university of groningen

faculty of science and engineering

Exploring Hybrid Morphological Neural Networks: A Comparative Study on CNNs

Bachelor's Project Computing Science

June 2025

Author: Reinier Huese
Student Number: S3834352
First supervisor: dr. Michael H.F. Wilkinson
Second supervisor: prof. dr. Kerstin Bunte

Abstract

Neural networks (NNs) were first introduced in 1943 by Warren McCulloch and Walter Pitts, in an attempt to emulate the biological neurons in the visual cortex of a cat. Neural networks have since been extended with convolutional layers, forming convolutional neural networks (CNNs), with applications in image classification.
A more recent variety of NN is the Morphological Neural Network (MNN) where the convolutional layers are replaced by morphological layers. Several studies have shown an increased performance when forming a combination of CNNs and MNNs into a Morphological Convolutional Neural Network (MCNN).
We theorize that increased performance with the same training data volume can be translated to the same performance with less training data. This would lower the requirements of getting a custom neural network, as less data is required for the purpose of training.

In our experiments, the MCNN has shown a significantly decreased accuracy in most cases with Gaussian noise and spectrally uncorrelated salt and pepper noise. For spectrally correlated salt and pepper noise, the MCNN achieves significantly higher performance compared to the CNN in all cases.
Both models show slight improvement when training on the Indian Pines (IP) dataset, prior to training and testing on the University of Pavia (UP) set, however the effect is insignificant in most cases. The retraining does result in the MCNN achieving significantly higher accuracies when compared to the CNN for certain low training percentages, where there was no significant difference before.
Additionally, we have found that the MCNN is able to achieve significantly higher accuracies compared to the CNN when training on 1% or more of the IP dataset, and 2% or more when trained on the UP dataset.
Another observation is that to achieve higher performance than the MCNN that has been trained with 5% of either dataset, the CNN needs to be trained on 15% of that dataset.

Contents

List of Figures

List of Tables

1 Introduction

Neural networks are an essential part of the area Intelligent Systems, making it possible for an algorithm to be developed based on very specific use cases. NNs make use of annotated data to learn how to produce the desired outcomes [14]. This learning happens during the training phase, where the NN is given a large amount of data, it will produce an output, and this output will be judged. Based on the judgement, the NN will make adjustments, aiming to improve the result. After a certain amount of data has been used to teach the NN, and it is producing the output at a sufficient accuracy, it will be put to use.

Convolutional neural networks make use of convolution layers, to pre-process images before feeding them into the neural networks [1]. Convolution enables detection of edges in images, making it easier and faster for the NN to classify the image. Another significant benefit is the CNN's ability to reduce the input image to smaller patches, lowering the number of NN nodes needed, and therefore the number of nodes that need to be trained. A CNN performs as an approximation machine, attempting to approximate a function that produces the correct output, based on the input [8]. With this method a CNN can achieve high performances, provided that it has enough trainable parameters, and enough training data for these parameters to gain the correct weights.

However, convolution has its limitations. Convolution is a linear operation, missing lines in the input image as a result of noise, artifacts or missing pixels, will not be detected. This algorithm cannot go beyond what is on the image, therefore setting high requirements on the input data. Alternatively, if the high requirements are not fulfilled, the NN behind it may need to compensate for less than optimal image detection, which could result in lower accuracy or more training time and data needed.
Furthermore, a CNN attempts to approximate the function of the input image using the linear convolution operation, whereas patterns and internal relationships within an image are better characterized by nonlinear operations [17].

An alternative to convolution is morphology, first explored in 1996 [16], using erosion and dilation to detect edges. Unlike convolution, morphology is able to fill in missing data points, or overwrite noise and artifacts. If a line has a gap in it, dilation and erosion could fill in this gap, and detect the full edge. This results in lower requirements of the input data. Another potential benefit could be that the NN behind it does not have to be as robust to make up for suboptimal edge detection. This could reduce training time and data volume, or the number of nodes needed in the NN. Similar to the CNN, an MNN also produces patches to feed into the NN, reducing the number of nodes required. Additionally, morphology consisting of nonlinear operations will also be beneficial when attempting to find the patterns and interactions within the image as opposed to using linear operations.

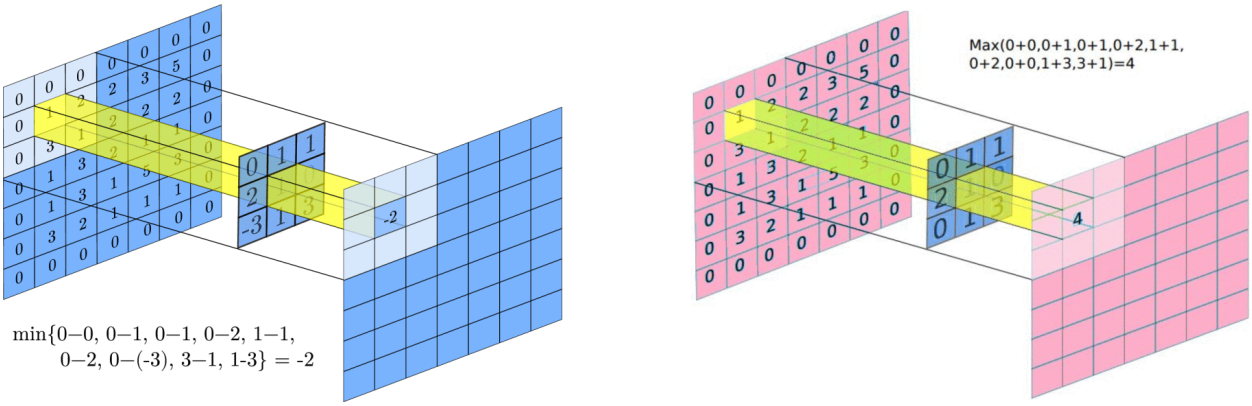
A concrete benefit of reduced data volume would be the lowered requirements on someone wanting to get an NN trained for a specific purpose. Training an NN requires manually annotated images, making judgement possible of whether the NN's output is accurate. This reduces the time spent annotating images by the person themselves, or by a hired expert.

Several studies that have combined some morphological layers with convolutional layers, into an MCNN, have shown increased performance, with the same amount of input data, compared to a pure CNN [10, 11, 17]. However we propose to look at the other implications of this finding; **"Does an MCNN need less input data to achieve equal results to the CNN?"** and **"Is an MCNN able to produce better results than a CNN when training both on noisy or incomplete data?"**. Additionally we have another smaller research question, with regard to the ability of both models to be trained on one set, and re-trained on another; **"Does an MCNN produce better results after previously being trained on another dataset, compared to its non-retrained self, and a retrained CNN?"**.

1.1 Algorithms

Some important algorithms for this research are the morphological and convolutional operations. As both models are implementing morphological and/or convolutional functions to process their input data, we will provide some insight into their functioning and differences.

For the purpose of MNNs, the morphological operation of erosion and dilation are used. These operations perform a weighted local minimum or local maximum operation for erosion and dilation respectively. An example of the erosion and dilation operations, performed on a 3x3 structuring element (SE) can be seen in Figure 1. The 3x3 SE has a set of weights, which are removed from or added to the pixel values during the erosion or dilation respectively. The MCNN is able to adjust these weights, and learn what weights produce the preferred result.



(a) Erosion with a 3x3 SE

(b) Dilation with a 3x3 SE

Figure 1: Morphological operations with a 3x3 SE, on a 7x7 patch [17]

In contrast, convolution applies a linear combination of several of the input pixels, as long as the weights exceed 0. The process of the convolutional operation can be observed in Figure 2. Here it can be seen that instead of adding to or subtracting from the input pixel values, the weights are multiplied with the pixel values, and added together to form the final value. Once again, these weights are learned by the CNN, to produce the optimal outcomes.

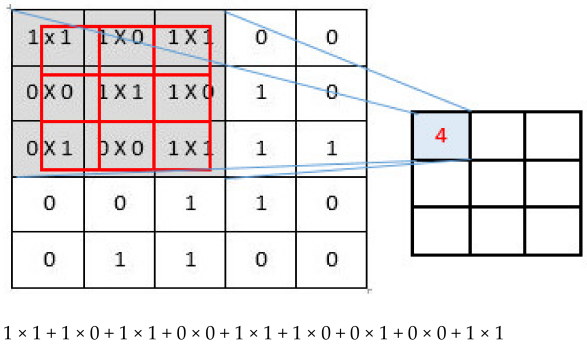


Figure 2: Convolution with a 3x3 SE, on a 5x5 patch [1]

1.2 Proposal

We propose to test in what ways MCNNs are able to extend the current state of the art of CNNs, and mitigate their limitations.
Several studies mentioned previously, have shown increased performance of MNNs, or MNN CNN hybrids compared to several CNN based counterparts. As a result, we know that MCNNs and MNNs have potential in several applications, but the exact benefits and conditions have not been fully explored. Improved performance has been shown, but does it also translate to lower requirements of an NN to achieve the same performance? Lower training data requirements would also be beneficial as a less long data gathering stage would be needed before an NN could be implemented in that specific setting. This leads to the following research questions:

- Is an MCNN able to perform as well as a CNN, with less training data?
- Can an MCNN perform better than a CNN, when trained on noisy or incomplete data?
- Is an MCNN able to outperform a CNN, after being retrained on a new unseen dataset?
- Does an MCNN need less retraining data than a CNN, to achieve similar performance, or is it able to perform better with the same amount?

Achieving the answer to these research questions could lead to the following contributions:

- Lower data requirements, allowing for noisy data, could make it possible for otherwise discarded data to be used.
- Less training data required, could make it easier to produce customized MCNNs.
- Higher performance after retraining on a new dataset, would make it more feasible to adapt an already highly trained model to a new set.
- Less retraining data would lower the cost of getting a pre-existing retrained for a new purpose.

Achieving these would reduce the load and cost on those trying to get a custom NN trained, as a result of fewer hours spent on annotating data, or lower cost from paying an expert to do it in their stead. One example is farmers wanting to get an NN to detect diseased leaves, but needing to annotate thousands of images with which leaves are healthy, and which ones are diseased.

2 Related Work

In this section we will describe the history and current state of neural networks, convolutional neural networks, morphological neural networks and their hybrid, the morphological convolutional neural network. The history and state of these subjects is relevant for the comparison between CNNs and MCNNs, which is the focus of this bachelor project.

2.1 History and background

NNs were first conceptualised in 1943 by Warren S. McCulloch and Walter Pitts [9] in an attempt to emulate biological neuron networks. A neural network consists of several layers of neurons, which are trained using a large amount of data. Once an NN has been trained, the NN should be able to make predictions about new data.

Convolutional neural networks were first explored in 1989 [6]. In this approach, the images are first processed using convolution, enabling edge and object detection. This initial processing enables the CNN to reduce the input image to smaller patches. This results in the NN part of the CNN not needing as many nodes in the layers as there are pixels in the image, but only as many as are in each patch. When each node has its weights that need to be trained, this reduction can have a significant impact on the amount of training data needed. After the convolution layers, there is a pooling layer, which scales down the input. Afterwards, there is the activation layer, which is able to filter or alter the input. Finally, there are the neural network layers which process the input into the final prediction [18]. The structure of a CNN can be viewed in Figure 3. The preprocessing that occurs, but more importantly the reduction in the number nodes, and therefore weights, in a CNN results in an increase in efficiency compared to NNs.

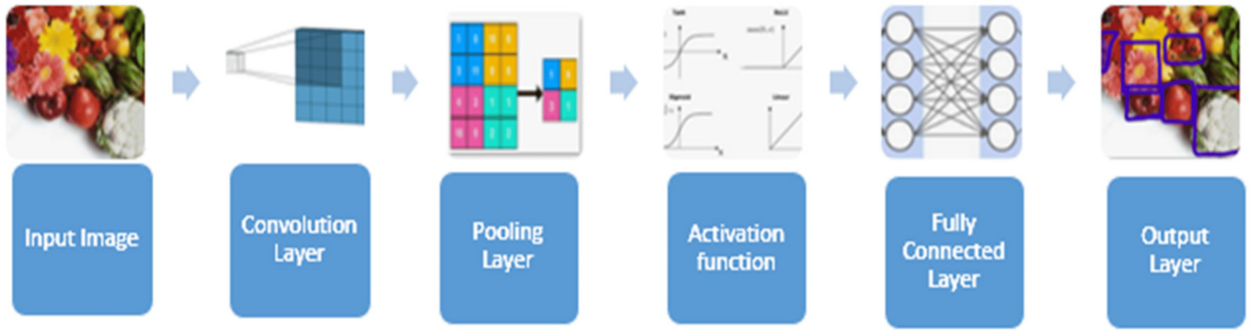


Figure 3: Visual representation of the layers of a CNN [1]

2.2 Types of CNN

As there are several layers of a CNN, there are many ways to adjust one or several of these layers, to affect the outcome. Another method of adjusting a CNN's functioning, is to perform operations before or in parallel with the CNN. Both of these types of adjusted CNNs will be discussed in the following section.

