Agrivision Project

Ron Benchetrit & Mor Lavon
Department of Computer Science,
Technion - Israel Institute of Technology

Supervisors

Alon Zvirin & Yaron Honen
Abstract

We extend the MSCG-Net with agricultural indices and evaluate the results. We were able to improve the model results with the addition of only a few parameters. We later on find that maybe the improvement stems from the extra parameters of the network and not the indices transforms themself.

preliminaries

ResNets

ResNets are a very popular family of CNNs utilizing residual connections allowing the network stable learning even with a big amount of layers.

ResNets are widely used today in transfer learning by taking the (pretrained) feature extractor (or part of it) and adding a new head. This saves a lot of time in training because the first few layers of the ResNet are trained to extract general features not tied to a specific Domain. ResNet has several depth architectures depicted in the table below:

ResNet architectures table

<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>1x1</td>
<td>112x112</td>
<td>7x7</td>
<td>7x7</td>
<td>7x7</td>
<td>7x7</td>
</tr>
<tr>
<td>conv2,4</td>
<td>56x56</td>
<td>3x3, 64</td>
<td>3x3, 64</td>
<td>1x1, 64</td>
<td>1x1, 64</td>
<td>1x1, 64</td>
</tr>
<tr>
<td>conv3,4</td>
<td>28x28</td>
<td>3x3, 128</td>
<td>3x3, 128</td>
<td>3x3, 128</td>
<td>3x3, 128</td>
<td>3x3, 128</td>
</tr>
<tr>
<td>conv4,5</td>
<td>14x14</td>
<td>3x3, 256</td>
<td>3x3, 256</td>
<td>1x1, 256</td>
<td>1x1, 256</td>
<td>1x1, 256</td>
</tr>
<tr>
<td>conv5,6</td>
<td>7x7</td>
<td>3x3, 512</td>
<td>3x3, 512</td>
<td>3x3, 512</td>
<td>3x3, 512</td>
<td>3x3, 512</td>
</tr>
<tr>
<td>average</td>
<td>1x1</td>
<td>3x3 main post, artic 2</td>
<td>1x1, 64</td>
<td>1x1, 64</td>
<td>1x1, 64</td>
<td>1x1, 64</td>
</tr>
<tr>
<td>FLOPs</td>
<td>1.1x10^7</td>
<td>3.6x10^7</td>
<td>3.8x10^7</td>
<td>7.6x10^7</td>
<td>1.3x10^7</td>
<td>1.3x10^7</td>
</tr>
</tbody>
</table>


For the MSCG-Net both the 50-layer and the 101-layer are used.

GCNs

Graph Convolutional Networks (GCNs) are neural networks designed to operate on and extract information from graphs. G = (A, x), A ∈ R^{n×n}, X ∈ R^{n×d} are the adjacency and feature matrices. At each layer, the GCN aggregates information in one-hop neighborhoods,
SCG

The Self-Constructing Graph (SCG) module allows the construction of undirected graphs, capturing relations across the image, directly from feature maps, instead of relying on prior knowledge graphs. It has less trainable parameters, outperforming much larger models. A feature map \( X \in \mathbb{R}^{h \times w \times d} \) consisting of high-level features (produced by a CNN) is converted to a graph \( G = (\hat{A}, X') \). \( X' \in \mathbb{R}^{n \times d} \) are the nodes where \( n = h' \times w' \leq h \times w \) denotes the number of nodes and \( \hat{A} \in \mathbb{R}^{n \times n} \) is the learned weighted adjacency matrix. Parameter-free pooling operations, in the paper’s case adaptive average pooling, are employed to reduce the spatial dimensions of \( X \) to \( h' \) by \( w' \) followed by a reshape to \( X' \).

