation would be that more data, even data not specifically containing solar

Best Models	F1 'Absent'	F1 'Present'	Accuracy	Epochs
With ImageNet	99%	95	98.1%	6
Without ImageNet	98%	92%	97.5%	3

panels objects, makes the model more robust. Any group of pixels that better help to describe solar panels or make better features to identify them during model evaluation can contribute to the result.

6 Conclusion

The best DeepSoLim model was able to classify with 97.6% accuracy the South Limburg dataset. Among the Present labeled images, it was able to recall 78.6% of them. For the time being, the following best practices were discovered: When making use of transfer learning, setting all layers as trainable in the network produced the best performing models. This was true for all the architectures tested. The InceptionResnet architecture performed the best in our experiments, in conjunction with the Nadam optimizer, using a learning rate of 1e-4. Nadam produced not only the best results on VGG16 and InceptionResnet, it is also the fastest optimizer by far. With as little as two or three epochs it is able to reach a better result than Adam or SGD do in two or three times as many epochs.

Noise can have a severe impact on model performance. The closer a model gets to perfectly classify images, the harder it becomes to distinguish improvements from noise. That limits our ability to mainly look for trends between models in graphs. To gain better insights, more powerful hardware would allow 10-fold cross validation for more exact results.

With over 23k annotated images, the project used sufficient training data to build its models. More data from Heerlen would have provided only marginal improvements, where exponentially more data would be required to see further improvements. Data from other sources would have a different, presumably larger impact. Aerial photos produce a huge amount of data, however annotating these images limits the amount of data that can be used. It is therefore reassuring that with as little as 350 images (with just 70 containing solar panels), a model was able to achieve an accuracy above 90%. With 1400 images (280 containing solar panels) it improved to above 95% accuracy. An improvement from 90% to 95% does not seem a lot, but it is considerable if you take into account the lower bound of 80% (below 80%, always guessing 'Absent' would result in a higher accuracy). It took 16k extra images to get to an accuracy of 98%.

One way to deal with class imbalance is adjusting class weights. However, tweaking these weights beyond or below their standard values, did not improve the results. This came somewhat as a surprise, for there is not just a class imbalance, but also an imbalance within the images themselves. A pixel imbalance. When we would break down an image containing solar panels into four smaller images, three of them would not contain any solar panels. Would we split the complete dataset like this, the class imbalance would shift from 80/20 to 95/5, while the data remained the same. Class weights offer no solution.

Throughout the study, models were trained and validated on data from Heerlen. It is more logical to validate the models on the area they are built for. However, when validating on the South Limburg dataset, both the validation and test results worsened. The extreme class imbalance combined with the rural setting of the South Limburg region are the likely culprits for the disappointing result.

Data augmentation can have a profound effect on the model's performance. However, their impact on the Heerlen validation set showed little promise. Vertical flip and channel shift have an outright bad influence. Image rotation and width shift seem to do little, though at higher settings have a clear negative impact. Where zoom and shear range seem harmless, horizontal flip is the one augmentation that can safely be applied, offering a clear performance boost. It is possible, however unlikely, that a combination of these augmentations have a positive

effect, where they do not in isolation. Testing these combinations was beyond the scope of this study. To offer more conclusive answers, these models would preferably be tested on a different dataset than Heerlen.

As training your own network using the ImageNet dataset has its obvious limitations, using ImageNet weights limits your choice in architecture. And although these weights are the result of training on a dataset containing no aerial images with solar panels, they did improve model performance. The increase in accuracy would indicate that even unrelated data contributes to the robustness of a model to help differentiate between what solar panels are and what not.

7 Discussion

DeepSolaris, upon which DeepSoLim was built, involved nine researchers from three different institutions trying to find out whether Deep Learning is a feasible method to detect solar panels on aerial images. They laid the groundwork for what made DeepSoLim possible and as such there is some overlap. Noticeably, the same machine, software, scripts and data was used and further built upon. The scripts got extended, more automated and the data got an upgrade as more annotated data was available. Despite these similarities, a comparison between the models of DeepSoLim and DeepSolaris would be somewhat comparing apples to oranges due to the change in data. DeepSolaris also looked into object detection, a step beyond the more basic image classification.

This overlap caused some issues to be discussed. In both studies, each time a new model was made, the script made a new training and validation set from randomly selected images from the same dataset. This meant that on occasion, the more unique and therefore harder-to-classify images could end up in the validation set. This would make the model seem perform worse than it actually did and vice versa. This additional noise makes it hard to compare one standalone model to the next and therefore comparing one hyperparameter to the other. This noise, as is also discussed in paragraph 5.1, was countered by not relying on single standalone results but by forming conclusions on trend lines in graphs based on multiple models. More importantly in DeepSoLim the most important figures are based on results from different test datasets, which were never altered and always the same, validating the results. Although this issue was found in time and hundreds of new models were made with fixed validation sets in a 4-fold cross validation setup, this study had to rely on earlier work due to the impact of Covid19. However a 4-fold cross validation proved to be problematic as well. Next to being four times slower, how to deal with results where two out of four models perform really well and two do really bad. Is it a bad hyperparameter or just an unlucky result? With new hardware, 10-fold cross-validation in combination with Deep Learning would become feasible, making it an issue of the past.