One type of CNN is the region-based convolutional neural network (R-CNN) [2]. An R-CNN extracts regions from the input image, which are individually passed to a CNN for classification. This allows object detection in an image rather than purely image classification.

One commonly used type of R-CNN is the "faster R-CNN" [15]. This version is an improvement of the "fast R-CNN" and the R-CNN. The faster R-CNN uses a fully convolutional network (FCN) to retrieve the regions from the input image. An FCN consists purely of Convolutional layers, and processes an image into semantically segmented output [7]. This method speeds up the process of retrieving the regions compared to previous versions, where it had a significant impact on the performance.

The Mask R-CNN [4] is an extension of the faster R-CNN, and performs masking on every region of interest in parallel with the CNN, using an FCN. This makes it possible for the output to consist of objects and classifications, with addition of a mask that shows what exact pixels belong to that object.

Lastly we have the CNN2D [13], a CNN adapted to be used for HSI images. This CNN enables feeding in 2D patches, along with their spectral bands into the neural network. This results in the network being able to take the underlying layers into account, when training on the patches. This is one of the two models we will be using in this thesis.

2.3 Morphological neural networks

MNNs are a variant of CNNs, first introduced in 1996 [16], using morphological operations instead of convolution. They work by replacing the convolution layers by layers with erosion and dilation. One main difference is that convolution is a linear operation, whereas dilation and erosion are nonlinear. As Roy et al. (2021) states "*nonlinear operations are able to better characterize the internal relationships and hidden patterns within complex remote sensing data, such as hyperspectral images (HSIs).*" [17]. Therefore, nonlinear functions can detect the more complex and hidden patterns within an image, and specifically within more complex images. Additionally, nonlinear functions might be able to achieve a higher accuracy, and more quickly, than the approximation using linear functions.

Nogueira et al. made use of a deep morphological network [11]. The network made use of morphological neuron layers, which were trained to perform simple and complex morphological operations. This enabled the network to outperform standard CNNs by approximately 5 percent in accuracy for pixel classification tasks.

One study combined morphology and convolution in so called MConv layers for letter recognition [10]. Here they tested using one of these layers, using two, and using several layers to replace those in a Deep CNN as a comparison to the state of the art. They achieved positive results in all, and managed to exceed the performance of the state of the art by using the combination of convolution and morphology.

The study performed by Roy et al. (2021) [17] concluded that their morphological CNN hybrid was able to outperform the classical CNNs in five widely used HSI datasets. The MorphConvHyperNet from this study, and the CNN2D they used for comparison, will be used to answer our research questions.

However, one direction that has not been explored, is whether this increased performance can be translated to lower requirements instead. This will be one of the main directions of this thesis.

2.4 Keywords

We have made use of web of science, using terms neural network, NN, convolutional neural network, CNN in the title, and image processing or computer vision in all fields, when looking for review papers.
For papers on MNNs, we used "morphological neural network*" or "morphological convolutional neural network*" or "morphological network*" in the title, with "image*" OR "computer vision", in all fields.

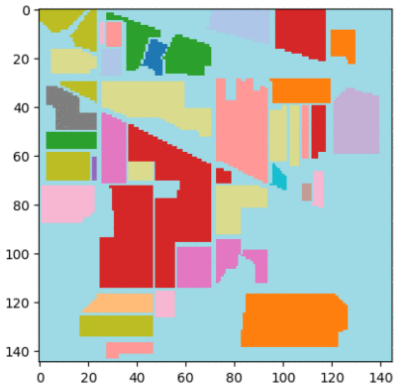
3 Materials and Methods

3.1 Data

The study we will be using as our starting point [17], contains measurements for several datasets, both for the unedited and disjointed versions. A disjointed dataset prevents testing pixels being visible in the surrounding neighbourhood of the training pixels. Training on a disjointed dataset results in the testing pixels and their neighbourhood, not being trained on, providing a more meaningful result.
During our research, we will not be making use of disjointed datasets. The reason for this choice is that the disjointed datasets were unavailable at the source mentioned by the article, at the time of this thesis. The unequal pixel division in the Indian Pines set and limitations in resources and time made it impractical to create a custom disjointed dataset.

As a consequence, for the training of the NNs, the Indian Pines (IP) and Pavia University (UP) datasets have been used. Both are Hyper Spectral Images (HSI), which provide a number of bands in different wavelengths. The images taken in these various bands are stacked on top of each other, and each pixel with its numerous bands has been labelled in the ground truth. These HSI datasets have been sourced from the Grupo de Inteligencia Computacional (GIC) [3].

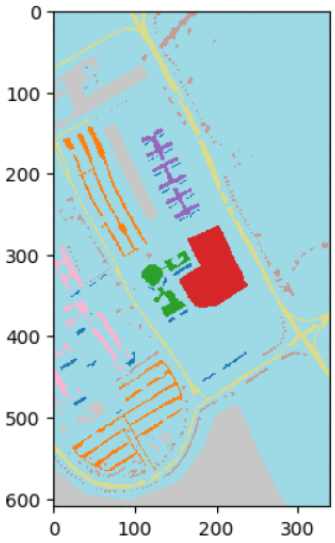
Most of the training and testing has been performed on the IP dataset, which consists of 145*145 pixels, with 200 spectral bands. The dataset has been gathered using an AVARIS sensor from a plane. This sensor is capable of gathering data in 220 different bands, however for this dataset certain bands are removed because of water absorption impacting the quality, reducing the number of bands to 200. The dataset contains pixels attributed to 16 different labels, with a distribution shown in Figure 4.



Color	Land cover type	Samples
	Background	10776
	Alfalfa	46
	Corn notill	1428
	Corn min	830
	Corn	237
	Grass/Pasture	483
	Grass/Trees	730
	Grass/pasture-mowed	28
	Hay windrowed	478
	Oats	20
	Soybeans notill	972
	Soybeans min	2455
	Soybean clean	593
	Wheat	205
	Woods	1265
	Bldg Grass Tree Drives	386
	Stone steel towers	93
	Total samples	21025

Figure 4: Ground truth of Indian Pines, with number of pixels per class [17]

For the purpose of testing retraining, the UP dataset has been used. This dataset consists of 610*610 pixels, with 103 spectral bands. The data has been gathered using a Rosis sensor with the use of a plane. The datasets has pixels with 9 different labels, their types and numbers are shown in Figure 5.



Color	Land cover type	Samples
	Background	164624
	Asphalt	6631
	Meadows	18649
	Gravel	2099
	Trees	3064
	Painted metal sheets	1345
	Bare Soil	5029
	Bitumen	1330
	Self Blocking Bricks	3682
	Shadows	947
	Total samples	207400

Figure 5: Ground truth of University of Pavia, with number of pixels per class [17]

3.2 Data Processing

The datasets consist of an image with several layers, however the dataset is not given to the NNs in their entirety. The images are first turned into patches, which consist of the labelled pixel at the centre, and the surrounding 11*11 pixel area, along with all spectral bands.
First the image gets padded, in case there are non-background pixels around the outside area, causing the surrounding patch to go out of bounds.
Afterwards, all non-background pixels are turned into 3D cubes, consisting of the 2D patch and their respective bands.
The created cubes are then split into a training and testing set, after which the testing set has a validation set split off. This validation set is used after each epoch, to determine the accuracy the NN achieves after training on the training set. The validation set is not trained on, but used as an in-between testing set, the metrics of which are then used by the model to improve.

3.3 Technology Stack

3.3.1 Software

To accomplish this thesis, we have made use of several different software and tools.
Firstly we require Docker Desktop. Docker is a software tool that makes it possible to run an instance of a different operating system, with its own libraries, on top of another operating system. For the project this was a requirement, as the libraries required by the project, needed specific Linux versions, while we were running this project on a Windows desktop.

To run Docker Desktop, WSL is needed. WSL stands for Windows Subsystem for Linux, which facilitates using Linux bash commands within a PowerShell environment. When the scope of the project is small enough, one can run a project completely within this environment, as if it were running on a Linux desktop. However, the scope of this project, and its requirements, made it unfeasible to rely on WSL alone.

Another major software we made use of, though technically optional, is the CUDA toolkit. This toolkit is essential to speed up the model training using the GPU, as opposed to being constrained to the CPU, resulting in a significant training speed increase. To accomplish this, the host needs to have an Nvidia GPU, along with its drivers, and the CUDA Container toolkit, enabling the Docker container to see the GPU. The container itself needs the CUDA toolkit and cuDNN, a deep neural network library that is required for training the models.

Additionally, Tensorflow and Keras were essential for this project. The MCNN makes use of Tensorflow, and a Keras version that is set up for Tensorflow. The CNN requires Keras on its own, without Tensorflow being required. We did decide to make the CNN use the same Tensorflow supported Keras version as the MCNN for consistency and preventing version issues.

As attempting to create a custom Docker image with Tensorflow and Cuda proved to be problematic, we made use of the nvcr.io/nvidia/tensorflow:25.02-tf2-py3 image. This docker image is made available by Nvidia, and comes with Tensorflow, accompanying CUDA libraries and Linux 24.04.

Scikit-learn library has been used for the purpose of processing the data into the train, testing and validation splits, and calculating the accuracies. This made it possible to interpret the performance of the models with the use of Overall Accuracy (OA) and Average Accuracy (AA).

Code editing was carried out using Python within Visual Studio Code, although any text editor with Python support should be sufficient. In our case it made it possible to neatly keep track of code and data produced, and provided a convenient environment for editing code.

3.3.2 Hardware

The hardware for the experiments consisted of a desktop computer with Windows 11, with a Nvidia 4080 GPU with 16 GB GDDR6X VRAM, accompanied by an AMD Ryzen 7 8900X3D 8-core CPU with 64 GB DDR5 RAM.

3.3.3 Experimental Setup

In order to compare the performance of a hybrid morphological convolutional neural network to a CNN, we have made sure to keep the setup as consistent as possible for both networks.

For experiments on the Indian Pines (IP) dataset, we have performed 10 runs of 500 epochs on each different variation. From these 10 separate runs, we then took the average OA and AA, and deviations thereof. These statistics were then used in the results section to compare the performance of the two models.

The choice of 10 runs has been made as a consequence of time constraints and the speed of the models, which differed significantly. The CNN was able to execute 10 runs of 500 epochs on 30% of the IP set in 12 minutes, this same session took the MCNN 112 minutes to complete. This effect was likely caused by the MCNN containing custom layers, which are not supported for compilation using the accelerated linear algebra (XLA) compiler. The CNN on the other hand was able to compile using XLA, resulting in significant optimizations.

Experiments on the Pavia University (UP) dataset, have been performed with 5 runs of 200 epochs, for each different experimental setup. This choice has been made as a consequence of the previously mentioned time and speed constraints. The reduction compared to the IP setup, was the result of the vastly increased number of pixels in the UP set compared to the IP set. The IP set contains 10.249 non-background pixels, with 200 bands resulting in 2.049.800 total pixels. The UP set contains 42.776 non-background pixels, with 103 bands, resulting in a total of 4.405.928 pixels. The increase in the number of pixels, combined with the lower number of unique labels, which in turn have a more equal pixel division, has resulted in the choice to reduce the number of epochs, and runs.

The exact setups with percentages for training and noise, will be mentioned in the results section, along with their respective results.