The module learns a mean matrix \( \mu \in \mathbb{R}^{n \times c} \) and a standard deviation matrix \( \sigma \in \mathbb{R}^{n \times c} \) of a gaussian using two single-layer convolutional networks. The latent embedding is produced by \( Z = \mu + \sigma \epsilon, \epsilon \in \mathbb{R}^{n \times c} \) is an auxiliary noise variable sampled from a normal distribution. Based on the learned embeddings, \( A' = \text{ReLU}(ZZ^T) \). After applying a diagonal regularization to stabilize training and preserve local information to \( A' \) we get \( A^* \). The symmetric normalized \( \hat{A} \) that SCG produces and that will be the input to later graph operations is computed as \( \hat{A} = D^{-1/2}(A^* + I)D^{1/2} \).

The SCG further produces an adaptive residual prediction \( \hat{y} = \gamma \mu(1 - \log \sigma) \), which is used to refine the final prediction of the network after information has been propagated along the graph.

Introduction

Agriculture-Vision dataset

The success of deep learning in computer vision in recent years has drawn a lot of attention, particularly in agriculture applications. However, little progress has been made to merge computer vision and agriculture science due to lack of suitable dataset. Moreover, problems in agriculture impose new challenges for computer vision. For example, semantic segmentation of aerial farmland images requires inference over extremely large-size images with extreme annotation sparsity.
To encourage research in computer vision for agriculture, UIUC, Intelinair and the university of Oregon present **Agriculture-Vision**, a large-scale aerial farmland image dataset for semantic segmentation of agricultural patterns. The 2020 dataset was created from around 95,000 high-quality aerial images from 3,432 farmlands across the US, where each image consists of RGB and Near-infrared (NIR) channels with resolution as high as 10 cm per pixel.

The dataset has nine types of annotations: double plant, drydown, endrow, nutrient deficiency, planter skip, storm damage, water, waterway and weed cluster. All of these patterns have substantial impacts on field conditions and the final yield. All annotations were reviewed by expert agronomists.

To allow for visualizing each field image and preparing for later experiments, the four channels are separated into a regular RGB image and an additional single-channel NIR image, and stored as two JPG images. Unprocessed farmland images have extremely large image sizes. This poses significant challenges to deep network training in terms of computation time and memory consumption. Moreover, some annotations are very sparse. This means training a segmentation model on the entire image for these patterns would be very inefficient, and would very possibly yield suboptimal results.

On the other hand, unlike common objects in other segmentation tasks, visual appearances of anomaly patterns in aerial farmland images are preserved under image subsampling methods such as flipping and cropping. This is because these patterns represent regions of the anomalies instead of individual objects. As a result, it is possible to sample image patches from these large farmland images by cropping around annotated regions in the image. This simultaneously improves data efficiency, since the proportion of annotated pixels is increased.

For these reasons the dataset is constructed by cropping annotations with a window size of 512 × 512 pixels. This image demonstrates the subsampling process. on the left is a field image with a size of 10875 × 3303 pixels. on the right are 3 512 × 512 pixels, produced by subsampling.

**Example of the subsampling process:**
The dataset is splitted in a 6/2/2 train/val/test ratio such that no cropped images from the same farmland will appear in multiple splits in the final dataset. The generated Agriculture-Vision 2020 dataset thus contains 56,944/18,334/19,708 train/val/test images.

Agriculture-Vision tackles some interesting tasks such as images beyond RGB, it was demonstrated that aerial agricultural semantic segmentation is more effective using NRGB images rather than just RGB images. This also induce an uncommon type of transfer learning where a model pretrained on RGB images of common objects is transferred to multi-spectral agricultural images. Moreover, Agriculture-Vision can provide the option to explore visual recognition tasks on extremely large images.
The 1st Agriculture-Vision Challenge

To encourage research in agricultural semantic segmentation the first Agriculture-Vision Challenge was held. A subset of the original dataset with 21,061 images was used. The dataset was split into 12901/4431/3729 train/val/test images respectively.

The labels in this dataset are not mutually exclusive, which means that a pixel can contain more than one pattern. As a result, a custom metric is designed to evaluate submissions.