Methodologically, there are several other things that should be done different in future projects. When the goal is to test how well a model generalizes in different geographic areas, training on one location, such as Heerlen and testing it on another, such as South Limburg, is a good approach. When the goal is to make the best possible model that will function effectively in a certain area and beyond, this approach is inadequate.

First, we need to be smarter about the data, where more isn't necessarily better. The training data should be less clinical and preferably as varried as possible. It requires those images that add the most new information to the training dataset. We don't need more data, but a higher variety and that means taking bits and pieces from everywhere, from both inside and outside the target area (in our case Limburg). The ImageNet dataset contains no solar panels, yet it provided a noticeable boost to model performance. This shows that a higher variety of data will make a more robust model. Data from outside Limburg could add to this. We also saw that with as little as 280 Present labeled images, the classifier could achieve 95% accuracy. It may be all the data we need from a single region such as Heerlen. Combining many such regions together offers a more diverse training set, resulting in better performing and more robust models. Since the goal would be quality over quantity, the resulting dataset could become much smaller, reducing class imbalance, reducing the annotation burden, and speeding up model training.

Second, there should be a clear distinction between the training set on the one hand and the validation and test set on the other hand. The quickest way to improve the current model for Limburg, is to include the dataset from North Rhine-Westphalia in the training dataset. It is important to note that any data can be included during training from any area, as long as it is not included in the validation or test set. Although the network learns from images and labels offered by the training set, at each epoch it is the validation loss that shows whether a model is improving or not. This is what Keras looks at when stopping the training process with Early Stopping. As we are building a model for Limburg, the validation loss should reflect how the model performs on Limburg and nowhere else.

Third, when considering which data to include in a new and improved dataset, register data has offered help in the past. However, register data might be incomplete or not available in every region. Another possibility is to use machine learning to select areas of interest, e.g. with mobile phone data. Phone usage could indicate where people remain at night, in their homes, some of which contain solar panels. This method could prevent overlooking smaller towns and villages with unique building architecture. It is these odd structures with solar panels we need more data of.

Fourth, we should regard the impact of pixel imbalance. We concluded that the pixel imbalance is intertwined with class imbalance, therefore even in a balanced dataset there would be relatively little information describing solar panels. This imbalance will become even more apparent when future architectures expect even larger images as input, as was the case with VGG16 (224x224 pixel input) and InceptionResnet (299x299 pixel input). With this in mind it would make more sense to not seek balanced classes, but class imbalance in favor of images with solar panels on them. This would increase the information for a network to learn how to distinguish solar panels on images. An alternative would be using Multi-Task learning where you also classify things beside solar panels, although this would require more annotation time.

Although data augmentation could be further analysed, specifically regarding interactions between the different augmentations, no huge differences would be expected from changing the augmentation parameters. Provisional best practices were given in the conclusion, and the study also showed the bandwidth in which hyperparameters would give sensible results. Together with the data augmentation results, these best practices have the potential to greatly speed up upcoming research asking different questions using the same or similar data.

8 Acknowledgements

This paper would not have been possible without the exceptional support of my supervisor, Joep Burger. His insights, knowledge and attention to detail have been an inspiration and kept my work on track. I appreciate the patience he showed when I challenged his perspective on methodology. I'm also grateful for the insightful comments offered by Marco Puts, Piet Daas and Tim de Jong and especially Herbert Kruitbosch. Their generosity and expertise have saved me from many errors. My special thanks go to Anke Consten, Yvonne Gootzen, Lyana Curier and Nick de Wolf, who patiently helped me with my struggles.

References

Jong, Tim De et al. (2020). *Monitoring Spatial Sustainable Development: semi-automated analysis of Satellite and Aerial Images for Energy Transition and Sustainability Indicators*. arXiv: 2009.05738 [cs.CV].

Yosinski, Jason et al. (2014). "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems 27*. Edited by Z. Ghahramani et al. Curran Associates, Inc., pages 3320–3328. URL: http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf.

Zhang, Chaoyun et al. (2015). "A Convolutional Neural Network for Leaves Recognition Using Data Augmentation". In: *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 2143–2150.

Colophon

Publisher
Statistics Netherlands
Henri Faasdreef 312, 2492 JP The Hague
www.cbs.nl

Prepress
Statistics Netherlands

Design
Edenspiekermann

Enquiries
Telephone: +31 88 570 70 70
Via contact form: www.cbs.nl/infoservice