4 Models

To perform this bachelor thesis, we have made use of 2 NN models, namely the MorphConvHyperNet[17] and the CNN2D [13]. These two models were among the models compared in the study "*Morphological Convolutional Neural Networks for Hyperspectral Image Classification*" [17], which we have used as a starting point for our comparison. The CNN2D was chosen from the alternatives because of its highly similar performance compared to the MorphConvHyperNet, resulting in a more equal comparison.
For the MorphConvHyperNet, the official code was not available, therefore we have had to make use of the unofficial code found on github [5], which does refer to the official article. For the CNN2D we have been able to find the official code on github [12]. For ease of writing and reading, the MorphConvHyperNet will be referred to as MCNN, as it is a hybrid making use of both morphology, and convolution. The CNN2D will be referred to as CNN, as it is a purely convolutional network.

As the code has been found at different sources, we have chosen to adjust the MCNN code to align with the CNN. This choice was made as a result of the more complete codebase provided by the CNN, containing the methods for reading files, making patches and cubes, and taking in arguments to set up the runs. The layers and functionality of the MCNN has been fully preserved through this process, and adjustments allow a more similar test setup for both models. The code has received some adjustments, to fit more with the needs of this study.

4.1 Code Structure

As the layers of both individual models remain unedited, they will not be described during this section. Edits have been made to the surrounding code, supporting the operation of the two models, hence the overarching codebase will be broadly explained here.

4.1.1 Parser

One major aspect of the code is the parser, partially provided by the original CNN codebase. The parser makes it possible for in-line arguments to be given to the code to set experiment variables, for instance the number of runs and epochs. Additions and changes that have been made are as follows:

1. Training percentage modified to receive a list of training percentages and a minimum number of pixels.

2. Continuing from an existing file, and complete the number of missing runs, to not waste data after a run crashes.

3. Gaussian noise functionality, where a Gaussian deviation can be added.

4. Salt and pepper noise functionality, where a list of percentages for salt and pepper noise can be provided.
5. Salt and pepper 3D switch, making it possible for salt and pepper noise to be distributed randomly over all flattened indexes, as opposed to 2D where the noise across all bands is in the same indices.
6. Models, where one can add "MNN" and/or "CNN", which will be trained with the set arguments.

4.1.2 Patch Creation and Noise Edits

The input HSI image needs to be turned into patches of 11*11 pixels, across all bands. This patch cube creation is accomplished with the provided createImageCubes function from the CNN2D codebase. After the creation of the cubes, which also filters out the background, cubes are then optionally edited using Gaussian noise or salt and pepper noise.

4.1.3 Result Storage

For the purpose of interpreting the data at a later point, a large amount of data is stored from each run. This includes the pixels per label, for the training and testing set, the accuracies per label and the overall accuracy (OA), average accuracy (AA) and Cohen's Kappa score (K). These are all stored in a filename that contains data about the run, such as the model used to produce it, dataset, start time, runs, training percentage, epochs, batch size. This information facilitates sorting of the files at a later point, without having to open each individual file to find out what it contains.

4.1.4 Data Split

The pixel cubes have to be split into a train, test and validation set. Most of this section comes from the CNN2D codebase, however additions have been made, making it possible for a minimum number of trainpixels to be set, even when the training percentage would result in fewer pixels. This addition assists when training with lower percentages, on an unequal dataset such as Indian Pines, where one class, Soybean-mintill, has 2455 labeled pixels, whereas the Oats class has 20. Training with low percentages runs the risk of including 0 pixels for Oats in the training set, which is clearly suboptimal.

4.1.5 Training and Result Interpretation

The training is performed on the selected model, where the CNN is compiled using accelerated linear algebra (XLA), which is not possible for the MCNN. During the training, the best model is stored based on the val_accuracy, which is the accuracy it achieves on the unseen validation pixels, which it has not been trained on. At the end the best model is reloaded, from which the accuracies are found using the CNN2D codebase's report functionality, using sklearn. These accuracies are then stored in the created file, to be used at a later point. At the end of all runs, the average OA, AA and K are calculated and stored, along with their deviation, for the purpose of significance testing.

5 Results

During this section, the setups and results from several experiments will be provided. To compare the two models, the following experiments have been performed:

- The accuracy at certain training percentages (IP and UP).
- The accuracy at certain training pixel numbers (IP).
- The impact of Gaussian noise on accuracy (IP).
- The impact of salt and pepper noise of two varieties, on accuracy (IP).
- The performance of the models after retraining on the UP set, after prior training on the IP set.

In the following subsections, the graphs will show the Overall Accuracy (OA) and the Average Accuracy (AA) of varying setups.

The Overall Accuracy (OA) gives an indication of the accuracy achieved in the final test, the overall correct answers, divided by the total number of answers. This gives an insight into the performance one would expect when classifying a dataset similar to the one used for training. However with the IP dataset, the classes are not equally distributed, therefore the OA can be misleading. As a consequence of unequal distribution, some classes might be neglected, and achieve low accuracy after training with few pixels. Since there are few pixels of these classes in the testing set, their lack of accuracy will not impact the OA significantly.

Alternative to the OA we have the Average Accuracy (AA), representing the average between the individual accuracies per class. This results in each individual class accuracy having an impact of $\frac{1}{(number\_of\_classes)}$, this has the consequence of neglected classes with few pixels, being represented with the same weight as high performing classes with plenty of pixels. The AA therefore is able to represent whether overfitting is occurring, where some classes with high frequency are performing well, at the cost of classes with lower frequency.

Finally, the p value has been calculated for the results of each training percentage or training pixel number, to determine whether there is a significant difference between the performance of the MCNN and CNN. The p value is computed using Welch's two tailed t-test, also known as Welch's unequal variances t-test. We have chosen to make use of this alternative over the Student's t-test, as a result of the unequal variances produced by the experiments. We have selected the significance level (α) of 0.05 to determine the threshold, below which we declare a difference to be significant. Values below this threshold will be marked in bold in the tables in the following section.

In this section, we will additionally make some conclusions about the results achieved, as there are too many experiments for this to be done in the conclusion section alone.

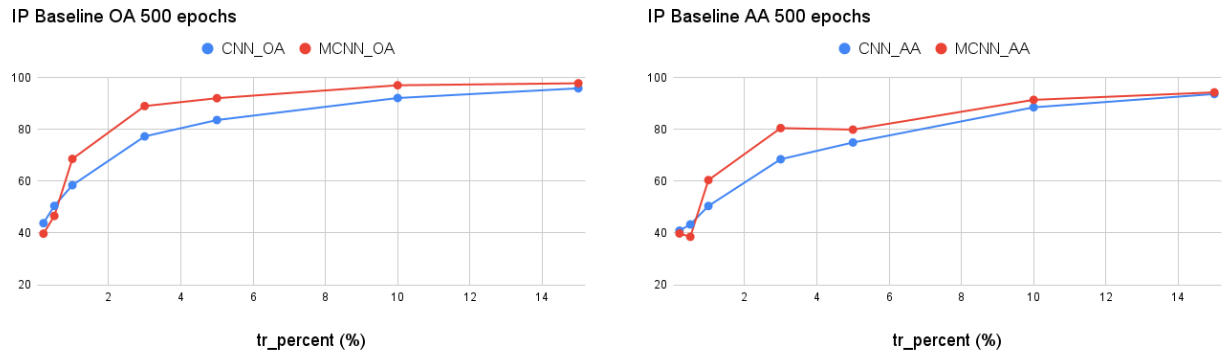
5.1 Baselines

The initial experiments that have been conducted, are to verify our setup achieves similar results to those in "*Morphological Convolutional Neural Networks for Hyperspectral Image Classification*" [17].
The percentages used in that study are 5 runs of 200 epochs, however for the purpose of statistical testing, we have increased the numbers to 10. Additionally, as we saw a difference in performance between 200 epochs and 500, we chose to run 500 epochs, to find the maximum performance that can be achieved using certain training percentages, the difference can be seen in this section.

5.1.1 IP Baselines

The IP dataset with varying training percentages has been used for the purpose of establishing a baseline, and verifying whether our results align with those achieved in the MCNN study. The percentages are used, up to a minimum of 1 pixel per class, to prevent a small class from receiving 0 pixels in the training set. The baseline can be seen in Figure 6, where the results from varying percentages of the training set are displayed. In Table 1, the numerical values of the OAs and AAs can be seen, along with their calculated p value. In Appendix B we have added some residual maps, showing where both models make their mistakes.

As the table shows, the OA between the two models differs significantly, where with training percentage, tr_percent, of 0.2% and 0.5%, the CNN outperforms the MCNN, however the OA achieved is does not exceed 51%. From a training percentage of 1% and higher, the MCNN performs significantly better than the CNN, with regard to the OA.
In terms of the AA, the difference between the CNN and MCNN is not significant for training percentages 0.2%, 0.5%, and 15%, for all other training percentages, the MCNN achieves significantly higher accuracies than the CNN.



(a) Overall Accuracy

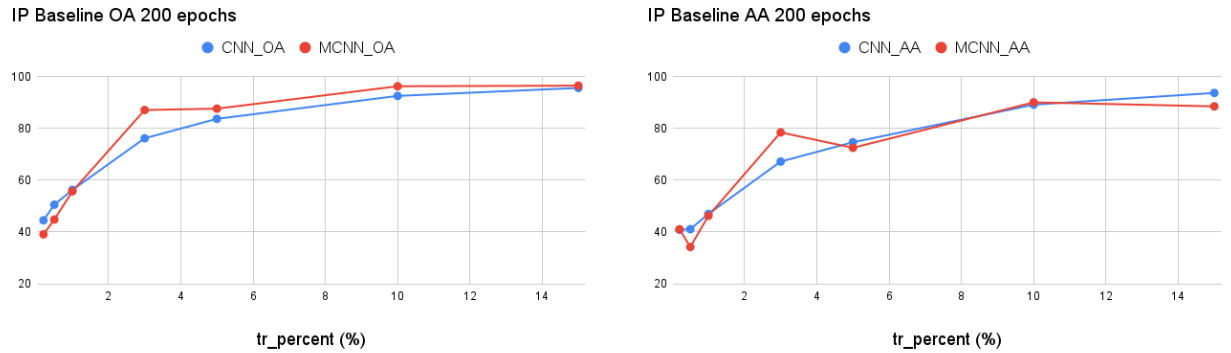
(b) Average Accuracy

Figure 6: IP training with varying training percentages, OA and AA, 500 epochs

tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_AA	MCNN_AA	p value
0.2	43.779	39.731	**3.24E-02**		0.2	40.896	39.799	6.22E-01
0.5	50.449	46.576	**5.89E-03**		0.5	43.287	38.516	5.63E-02
1	58.456	68.600	**2.45E-07**		1	50.412	60.415	**2.31E-05**
3	77.295	89.004	**5.75E-12**		3	68.473	80.501	**3.37E-08**
5	83.609	92.039	**4.64E-10**		5	74.934	79.910	**6.12E-04**
10	92.113	97.037	**1.22E-09**		10	88.517	91.407	**1.45E-02**
15	95.854	97.807	**9.75E-07**		15	93.666	94.296	5.18E-01

Table 1: OA, AA and calculated p value for varying training percentages on the IP set, 500 epochs

We have also included a test using 200 epochs for training as opposed to 500 epochs, the results of which can be seen in Figure 7 and Table 2. When compared to Figure 6 and Table 1, it can be seen that after training for 200 epochs, the MCNN outperforms the CNN in fewer cases than with 500 epochs.