The included labels are: background, cloud shadow, double plant, planter skip, standing water, waterway and weed cluster).

To accommodate for overlapping labels, the conventional mean Intersection-over-Union (mIoU) metric was modified by categorizing predictions of any label in a pixel as a correct prediction. This enables easy adaptation of typical semantic segmentation models into our agriculture challenge. Specifically, to compute the modified mIoU, a confusion matrix $M^{(c \times c)}$ ($c = 7$ is the number of classes plus background) is first computed with the following rules:

For each prediction $x$ and label set $Y$ at a pixel:

1. If $x \subseteq Y$, then $M_{y,y} = M_{y,y} + 1 \ \forall y \in Y$
2. Otherwise, $M_{x,y} = M_{x,y} + 1 \ \forall y \in Y$

Finally, the modified mIoU is computed by:

$$\frac{1}{c} \sum_c \frac{\text{True positive}_c}{\text{Prediction}_c + \text{Target}_c - \text{True positive}_c}$$

The modified mIoU increases the reward for a correct prediction by allowing any correct prediction to count as a true positive for all ground truth labels. It also heavily penalizes the model if the prediction does not match any of the ground truth labels.

Submissions for the challenge were evaluated on the challenge test set with 3729 and ranked based on the modified mIoU.

Some notable submissions stand out with their methodologies. We chose to focus on the Team SCG Vision submission, the Multi-view Self-Constructing Graph Convolutional Network.
Multi-view Self-Constructing Graph Convolutional Networks with Adaptive Class Weighting Loss

Currently, the end-to-end semantic segmentation models are mostly inspired by the idea of fully convolutional networks. To achieve higher performance, CNN-based end-to-end methods normally rely on deep and wide CNN architectures to create a large receptive field in order to obtain strong local patterns, but also capture long range dependencies between objects of the scene. However, this approach for modeling global context relationships is highly inefficient and typically requires a large number of trainable parameters, considerable computational resources, and large labeled training datasets.

Recently, graph neural networks and Graph Convolutional Networks have received increasing attention and have been applied to many tasks in computer vision. However, these approaches are quite sensitive to how the graph of relations between objects is built and previous approaches commonly rely on manually built graphs based on prior knowledge.

In order to address this problem and learn a latent graph structure directly from 2D feature maps for semantic segmentation, the Self-Constructing Graph module (SCG) was recently proposed and has obtained promising results.

The multi-view self-constructing graph convolutional network (MSCG-Net) extends the SCG to explicitly exploit the rotation invariance in airborne images by augmenting the input features to obtain multiple rotated views and then fuses the multi-view global contextual information before projecting the features back onto the 2-D spatial domain.

Incorporating multiple views has significantly improved the iMoU score by around 5%.

The MSCG-Net also utilizes a novel adaptive class weighting loss (ACW loss) that addresses the issue of class imbalance commonly found in semantic segmentation datasets, particularly in the Agriculture-Vision challenge dataset (e.g. most pixels in the images belong to the background class and only few belong to classes such as planter skip and standing water).

To address this problem existing methods make use of weighted loss functions with pre-computed class weights based on the pixel frequency of the entire training data to scale the loss for each class-pixel according to the fixed weight before computing gradients.

Instead of computing the fixed weights over the whole dataset the ACW loss is based on iterative batch-wise class rectification.

Incorporating the ACW loss had an improvement of around 1.5% (mIoU) compared to Dice.

The MSCG-Net architecture is composed of a backbone CNN network (first three bottleneck layers of a pretrained resnet) to extract the high level features \(X\), an augmentation module that expands the feature map \(X\) to multiple views \((X_{90}, X_{180})\), the SCG and the GCN \((K=2)\).
It is important to note that all the multiple views pass through the same SCG and GCN. The fusion layer merges all the predictions together by reversed rotations and element-wise additions and finally the fused outputs are projected and up-sampled back to the original size.

**The MSCG-Net high level architecture:**

![MSCG-Net architecture diagram](image)


By incorporating learnable latent variables to self construct the underlying graphs, and to explicitly capture multi-view global context representations with rotation invariance in airborne images and employing an adaptive class weighting loss the MSCG-Net achieves robust and competitive results while using less parameters and being more computationally efficient.

**Our Contribution:**

For the purpose of our research there were several changes needed in the original source code of the paper. They will be more broadly explained in the implementation section.

in a high level view:

- Added configuration abilities for using agricultural indices.
- Added user ability to input which indices are concatenated.
- Added the ability to have a channel with learned indices, and initialize it to the original indices of an index of your choosing
- changed the architecture of the original model to include a higher dimension input.
- Added support for “weights and biases” logging of experiments.

**Implementation**

**Onboarding**
We first cloned the MSCG-net environment from their [github page](https://github.com) and created an Anaconda environment to run the network on the faculty GPUs server.

We added an infrastructure for logging the runs using Weights and Biases (wandb). Through wandb we are able to track the training and inference online and log the results and the different parameters used.

We added a parser to allow the user to specify the run configuration. Whether logging to wandb, the run name and the index transforms configurations.

## index color transforms

Geo-color indices such as the Normalized-Difference-Vegetation-Index (NDVI) and Excess-Green-Index (ExG) are widely used in agricultural studies to extract meaningful information through images.

These indices have high correlation with land information such as water and plantations. **Examples of indices used:**

![RGB Image, NIR Image, NDVI Image](https://crops.extension.iastate.edu/cropnews/2016/05/choosing-right-imagery-best-management-practices-color-nir-and-ndvi-imagery)

This image demonstrates the NDVI color transform, making healthy crops stand out from the background.

We had implemented numerous color transforms, allowing us to augment the inputed NRGB images with additional transformed channels before feeding them to the backbone CNN. We also allow for parametrized color transforms which are learned jointly with the network parameters.

We will note that most of the indices are not linear (because of the denominator) and therefore cannot be simply learned implicitly in the first convolutional filters.

In general learning a nonlinear function is possible in Neural networks thanks to the non linear activation functions but it might be hard.
Here is a list of the supported indices and their formulas:

<table>
<thead>
<tr>
<th>n.o</th>
<th>Index name</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Normalized Difference Vegetation Index</td>
<td>$NDVI = \frac{NIR - R}{NIR + R}$</td>
</tr>
<tr>
<td>2.</td>
<td>green Normalized Difference Vegetation Index</td>
<td>$gNDVI = \frac{NIR - G}{NIR + G}$</td>
</tr>
<tr>
<td>3.</td>
<td>Soil Adjusted Vegetation Index</td>
<td>$SAVI = \frac{NIR - R}{(NIR + R + L)} \times (1 + L)$</td>
</tr>
<tr>
<td>4.</td>
<td>Ratio Vegetation Index</td>
<td>$RVI = \frac{R}{NIR}$</td>
</tr>
<tr>
<td>5.</td>
<td>Difference Vegetation Index</td>
<td>$DVI = NIR - R$</td>
</tr>
<tr>
<td>6.</td>
<td>Visible Difference Vegetation Index</td>
<td>$VDVI = \frac{2G - R - R}{2G + R + B}$</td>
</tr>
<tr>
<td>7.</td>
<td>Green Chromatic Coordinate</td>
<td>$GCC = \frac{G}{R + G + B}$</td>
</tr>
<tr>
<td>8.</td>
<td>Enhanced Vegetation Index</td>
<td>$EVI = \frac{NIR - R}{(NIR + 6R - 7.5B + 1)} \times 2.5$</td>
</tr>
<tr>
<td>9.</td>
<td>Visible Atmospherically Resistant Index</td>
<td>$VARI = \frac{G - R}{G + R - B}$</td>
</tr>
</tbody>
</table>

The most significant one is the change in infrastructure to provide support for additional channels that contain the agricultural indices at the input of the model. The original implementation idea was done by generating the agricultural indices through the data loader, since it required changes in the original data loader, and (more importantly) it prevented the ability of a proper learning through backpropagation on each channel separately. This is due to the fact that the indices formula used are not a part of the forward pass of the network, but a part of the data loading process therefore a backward pass that would change the weights in those channels is impossible.