(a) Overall Accuracy

(b) Average Accuracy

Figure 7: IP training with varying training percentages, OA and AA, 200 epochs

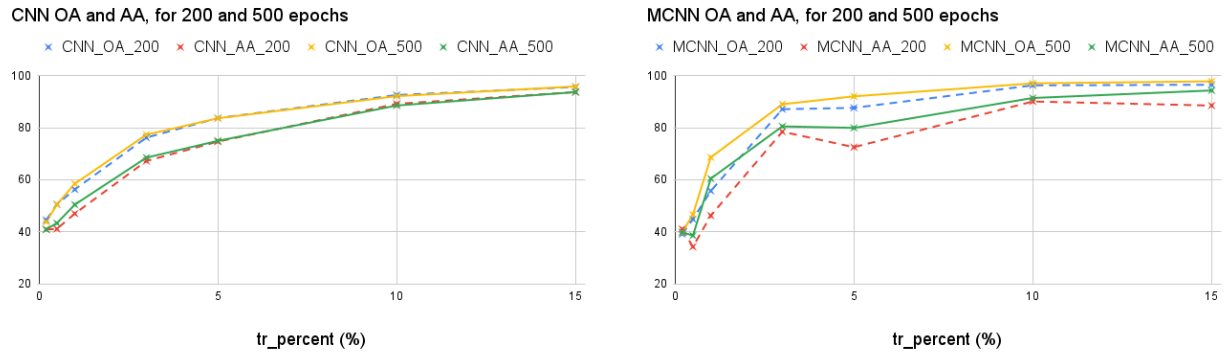
tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_AA	MCNN_AA	p value
0.2	44.492	39.085	**6.46E-04**		0.2	40.858	41.040	9.20E-01
0.5	50.522	44.801	**1.23E-02**		0.5	41.079	34.203	**1.20E-02**
1	56.245	55.683	6.53E-01		1	46.957	46.217	7.00E-01
3	76.175	87.094	**1.50E-11**		3	67.180	78.456	**8.99E-08**
5	83.693	87.616	**5.35E-03**		5	74.646	72.495	1.18E-01
10	92.541	96.269	**7.84E-08**		10	89.156	90.057	4.32E-01
15	95.621	96.506	**1.45E-02**		15	93.692	88.501	**3.78E-03**

Table 2: OA, AA and calculated p value for varying training percentages on the IP set, 200 epochs

In Figure 8 it can be seen that the values for the CNN remain mostly similar, whereas the MCNN appears to have large improvements in some cases. This is confirmed in the significance testing, the results of which can be found in Table 3.

With regard to the CNN, in Table 3 it can be seen that the difference in epochs, results in a significant difference in OA and AA in only 1 case. This shows that the CNN does not benefit from an increase in epochs, for most cases.

In terms of the MCNN, the table reveals that in 4 cases the OA is increased significantly as a result of the larger number of epochs. Additionally the AA is significantly higher in 3 cases. Therefore it can be concluded that in many cases, the MCNN does improve both types of accuracy with an increase of epochs.



(a) CNN comparison

(b) MCNN comparison

Figure 8: IP training for 200 and 500 epochs, visualised per model, OA and AA

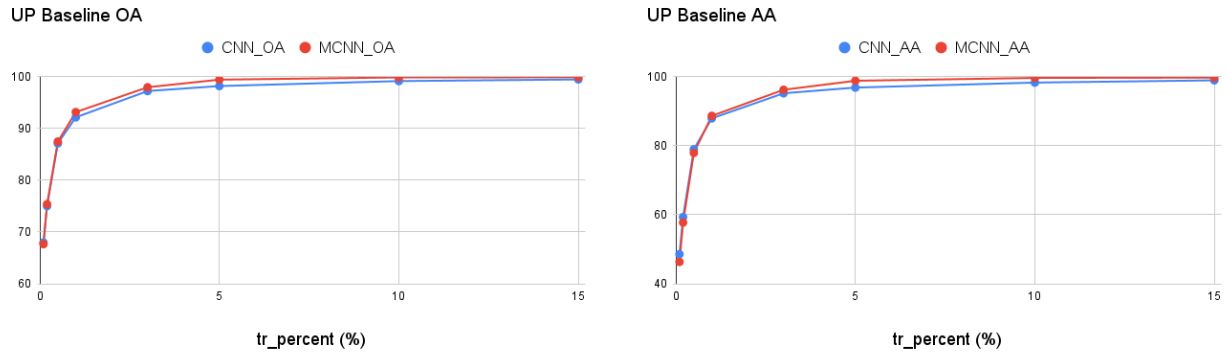
tr_percent	CNN_OA_200	CNN_OA_500	p value		tr_percent	CNN_AA_200	CNN_AA_500	p value
0.2	44.492	43.779	5.41E-01		0.2	40.858	40.896	9.83E-01
0.5	50.522	50.449	9.39E-01		0.5	41.079	43.287	3.08E-01
1	56.245	58.456	**3.07E-02**		1	46.957	50.412	**5.73E-03**
3	76.175	77.295	7.66E-02		3	67.180	68.473	3.18E-01
5	83.693	83.609	8.49E-01		5	74.646	74.934	8.05E-01
10	92.541	92.113	2.90E-01		10	89.156	88.517	4.56E-01
15	95.621	95.854	4.28E-01		15	93.692	93.666	9.78E-01
tr_percent	MCNN_OA_200	MCNN_OA_500	p value		tr_percent	MCNN_AA_200	MCNN_AA_500	p value
0.2	39.085	39.731	7.23E-01		0.2	41.040	39.799	5.85E-01
0.5	44.801	46.576	3.99E-01		0.5	34.203	38.516	1.16E-01
1	55.683	68.600	**2.19E-08**		1	46.217	60.415	**7.91E-06**
3	87.094	89.004	**2.13E-02**		3	78.456	80.501	1.30E-01
5	87.616	92.039	**2.48E-03**		5	72.495	79.910	**3.29E-05**
10	96.269	97.037	5.16E-02		10	90.057	91.407	3.08E-01
15	96.506	97.807	**5.70E-04**		15	88.501	94.296	**1.82E-03**

Table 3: Comparison for both models trained for 200 or 500 epochs, OA, AA and calculated p value

5.1.2 UP Baselines

The University of Pavia dataset, is a larger, more equally distributed dataset with fewer unique classes. Therefore the choice has been made to use lower training percentages, to make a visible improvement possible after retraining a model that has been trained on the IP set.

The results of the tests performed on the UP set can be seen in Figure 9 and Table 4. Here it can be observed that the MCNN achieves a significantly higher OA starting from a training percentage of 1%, and a significant improvement in the AA compared to the CNN, starting from a training percentage of 3%.



(a) Overall Accuracy

(b) Average Accuracy

Figure 9: UP training with varying training percentages, OA and AA, 200 epochs

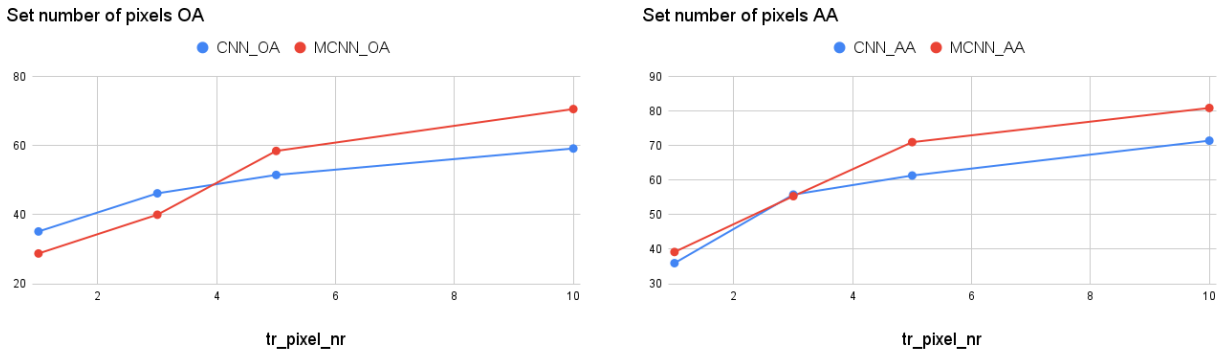
tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_AA	MCNN_AA	p value
0.1	67.951	67.652	8.35E-01		0.1	48.569	46.326	2.20E-01
0.2	75.002	75.356	7.75E-01		0.2	59.305	57.741	6.66E-01
0.5	87.099	87.456	6.73E-01		0.5	78.946	77.893	6.69E-01
1	92.136	93.144	**4.52E-02**		1	87.902	88.630	5.30E-01
3	97.211	97.942	**3.29E-03**		3	95.173	96.176	**5.07E-03**
5	98.177	99.409	**1.99E-05**		5	96.822	98.792	**8.79E-04**
10	99.135	99.843	**4.91E-08**		10	98.267	99.615	**2.62E-06**
15	99.449	99.864	**1.80E-03**		15	98.919	99.710	**3.06E-03**

Table 4: OA, AA and calculated p value for varying training percentages on the UP set

5.2 Set Number of Training Pixels

Another small experiment we performed, is one where the models were trained with a set number of training pixels, as opposed to a training percentage. With the IP dataset, where the classes are unequally divided, this simulated the training of a set with a more equal class division. This experiment also extends the baseline test by training with fewer pixels for most classes.

The results of this experiment can be seen in Figure 10 and in Table 5. The table shows that the OA for each training pixel number differs significantly. For the first two training percentages, the CNN outperforms the MCNN significantly, whereas the MCNN outperforms the CNN in the latter two percentages. However due to the small sample size, and large deviation, the difference is on the border of being significant. For the AA, we cannot state that every number of pixels used for training produces a significant difference, as the p value remains above the threshold of 0.05 for 1 and 3 training pixels. The table does show a significant difference for 5 and 10 training pixels.



(a) Overall Accuracy

(b) Average Accuracy

Figure 10: Training with set number of pixels, OA and AA

tr_pixels	CNN_OA	MCNN_OA	p value		tr_pixels	CNN_AA	MCNN_AA	p value
1	35.141	28.763	**2.55E-02**		1	35.965	39.195	2.70E-01
3	46.185	39.991	**1.14E-02**		3	55.822	55.363	8.05E-01
5	51.516	58.468	**4.58E-05**		5	61.345	71.009	**8.22E-07**
10	59.174	70.598	**2.18E-09**		10	71.439	80.910	**5.24E-11**

Table 5: Set number of pixels, OA, AA and calculated p value

5.3 Training Data Conclusions

In this section, we have made conclusions about both models' performance with varying training percentages and numbers of pixels. Both the peak performance, and data requirements will be discussed.

5.3.1 Indian Pines Dataset

When having been trained on the IP set, the MCNN achieves significantly higher scores after 500 epochs, compared to 200 epochs. This validates our choice in increasing the number of epochs to 500. The CNN benefits significantly in some cases, however the MCNN is able to significantly outperform the CNN in more cases, after having been trained for 500 epochs compared to 200 epochs.

Performance:
After a training percentage of 1%, the MCNN is able to outperform the CNN by up to 11.7% with regard to overall accuracy (OA), though this becomes more equal once we reach a training percentage of 10% and 15%.
In terms of average accuracy (AA), starting from 1% of the IP set, the MCNN is able to achieve higher AA with up to 11.9% difference.

Lower data requirements:
The MCNN is able to get to 92% OA with 5% of the IP set, whereas the CNN needs 10% of to achieve this OA. The OA of 97% of the MCNN at 10% training data, is not exceeded by the CNN with 15% training data. For the AA, the difference is smaller, where the CNN with training percentage of 0.2% can beat the MCNN using 0.5% of the training data. This is flipped at training percentage of 3% for the MCNN beating 5% for the CNN.

5.3.2 University of Pavia Dataset

Performance:
After 0.2% the MCNN performs better with regard to OA, however the difference in this case only goes up to around 1.2%. For the AA, the MCNN exceeds the CNN from 0.5% onwards, achieving an up to nearly 2% higher AA.

Lower data requirements:
The MCNN needs 5% of the UP set to achieve and OA higher than 99.4%, whereas the CNN needs 15% training data to exceed the 99.4% accuracy threshold. The 99.8% OA achieved by the MCNN at 10% is not exceeded by the CNN with 15% of the training data.
With regard to the AA, the MCNN achieves an accuracy of around 98.8% with 5% of the training data, however to exceed this, the CNN needs 15% of the training set, achieving an accuracy of 98.9%. Once again the CNN is unable to exceed the AA the MCNN achieves with 10% of the data, when it gets provided with 15% of the UP set.

5.3.3 Set Number of Training Pixels

Performance:
Starting from 5 pixels, the MCNN is able to significantly outperform the CNN, with up to 11.4% higher accuracy in terms of OA, and up to 9.6% improved average accuracy.

Lower data requirements:
To exceed an OA of 58%, the MCNN needs 5 pixels to train on, to exceed this same threshold, the CNN needs 10 training pixels. With regard to the average accuracy, at 5 pixels the MCNN achieves an accuracy of 71%, which once again the CNN needs to train on 10 pixels to exceed.

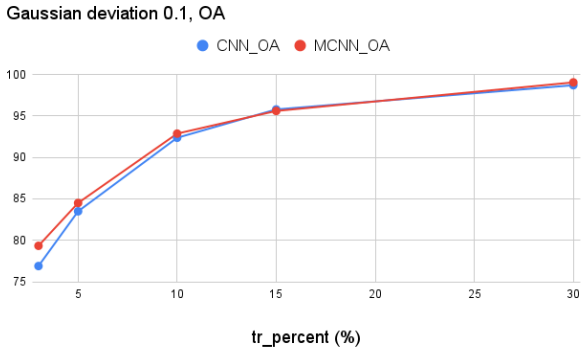
5.4 Gaussian Noise

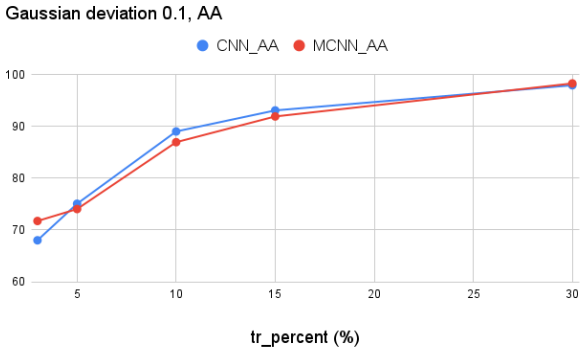
One type of noise that can affect an image, is simulated by Gaussian noise. This type of noise is distributed according to the normal distribution, with a provided standard deviation around a mean of 0. It will increase or decrease the value that is attributed to each pixel, simulating a less precise measuring device. In a neighbourhood, the noise averages out, as negative and positive values are added to all pixels.
As a result, convolution should be better able to deal with this type of noise, as all pixels in a neighbourhood are taken into account, and therefore the final result should be affected in a less extreme way. Morphology on the other hand, picks a single pixel to determine its value, either the max or min, depending on the operation. Therefore the full noise will be passed onto the next layer.

This effect is shown in Figure 11 and Table 6. Here we can see that only in some cases for the least amount of noise added, namely Gaussian deviation 0.1, the MCNN is able to slightly outperform the CNN. However this performance difference is only significant for a training percentage of 3% and 30% in terms of OA, and only 3% shows significant superior performance compared to the CNN, in terms of AA. Nearly all the other tests with varying Gaussian deviation show that the CNN significantly outperforms the MCNN.

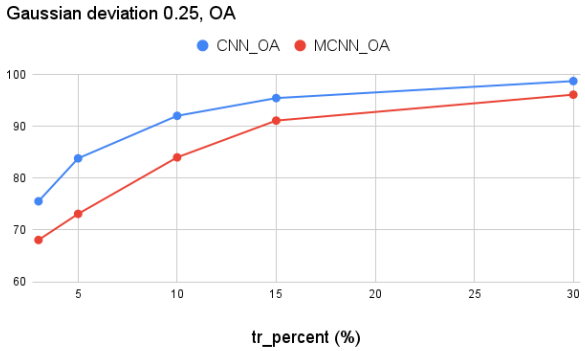
Gauss deviation 0.1					Gauss deviation 0.25					Gauss deviation 0.5			
tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_OA	MCNN_OA	p value
3	76.894	79.335	**1.42E-02**		3	75.533	68.063	**3.13E-05**		3	75.590	61.865	**4.81E-10**
5	83.500	84.506	1.99E-01		5	83.824	73.102	**1.03E-05**		5	82.438	65.017	**5.79E-07**
10	92.374	92.878	4.15E-01		10	92.040	84.015	**1.24E-04**		10	91.201	77.118	**1.17E-05**
15	95.797	95.612	6.34E-01		15	95.462	91.121	**7.41E-03**		15	94.921	84.192	**1.45E-05**
30	98.717	99.060	**5.98E-03**		30	98.745	96.123	**5.05E-04**		30	98.429	93.428	**1.32E-04**
tr_percent	CNN_AA	MCNN_AA	p value		tr_percent	CNN_AA	MCNN_AA	p value		tr_percent	CNN_AA	MCNN_AA	p value
3	68.003	71.729	**9.45E-03**		3	66.824	61.858	**6.17E-03**		3	66.588	52.163	**4.16E-06**
5	75.061	74.040	5.50E-01		5	74.856	64.226	**1.98E-05**		5	73.765	57.068	**1.34E-09**
10	89.017	86.949	1.93E-01		10	88.875	81.593	**2.37E-05**		10	87.717	74.801	**2.21E-04**
15	93.083	91.916	1.58E-01		15	92.592	89.472	6.04E-02		15	92.842	83.507	**7.61E-05**
30	97.937	98.293	3.04E-01		30	98.134	97.006	**1.59E-02**		30	98.055	94.153	**7.73E-05**

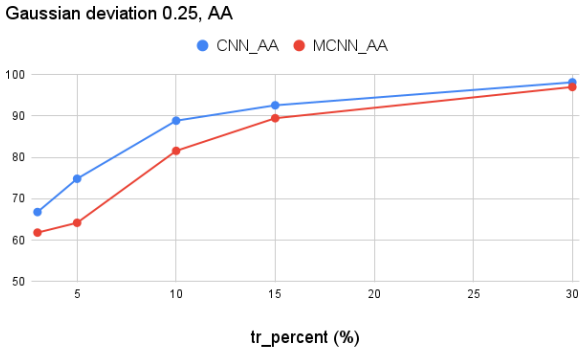
Table 6: Gaussian noise, OA, AA and calculated p value



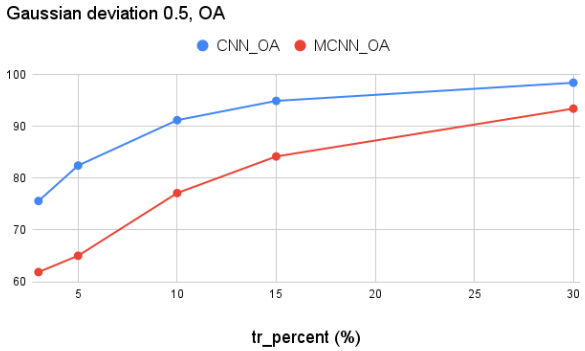


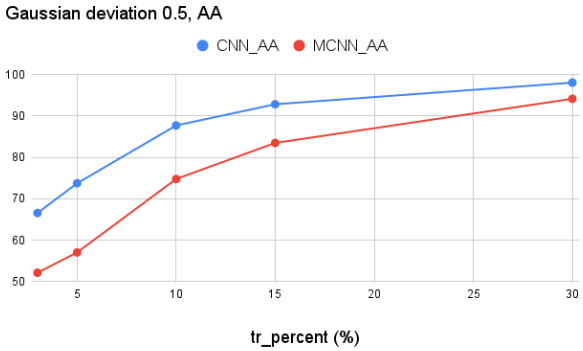
(a) Overall and average accuracy with Gaussian deviation 0.1





(b) Overall and average accuracy with Gaussian deviation 0.25





(c) Overall and average accuracy with Gaussian deviation 0.5

Figure 11: Training after adding Gaussian noise, OA and AA

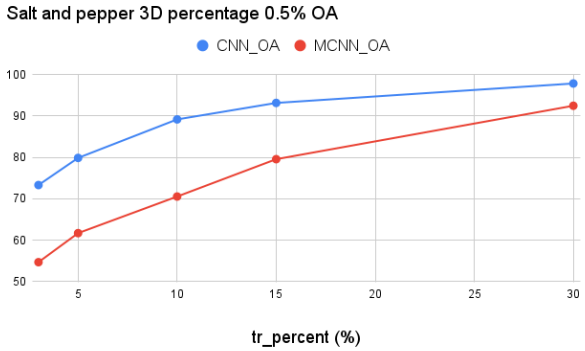
5.5 Salt and Pepper Noise

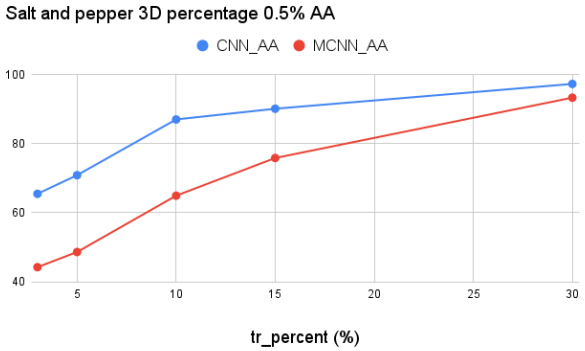
Salt and pepper noise turns certain randomly selected pixels into the minimum or maximum value of the image, such as black and white pixels on a greyscale image. Two varieties of salt and pepper noise have been used. For one type of noise the pixels were randomly distributed over all pixels and bands, which we will refer to as 3D noise, also known as spectrally uncorrelated. The other type of noise was added to the 2D patch, and then carried through to the following bands, resulting in the noisy pixels being on the same 2D coordinates throughout every band. This second type of noise will be referred to as 2D noise, also known as spectrally correlated.

5.5.1 3D Noise

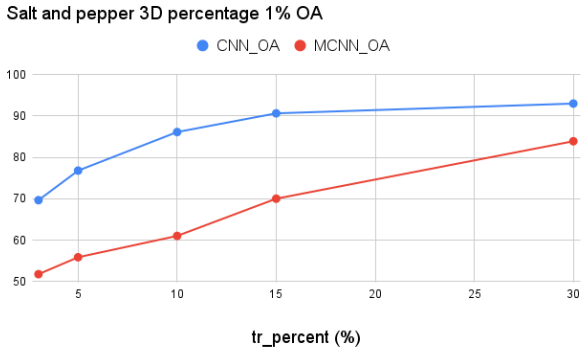
In this section, each layer can contain noisy pixels, at random indices. Due to an error, the "replace" variable was left on, resulting in a specific pixel possibly being selected repeatedly, instead of exclusively unique pixels. This error might make a slight difference in the results, compared to a run without replacement, however the overall comparison between MCNN and CNN should still be valid.

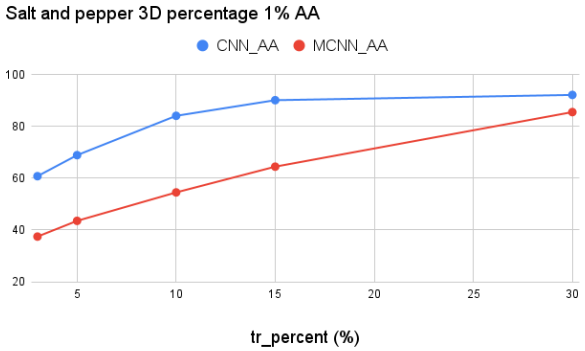
As can be seen from Figure 12, for all tested salt and pepper noise percentages across the 3D plane, the CNN performs better than the MCNN. It also shows that the MCNN does close the gap with lower noise percentages, and higher training percentages. Table 7 shows that both the OA and AA performance of the CNN is significantly greater in all tests compared to the MCNN. One possible explanation to this could be that the morphological operation might be better able to fill in gaps, but is also more dependent on structures across the bands. Therefore noise randomly distributed across all bands, results in structures on one band being less visible due to noise in another band.