Our chosen implementation in the end was for each index added - add another layer to the model that expands the out channels to be the in-channels+1. and initializes weights for those channels. In that way a learning process through back propagation was possible.

All the added layers are banded together through the nn module “AppendGenericAgriculturalIndices” (implemented in model.py). This module’s purpose is to concatenate the needed indices to the original 4 channels before feeding them as input to the CNN.
In addition it normalizes the added channels with min max normalization to [0,1] range to avoid enlarged output since in some indices you can divide by very small numbers. It has the flexibility to integrate any number of channels/ any chosen channels given to it as input.

Experiments documentation - weights and biases

A browser window to our “weights and biases” project for illustration:

Our tool to measure our loss and accuracy was the “weights and biases” library. For every run, we had real time results of the training loss, validation loss, learning rate and more. We needed to add the initialization and logging to the original source code, and added a configuration parameter that allows you to choose to track or not to track the results and add the run name to be recognized when checking the results. such as “run resnet-50 with ndvi channel” for example.
The track with weights and biases allows you to track in real time runs and compare them to previous ones, so you can also “kill” unsuccessful experiments midway through when noticing there won't be an improvement later on, and in that way save time.

A more detailed explanation of the use of the library is given in our project's README file. It does require a quick registration to their website and a login before generating a new run plot.

configuration and control over the model

In order to configure the experiments differently and automatically through scripts, we needed to add infrastructure to the original source code. the way we chose the configuration to happen was through input args that were passed to the run through the command line. those arguments are:

--wandb : a boolean flag that dictates weather use weights and biases or not
--NDVI - add NDVI channel.
--gNDVI- add gNDVI channel.
--SAVI- add SAVI channel.
--RVI- add RVI channel.
--DVI- add DVI channel.
--VDVI- add VDVI channel.
--GCC- add GCC channel.
--EVI- add EVI channel.
--VARI- add VARI channel.
--GAI : add a especially modified channel followed by a list of coefficients
--learn : add a learned channel that learns the coefficients.
--run_name : a run name for the weights and biases plots.

Experiments

Baseline Experiment

For our baseline experiment we firstly tried to replicate the results specified in the original MSCG-net paper. Our benchmark throughout the project is the mean-iou metric. We hoped to get an exact replica of the results, but our results came up short to the original paper. While only using resnet-50 the paper’s results were a mean-iu of 0.547 and our attempt gave us 0.507.

The following chart shows the mean-iu per epoch of training:

For resnet-101 the we also fell a little short of replicating the results with mIoU of 0.5442 compared to 0.55

Baseline resnet-101 validation score per epoch:(miou)
Not surprisingly the MSCG-Net with a backbone of resnet-101 performed slightly better than the MSCG-Net with a backbone of resnet-50 with a cost of much more parameters (30.99 million vs 9.59 million) and almost double the inference time.

Single Channel Addition Experiment

The next step in our research was adding new channels to the existing models and training them in hopes of improving the results. The channels we originally added were NDVI, gNDVI, SAVI, RVI, DVI, VDVI, GCC, EVI and VARI with their formulas available in the index color transforms sub-section.

All models were trained with 30 epochs and the hyperparameters specified in the paper.

The result of our runs are as follows:
For clarity we decided to display only the baseline and the best 5 index transforms. On the tables we highlight the best models in terms of mIoU.