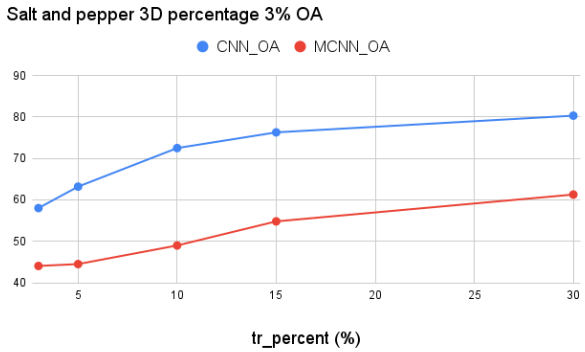


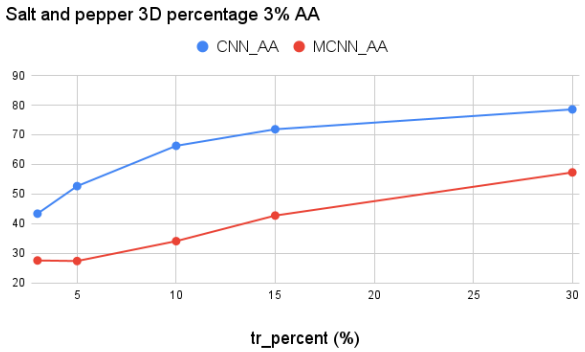
(a) Overall and average accuracy for 3D salt and pepper noise of 0.5%



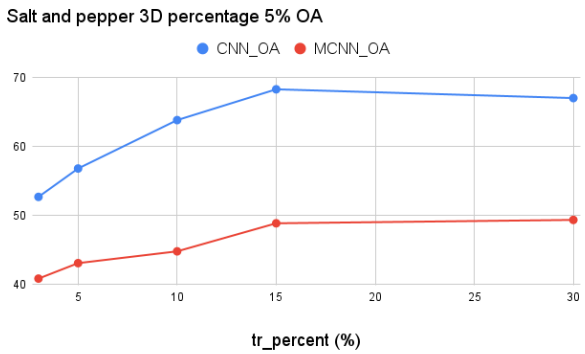


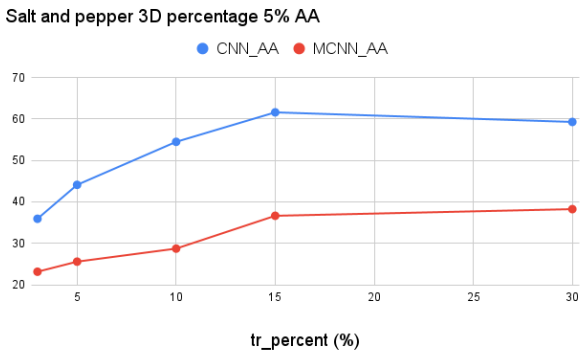
(b) Overall and average accuracy for 3D salt and pepper noise of 1%





(c) Overall and average accuracy for 3D salt and pepper noise of 3%





(d) Overall and average accuracy for 3D salt and pepper noise of 5%

Figure 12: Training after adding 3D salt and pepper noise, OA and AA

salt and pepper percentage 0.5%					salt and pepper percentage 1%			
tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_OA	MCNN_OA	p value
3	73.337	54.721	**9.57E-10**		3	69.686	51.824	**7.45E-10**
5	79.889	61.734	**8.88E-16**		5	76.823	55.908	**1.99E-08**
10	89.186	70.567	**1.80E-09**		10	86.133	61.061	**2.07E-11**
15	93.162	79.584	**1.63E-06**		15	90.649	70.047	**3.36E-07**
30	97.853	92.480	**3.08E-03**		30	93.005	83.913	**3.02E-05**
tr_percent	CNN_AA	MCNN_AA	p value		tr_percent	CNN_AA	MCNN_AA	p value
3	65.467	44.216	**6.81E-10**		3	60.765	37.436	**7.70E-09**
5	70.864	48.608	**1.34E-11**		5	68.890	43.517	**3.76E-08**
10	87.012	64.926	**1.32E-06**		10	84.079	54.496	**1.91E-07**
15	90.113	75.841	**6.32E-07**		15	90.115	64.441	**2.27E-07**
30	97.302	93.315	**1.87E-04**		30	92.149	85.516	**1.33E-03**
salt and pepper percentage 3%					salt and pepper percentage 5%			
tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_OA	MCNN_OA	p value
3	58.019	44.052	**1.30E-12**		3	52.723	40.889	**2.00E-07**
5	63.200	44.505	**3.25E-13**		5	56.834	43.128	**2.20E-11**
10	72.503	48.994	**3.33E-15**		10	63.843	44.837	**1.13E-14**
15	76.298	54.811	**6.69E-07**		15	68.305	48.900	**9.16E-10**
30	80.327	61.286	**1.02E-10**		30	67.032	49.388	**4.15E-12**
tr_percent	CNN_AA	MCNN_AA	p value		tr_percent	CNN_AA	MCNN_AA	p value
3	43.340	27.524	**8.22E-10**		3	35.922	23.142	**2.66E-08**
5	52.644	27.335	**7.23E-13**		5	44.110	25.574	**1.90E-08**
10	66.270	34.054	**3.04E-12**		10	54.493	28.721	**3.64E-10**
15	71.864	42.702	**2.50E-07**		15	61.611	36.644	**2.59E-08**
30	78.582	57.280	**2.30E-07**		30	59.294	38.248	**2.85E-08**

Table 7: 3D salt and pepper noise, OA, AA and calculated p value

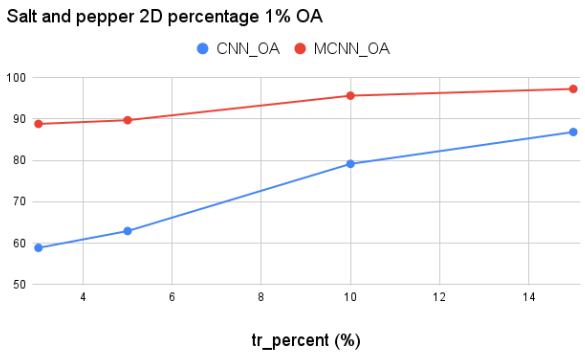
5.5.2 2D Noise

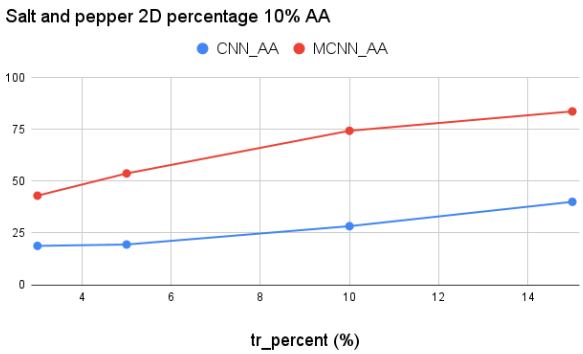
Alternative to the 3D noise, we have the 2D noise, where noisy pixels on the band 0 patch are carried through all bands. This results in the same gaps existing in structures on all bands, which the MCNN might be able to learn to deal with.

The results of these runs can be observed in Figure 13, where it can be seen that the MCNN is achieving higher accuracy than the CNN, as opposed to with the 3D metrics. Table 8 shows that these differences are significant for each experiment and training percentage, for both the OA and AA.

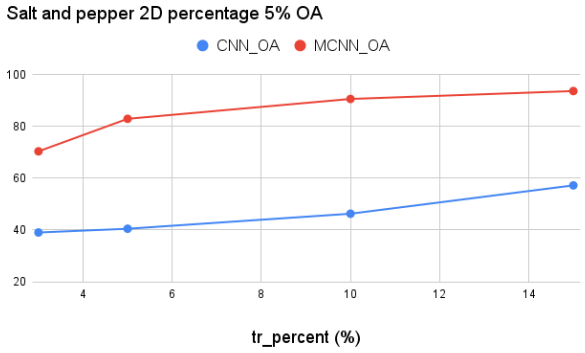
salt and pepper percentage 1%					salt and pepper percentage 5%					salt and pepper percentage 10%			
tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_OA	MCNN_OA	p value
3	58.909	88.824	**7.19E-14**		3	39.022	70.360	**2.63E-08**		3	36.190	61.898	**2.39E-08**
5	62.961	89.714	**4.44E-16**		5	40.468	82.926	**2.51E-14**		5	36.717	69.661	**1.58E-06**
10	79.171	95.633	**3.49E-14**		10	46.268	90.591	**2.16E-13**		10	41.932	83.697	**1.27E-07**
15	86.858	97.260	**1.24E-13**		15	57.213	93.643	**2.22E-16**		15	49.388	90.727	**0.00E+00**
tr_percent	CNN_AA	MCNN_AA	p value		tr_percent	CNN_AA	MCNN_AA	p value		tr_percent	CNN_AA	MCNN_AA	p value
3	47.279	80.247	**6.19E-13**		3	20.352	52.978	**1.23E-06**		3	18.759	43.001	**2.13E-06**
5	52.722	77.176	**2.17E-09**		5	24.624	67.176	**2.18E-10**		5	19.420	53.730	**1.01E-05**
10	73.369	87.413	**6.77E-09**		10	35.738	81.892	**2.67E-11**		10	28.273	74.328	**8.09E-07**
15	82.550	94.381	**3.85E-09**		15	47.670	86.425	**7.62E-10**		15	40.026	83.681	**3.54E-11**

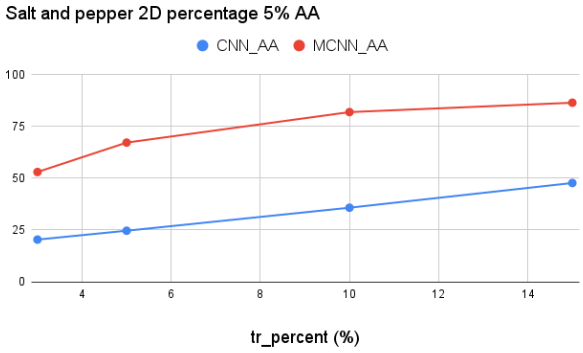
Table 8: 2D salt and pepper noise, OA, AA and calculated p value



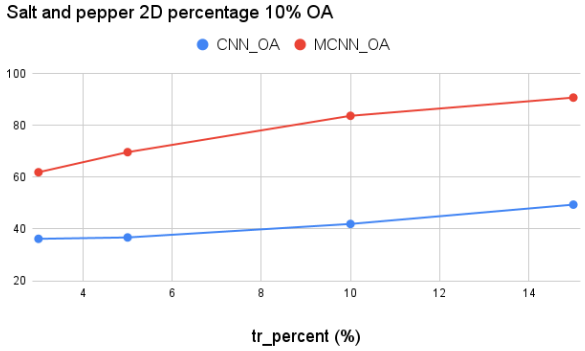


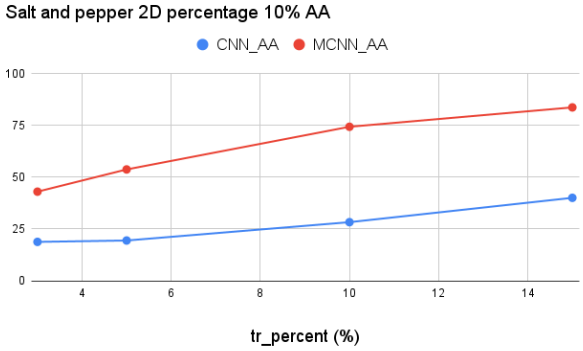
(a) Overall and average accuracy for 2D salt and pepper noise of 1%





(b) Overall and average accuracy for 2D salt and pepper noise of 5%





(c) Overall and average accuracy for 2D salt and pepper noise of 10%

Figure 13: Training after adding 2D salt and pepper noise, OA and AA

5.6 Noise Conclusions

In this section we will draw some conclusions about the ability for both models to deal with varying types of noise, and on how the two models compare to each other.

With regard to Gaussian noise, the CNN outperforms the MCNN consistently starting from a Gaussian deviation of 0.1. Before this point, the MCNN is able to exceed the CNN in some cases, or the differences are not significant enough to draw conclusions from.

In terms of 3D salt and pepper noise, once again the CNN outperforms the MCNN in every single training percentage and salt and pepper percentage. This shows that the MCNN is not very capable of adapting to this type of noise, whereas the CNN is able to adapt better comparatively.

Lastly we have 2D salt and pepper noise, here the MCNN is vastly better at adapting to the changes. The MCNN achieves up to 44% higher overall accuracy, which is achieved with a salt and pepper percentage of 5%, and a training percentage of 10. The largest AA difference is achieved with a total of 46% difference. This shows that the MCNN is able to better adapt to the 2D salt and pepper noise compared to the CNN.

5.7 Retraining

The final experiment executed to compare the MCNN and CNN, was when attempting to train on a certain percentage of a new dataset, namely the UP dataset, after having been trained on a set percentage of the IP set. This experiment might show whether the MCNN is able to better transfer what it has learned in one set, when applied to a new set, compared to the CNN.

Several adjustments had to be made between the IP training session and the UP training. One was the removal of the final layer, and therefore what that layer learned, of the both models, as their functioning was dependent of the number of classes in the dataset. Another change that had to be made was to pad the UP set with 97 empty bands, as the IP set contains 200 spectral bands, where the UP set contains 103. This reduces some of the lessons the model has learned from the IP set.
Each experiment is run 5 times, these 5 runs each use a different model trained on their respective percentage of the IP set, however this model is reused between different setups. Therefore, there are 10 trained CNN and MCNN models, 5 with a training percentage of 5%, and 5 with a training percentage of 30%. This choice might have an impact on the validity of the results, however due to time constraints we have chosen for this setup.