**Resnet-50 single channel addition compared to baseline:**
<table>
<thead>
<tr>
<th>Model</th>
<th>Miou</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-50 baseline</td>
<td>0.507</td>
</tr>
<tr>
<td>Resnet-50 + NDVI</td>
<td>0.5142</td>
</tr>
<tr>
<td>Resnet-50 + gNDVI</td>
<td>0.491</td>
</tr>
<tr>
<td><strong>Resnet-50 + SAVI</strong></td>
<td><strong>0.5236</strong></td>
</tr>
<tr>
<td>Resnet-50 + RVI</td>
<td>0.27</td>
</tr>
<tr>
<td>Resnet-50 + DVI</td>
<td>0.5168</td>
</tr>
<tr>
<td>Resnet-50 + VDVI</td>
<td>0.519</td>
</tr>
<tr>
<td>Resnet-50 + GCC</td>
<td>0.5149</td>
</tr>
<tr>
<td>Resnet-50 + EVI</td>
<td>0.42</td>
</tr>
<tr>
<td>Resnet-50 + VARI</td>
<td>0.32</td>
</tr>
</tbody>
</table>

We got an improvement for most indices. with the best being **SAVI** with an improvement of over 1.6% in mIoU.

Resnet-101 single channel addition compared to baseline:
### Model

<table>
<thead>
<tr>
<th>Model</th>
<th>Miou</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-101 baseline</td>
<td>0.5442</td>
</tr>
<tr>
<td>Resnet-101 + NDVI</td>
<td>0.5376</td>
</tr>
<tr>
<td>Resnet-101 + gNDVI</td>
<td>0.5366</td>
</tr>
<tr>
<td>Resnet-101 + SAVI</td>
<td>0.523</td>
</tr>
<tr>
<td>Resnet-101 + RVI</td>
<td>0.5421</td>
</tr>
<tr>
<td>Resnet-101 + DVI</td>
<td>0.5404</td>
</tr>
<tr>
<td>Resnet-101 + VDVI</td>
<td>0.5291</td>
</tr>
<tr>
<td>Resnet-101 + GCC</td>
<td>0.5321</td>
</tr>
<tr>
<td><strong>Resnet-101 + EVI</strong></td>
<td><strong>0.5446</strong></td>
</tr>
<tr>
<td>Resnet-101 + VARI</td>
<td>0.5267</td>
</tr>
</tbody>
</table>

Unlike the resnet-50, We barely got an improvement for the additional channels. We assume this is because the resnet-101 can learn more complex patterns. Hence, the index transform might be redundant and doesn't affect the training as much as the resnet-50.

### “Dummy” Channel Addition Experiment

we wanted to see how meaningful the color indices are. We experiment with this by adding a “dummy” channel (Red channel) and see whether the indices are actually helpful or is it just the extra filter of the first convolutional layer that improves the results.

**Resnet-50 dummy channel addition compared to baseline:**
Surprisingly, duplicating the red channel has actually improved the mIoU validation score compared to the baseline model with the model with the duplicated red channel topping at 52.15% compared to the baseline topping at 50.7%.

Since the difference is more than a percent we assume that it is more than just stochasticity in the training.

This result is surprising because theoretically, we can just sum the filters learned for the first and the second red channels and apply it to the original red channel.

**Multiple Channel Addition Experiment**

Our next step was examining if a combination of channels can improve the results of only adding a single channel.
Multiple channels addition for resnet-50:

<table>
<thead>
<tr>
<th>Model</th>
<th>Miou</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-50 + NDVI + gNDVI + SAVI</td>
<td>0.531</td>
</tr>
<tr>
<td>Resnet-50 + NDVI + SAVI</td>
<td>0.5234</td>
</tr>
<tr>
<td>Resnet-50 + EVI + SAVI</td>
<td>0.411</td>
</tr>
<tr>
<td>Resnet-50 + GCC + SAVI</td>
<td>0.5137</td>
</tr>
</tbody>
</table>

We can see that the combination of 3 channels yielded the best results allowing the model to learn more complex patterns.