Firstly we will use the baseline of the UP set, to compare the performance of the retrained MCNN and CNN to. In this case, there are two relevant sets of results. One result that we will interpret is to compare the performance of the MCNN and CNN after retraining. The second result is whether retraining has resulted in a significant improvement in performance, compared to training on the same amount of UP data, without prior training on IP data. Both of these results can be seen in Figure 14.

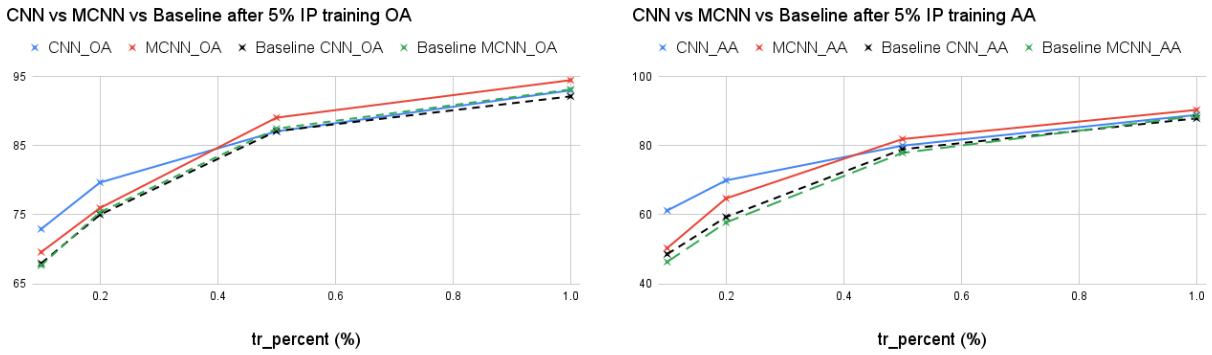
5.7.1 MCNN Versus CNN

The comparison and significance between MCNN and CNN can be viewed in Table 9. In this table we can see that there is a significant difference in performance between the CNN and MCNN.
After having been trained on 5% of the IP set and training on 0.2% of the UP set, the CNN performs significantly better than the MCNN in terms of OA, this performance difference is reversed for the 0.5% and 1%, where the MCNN outperforms the CNN. The AA is exclusively significantly greater for the CNN after training on 0.1% of the UP set. These results are interesting, as in Table 4, the only significant difference within this percentage range can be found in the case of 1% training, where the OA is significantly higher for the MCNN compared to the CNN. Therefore the retraining has caused a larger number of significant performance differences between the two models within this UP training percentage range.
After training on 30% of the IP set, the OA exclusively differs with a high significance after training on 1% of the UP set, in favour of the MCNN. After training on 0.1% of the UP set, the CNN performs better than the MCNN with regard to the AA.

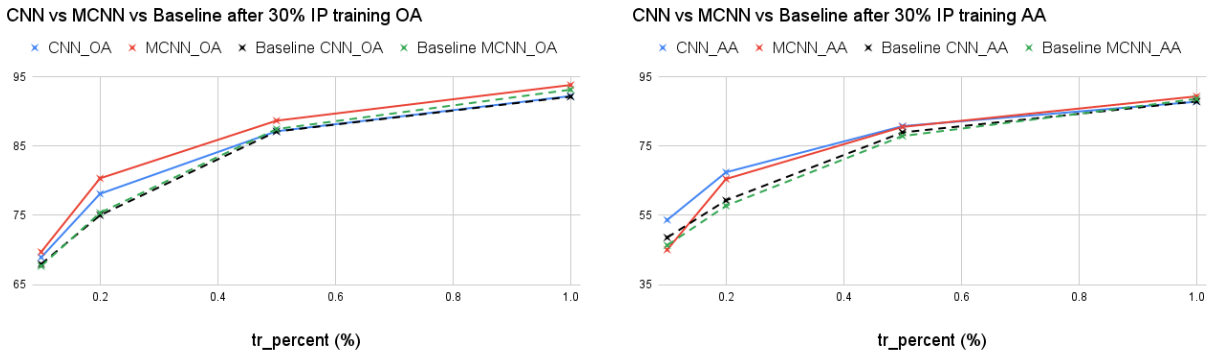
In conclusion, the retraining does result in a larger number of significant performance differences, where previous training on 5% of the IP set appears to have the largest benefits.

Trained on 5% IP					Trained on 30% IP			
tr_percent	CNN_OA	MCNN_OA	p value		tr_percent	CNN_OA	MCNN_OA	p value
0.1	72.926	69.583	1.64E-01		0.1	68.928	69.706	6.66E-01
0.2	79.663	75.971	**3.19E-02**		0.2	78.101	80.337	1.02E-01
0.5	87.060	89.055	**3.73E-02**		0.5	87.141	88.666	1.82E-01
1	93.010	94.484	**1.72E-02**		1	92.222	93.819	**6.74E-03**
tr_percent	CNN_AA	MCNN_AA	p value		tr_percent	CNN_AA	MCNN_AA	p value
0.1	61.198	50.315	**1.15E-02**		0.1	53.648	45.052	**4.55E-02**
0.2	69.932	64.712	9.95E-02		0.2	67.427	65.483	4.93E-01
0.5	79.999	81.909	1.02E-01		0.5	80.762	80.532	9.21E-01
1	88.910	90.348	1.85E-01		1	87.809	89.363	1.23E-01

Table 9: Comparison between MCNN and CNN, after training on 5% and 30% of the IP set



(a) Overall and average accuracy after training on 5% IP, compared to baseline



(b) Overall and average accuracy after training on 30% IP, compared to baseline

Figure 14: Comparison between baseline and models previously trained on 5% or 30% of IP

5.7.2 Base Versus Retrained

The comparison of both models, with and without prior training on the IP set, can be seen in Table 10. This shows whether the models individually gain from having been trained previously on the IP set.

Firstly we will discuss the CNN, after training on 5% of the IP set, the retrained CNN achieves significant improvements compared to the base case, for a UP training percentage of 0.1% and 0.2%, for both the OA and AA.
After having been trained on 30% of the IP set, the retrained CNN achieves increased performance for a training percentage of 0.2% with regard to the OA and the AA.

Secondly we will be discussing the results of the MCNN, here the OA is significantly improved for 1% training on the UP set, after training on 5% of the IP set. The AA does not achieve significant improvements for a training percentage of 5% on the IP set.
After training on 30% of the IP set, the OA and AA improve significantly when training on 0.2% of the UP set.

In conclusion, both models individually benefit from having been trained on 5% and 30% of the IP set. Overall the CNN gains more from having been pre-trained on the IP set.

Base vs trained on 5% IP, CNN					Base vs trained on 5% IP, MCNN			
tr_percent	Base CNN_OA	CNN_OA	p value		tr_percent	Base MCNN_OA	MCNN_OA	p value
0.1	67.951	72.926	**1.27E-02**		0.1	67.652	69.583	3.98E-01
0.2	75.002	79.663	**1.06E-03**		0.2	75.356	75.971	7.10E-01
0.5	87.099	87.060	9.61E-01		0.5	87.456	89.055	1.03E-01
1	92.136	93.010	1.09E-01		1	93.144	94.484	**1.22E-02**
tr_percent	Base CNN_AA	CNN_AA	p value		tr_percent	Base MCNN_AA	MCNN_AA	p value
0.1	48.569	61.198	**9.09E-04**		0.1	46.326	50.315	2.24E-01
0.2	59.305	69.932	**7.36E-03**		0.2	57.741	64.712	9.23E-02
0.5	78.946	79.999	6.27E-01		0.5	77.893	81.909	5.05E-02
1	87.902	88.910	2.38E-01		1	88.630	90.348	2.17E-01
Base vs trained on 30% IP, CNN					Base vs trained on 30% IP, MCNN			
tr_percent	Base CNN_OA	CNN_OA	p value		tr_percent	Base MCNN_OA	MCNN_OA	p value
0.1	67.951	68.928	4.90E-01		0.1	67.652	69.706	2.81E-01
0.2	75.002	78.101	**3.54E-03**		0.2	75.356	80.337	**1.21E-02**
0.5	87.099	87.141	9.28E-01		0.5	87.456	88.666	3.31E-01
1	92.136	92.222	8.28E-01		1	93.144	93.819	1.79E-01
tr_percent	Base CNN_AA	CNN_AA	p value		tr_percent	Base MCNN_AA	MCNN_AA	p value
0.1	48.569	53.648	1.32E-01		0.1	46.326	45.052	6.44E-01
0.2	59.305	67.427	**4.17E-02**		0.2	57.741	65.483	**4.37E-02**
0.5	78.946	80.762	3.93E-01		0.5	77.893	80.532	3.47E-01
1	87.902	87.809	8.48E-01		1	88.630	89.363	5.87E-01

Table 10: Base case compared to retrained model for CNN and MCNN. Trained on 5% or 30% IP

6 Discussion

While attempting to answer the research questions of this thesis, there were a couple difficulties. The first was gaining access to the code, and after achieving this, the next task was to make old deprecated Python code work on a Windows pc.
Throughout this project, a large concern was getting the Docker container set up with the proper libraries and Linux versions. It would have been preferable to have a pure MNN running to compare to a CNN. However due to time constraints, and issues with finding the right versions of Linux and libraries, we were unable to make it work. This is despite Dr. Nogueira kindly providing us with the code-base for "*An Introduction to Deep Morphological Networks*". For future reference, the codebase seems to require Linux 16.04, which we discovered after completing the setup with all libraries in Linux 18.04.

Due to these difficulties in getting to a setup where we were able to run the code, with the use of GPU acceleration, a lot of time was lost, which reduced the number of experiments we could run. Therefore we were able to achieve less significance, mainly for the experiments relating to the UP set, as it contains double the number of non-background pixels of the IP set. It would have been preferable to perform the experiments with a larger number of runs, from 5 to 10 at least, and from 200 epochs to 500. Additionally getting more measuring points with regard to the retraining experiment would have been meaningful, as we were unable to make many definite conclusions from only 2 different IP pre-training percentages.

Overall having more runs would have enabled us to make more meaningful conclusions, or find more exact differences between the two models. 50 runs would have been preferable for the purpose of significance, compared to the 10 we have chosen as a result of our time constraints. The CNN is able to run faster than the MCNN, and would have been able to achieve the 50 runs for most experiments, however for the purpose of a more equal comparison, we have kept the variables for both models equal.

7 Conclusion

Here we come back to the overarching research questions, **"Does an MCNN need less input data to achieve equal results to the CNN?"**, **"Is an MCNN able to produce better results than a CNN when training both on noisy or incomplete data?"** and **"Does an MCNN produce better results after previously being trained on another dataset, compared to its non-retrained self, and a retrained CNN?"**.

With regard to needing less input data, for both tested datasets, the MCNN is able to significantly outperform the CNN at most training percentages, and achieve accuracies that the CNN needs two or three times as much data to exceed. The more elaborate conclusions for these results have been mentioned in the results section.

In terms of producing better results than a CNN, after being trained on noisy data, in two of the three noise types, the MCNN performed worse in most cases. For Gaussian noise, with exception of the case where Gaussian deviation was set to 0.1, the CNN outperformed the MCNN significantly.
With 3D salt and pepper noise, where the noise is randomly distributed across all 3 dimensions (spectrally uncorrelated), the CNN achieved significantly higher accuracies, with up to 25% higher OA and 32% higher AA. Therefore from our tests, the CNN is significantly better at dealing with this type of noise in all cases.
Lastly we have 2D salt and pepper noise (spectrally correlated), where the noise is applied to the 2D patch, and then carried through to all bands. In this aspect the MCNN performs significantly better than the CNN, with up to 44% higher overall accuracy, and up to 46% higher average accuracy. This shows that the MCNN is able to better deal with this type of noise than the CNN. For the varying noise types, the results section contains more in depth conclusions.

Then we have the question whether an MCNN performs better when retrained, and better compared to a retrained CNN.
Compared to their non-retrained versions, both of the retrained models functioned slightly better in most cases, however the difference was often not significant.
Comparing the two models to each other, when very few pixels are involved, such as 0.1% of the dataset, the CNN is able to perform slightly better, however with limited significance. At the slightly higher training percentages, such as 0.5%, the MCNN can outperform the CNN slightly, but once again the significance is limited.

Lastly, we have added two residual maps in Appendix B, Showing where the models are making their mistakes. From these we can observe that there is quite little overlap, 34 or 20 pixels. We can also observe that the CNN performs less well in corners, whereas the MCNN makes more mistakes on some edges, and the occasional corner.

In conclusion, the MCNN is able to achieve the same performance, with significantly less training data in many cases compared to the CNN. The MCNN performs significantly worse than the CNN for most Gaussian noise deviations, and the less common 3D salt and pepper noise. It does perform significantly better when exposed to 2D salt and pepper noise, when compared to the CNN. For the purpose of retraining, the results we obtained were insignificant in most cases. Additionally, it is recommended to train the MCNN with 500 epochs, rather than 200, as it shows significant improvement, with no additional cost except time.

8 Acknowledgments

Firstly I would like to extend my gratitude to Dr. Michael H.F. Wilkinson, the supervisor for this bachelor thesis, for giving me the opportunity to work on this interesting project, and guiding me throughout.
I would also like to thank prof. Dr. Kerstin Bunte, the second supervisor for this project, for being willing to assess the final version of this thesis.
Lastly I would like to thank Dr. Nogueira for providing the code of "*An Introduction to Deep Morphological Networks*".

References

[1] D. Bhatt, C. Patel, H. Talsania, J. Patel, R. Vaghela, S. Pandya, K. Modi, and H. Ghayvat. CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope. *Electronics*, 10(20):Article 2470, OCT 2021.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, June 2014.

[3] M. Graña, M. Veganzons, and B. Ayerdi. Hyperspectral Remote Sensing Scenes. https://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes. Accessed: 28-06-2025.

[4] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):386–397, FEB 2020.

[5] M. Kuk. MorphConvHyperNet [Unofficial GitHub repository]. https://github.com/max-kuk/MorphConvHyperNet. Accessed: 28-06-2025.

[6] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

[7] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.

[8] S. Mallat. Understanding Deep Convolutional Networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374, 2016.

[9] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, Dec 1943.

[10] D. Mellouli, T. M. Hamdani, J. J. Sanchez-Medina, M. Ben Ayed, and A. M. Alimi. Morphological Convolutional Neural Network Architecture for Digit Recognition. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2876–2885, Sep. 2019.

[11] K. Nogueira, J. Chanussot, M. D. Mura, and J. A. D. Santos. An Introduction to Deep Morphological Networks. *IEEE Access*, 9:114308–114324, 2021.

[12] M. Paoletti, J. Haut, J. Plaza, and A. Plaza. Deep Learning Classifiers for Hyperspectral Imaging: A Review [GitHub repository]. https://github.com/mhaut/hyperspectral_deeplearning_review/tree/master. Accessed: 28-06-2025.

[13] M. Paoletti, J. Haut, J. Plaza, and A. Plaza. Deep Learning Classifiers for Hyperspectral Imaging: A Review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 158:279–317, 2019.

[14] A. Prieto, B. Prieto, E. Martinez Ortigosa, E. Ros, F. Pelayo, J. Ortega, and I. Rojas. Neural Networks: An Overview of Early Research, Current Frameworks and New Challenges. *Neurocomputing*, 214:242–268, NOV 2016.

[15] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(06):1137–1149, June 2017.

[16] G. Ritter and P. Sussner. An introduction to morphological neural networks. 4:709–717 vol.4, 1996.

[17] S. K. Roy, R. Mondal, M. E. Paoletti, J. M. Haut, and A. Plaza. Morphological Convolutional Neural Networks for Hyperspectral Image Classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:8689–8702, 2021.

[18] M. M. Taye. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. *Computation*, 11(3):Article 52, 2023.

A Link to Github Repository

https://github.com/Reinier-H/MCNN_CNN_Comparison

B Residual Map

The following two images show whether both models make the same errors, or could be used to complement each other, or are better suited in different use cases.

- blue, errors by the CNN
- red, errors from the MCNN
- green, errors by both models
- white, both models correct
- black, background

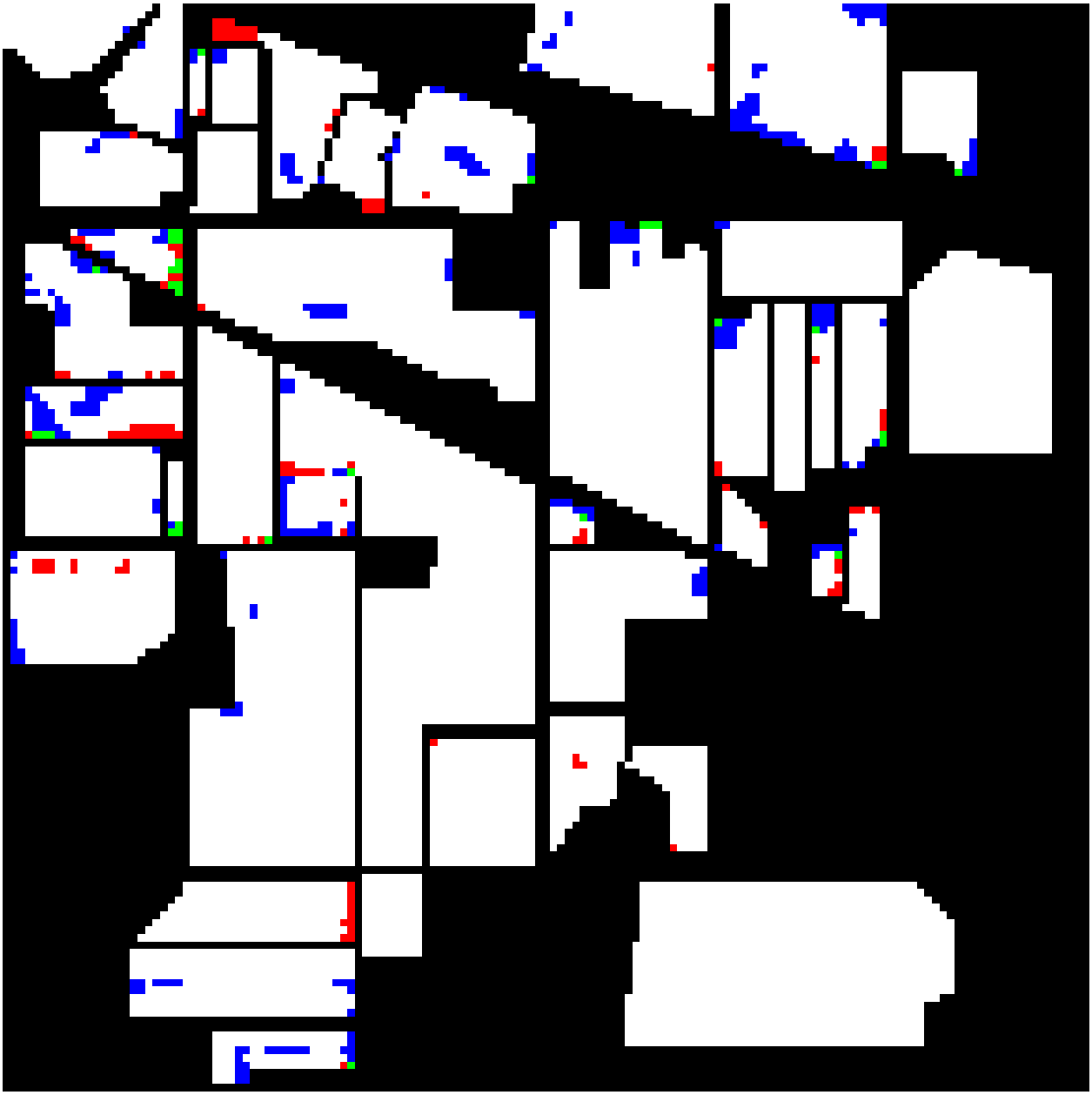


Figure 15: Residual map, with 34 overlapping pixels
MCNN: OA 98.18, AA 94.83
CNN: OA 95.76, AA 93.10
training on 15% IP, 500 epochs

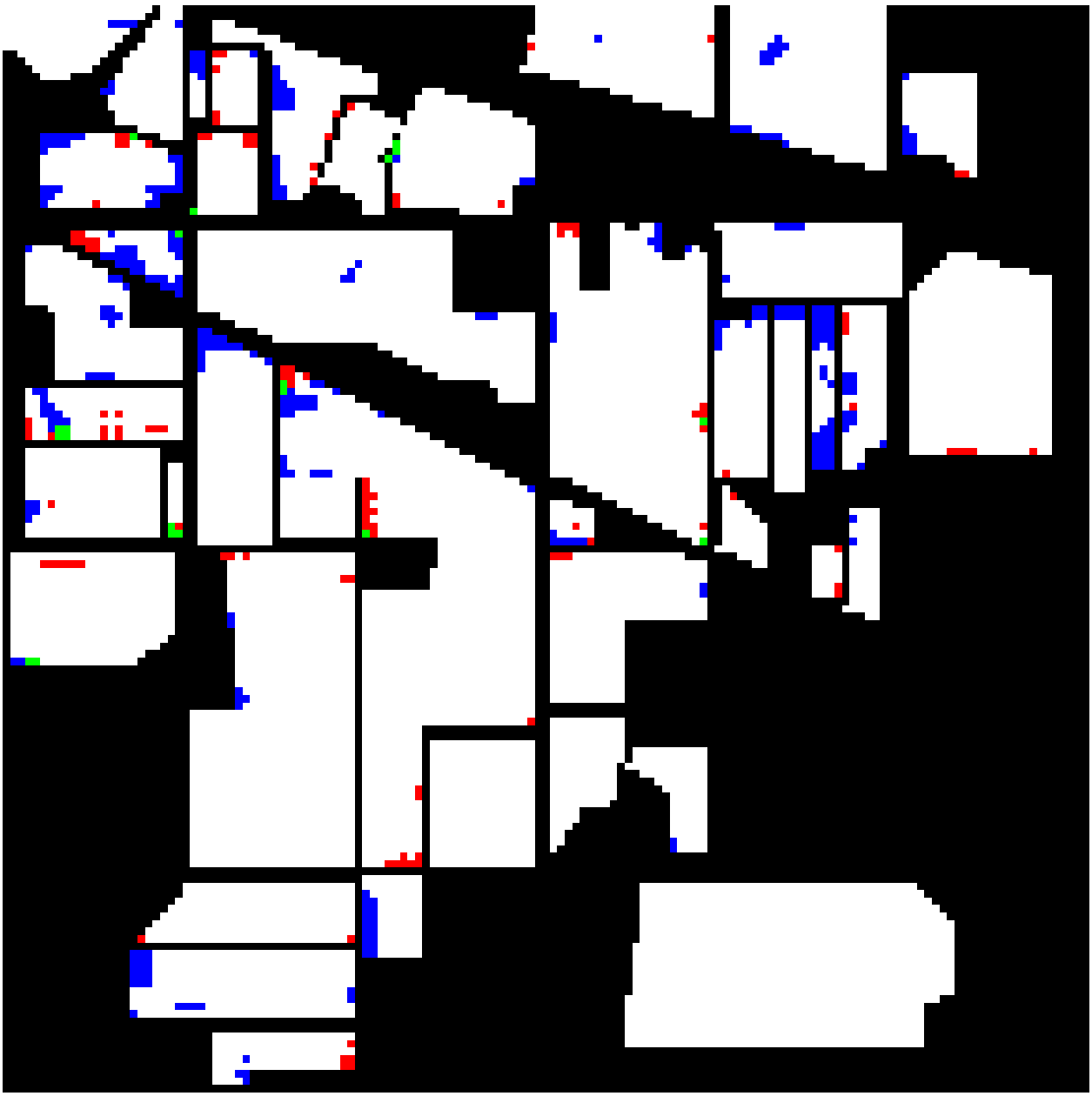


Figure 16: Residual map, with 20 overlapping pixels
MCNN: OA 98.30, AA 96.8
CNN: OA 95.68, AA 95.46
training on 15% IP, 500 epochs