Learned Channel Experiment

After exploring the static channels addition, we created an option for an additional learned channel that can be initialized by one of the static channels as its coefficients, and had the ability to change those coefficients according to the loss gradient.
Learned channel addition for resnet-50:

<table>
<thead>
<tr>
<th>Model</th>
<th>static (mIoU)</th>
<th>learned (mIoU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-50 + NDVI</td>
<td>0.5142</td>
<td>0.5095</td>
</tr>
<tr>
<td>Resnet-50 + gNDVI</td>
<td>0.491</td>
<td>0.5021</td>
</tr>
<tr>
<td>Resnet-50 + SAVI</td>
<td>0.5236</td>
<td>0.5195</td>
</tr>
<tr>
<td>Resnet-50 + RVI</td>
<td>0.27</td>
<td>0.471</td>
</tr>
<tr>
<td>Resnet-50 + DVI</td>
<td>0.5168</td>
<td>0.5288</td>
</tr>
<tr>
<td>Resnet-50 + VDVI</td>
<td>0.519</td>
<td>0.406</td>
</tr>
<tr>
<td>Resnet-50 + GCC</td>
<td>0.5149</td>
<td>0.5122</td>
</tr>
<tr>
<td>Resnet-50 + EVI</td>
<td>0.42</td>
<td>0.44</td>
</tr>
<tr>
<td>Resnet-50 + VARI</td>
<td>0.32</td>
<td>0.32</td>
</tr>
</tbody>
</table>

From the results we conclude that there is not much of an improvement. We assume this learning scheme is not effective.

Batch Normalization addition Experiment

Batch Normalization (BN) is widely used in Deep Learning these days to make learning more stable. We wanted to explore how this affects learning when using a static channel (most indices are not a linear function) and when using a learned channel. We used BN only for the augmented indices channels.

For the resnet-50 we evaluated the SAVI index, both with batch normalization and without.
The model without the Batch normalization performed better than the one with topping at 52.36% compared to 51.72%.

**Batch Normalization for learned channel addition to resnet-50:**

As for the learned channels, BN might slightly address overfitting but there is no notable difference in the performance.

Overall, BN did not yield any significant improvement.

**Conclusions**

Based on the experiments. We conclude that adding extra channels is useful for shallow networks but not as much for deep networks. Moreover, the red “dummy” channel has also improved the results.
We hoped that the learning of the index coefficient would yield a new meaningful index but in reality the learned channel did not improve the results and we did not experience a large deviation from the initialization.

Another observation is that there was no correlation between useful indices for resnet-50 and resnet-101. For example, We obtained a significant improvement for SAVI in the resnet-50 but not in resnet-101.

Based on the above observations we assume that the improvement stems from increasing the models complexity and not from the indices themself.
Further work and research

- Since duplicating the red “dummy” channel has improved the results for resnet-50 it might be interesting to try duplicating channels as a training tool. Since it only directly affects the first convolution and because convolution is a linear operator we can compress the network back to its original size by summing the learned filters for each duplicated channel. i.e. \( W'_{R} = W_{R_1} + W_{R_2} + \ldots + W_{R_n} \). This can be performed as a post-training process.

- We encountered that our network is struggling with a specific class (planter skip). An Expert network trained only to detect that class could be helpful by using ensemble methods.

Confusion matrix of our baseline resnet-101:

![Confusion matrix](image)

The planter skip label is significantly lower than the other labels.

- Some ideas to improve the current model:
  - changing the learning rate on the added learned channel.
  - changing the loss function
  - dataloader changes: transforms to the images, mixing train and validation etc.
  - hyper parameter tuning: batch size, learning rate smaller image resolution etc.
  - pre train on a smaller reliable set before full training.
References

1. The Geometrical Image Processing Lab (GIP)
7. The original MSCG implementation: https://github.com/samleogh/MSCG-Net