7-2022

A Deep Learning Image Segmentation Model for Agricultural Irrigation System Classification

Ehsan Raei
Shiraz University

Ata Akbari Asanjan
Universities Space Research Association (USRA) at NASA Ames Research Center

Mohammad Reza Nikoo
Sultan Qaboos University

Mojtaba Sadegh
Boise State University

Shokoufeh Pourshahabi
Shiraz University

See next page for additional authors

Publication Information

Raei, Ehsan; Asanjan, Ata Akbari; Nikoo, Mohammad Reza; Sadegh, Mojtaba; Pourshahabi, Shokoufeh; and Adamowski, Jan Franklin. (2022). "A Deep Learning Image Segmentation Model for Agricultural Irrigation System Classification". *Computers and Electronics in Agriculture, 198*, 106977. https://doi.org/10.1016/j.compag.2022.106977

A Deep Learning Image Segmentation Model for Agricultural Irrigation System Classification

Ehsan Raei
Research Assistant
Department of Civil and Environmental Engineering
Shiraz University
Shiraz, Iran

and

PhD Student
Department of Bioresource Engineering
McGill University
Saint-Anne-de-Bellevue, QC, Canada
ehsan.raeiezabadi@mail.mcgill.ca

Ata Akbari Asanjan
Research Scientist
Universities Space Research Association (USRA)
NASA Ames Research Center
Mountain View, CA, USA
ata.akbariasanjan@nasa.gov

Mohammad Reza Nikoo
Associate Professor
Department of Civil and Architectural Engineering
Sultan Qaboos University
Muscat, Oman
m.reza@squ.edu.om

Mojtaba Sadegh
Assistant Professor
Department of Civil Engineering
Boise State University
Boise, ID, USA
mojtabasadegh@boisestate.edu

Shokoufeh Pourshahabi
Researcher
Department of Civil and Environmental Engineering
Shiraz University
Shiraz, Iran
pshahabi@shirazu.ac.ir

Jan Franklin Adamowski
Department of Bioresource Engineering
McGill University
Sainte-Anne-de-Bellevue, QC, Canada
jan.adamowski@mcgill.ca

Abstract

Effective water management requires a large-scale understanding of agricultural irrigation systems and how they shift in response to various stressors. Here, we leveraged advances in Machine Learning and availability of very high resolution remote sensing imagery to help resolve this long-standing issue. To this end, we developed a deep learning model to classify irrigation systems at a regional scale using remote sensing imagery. After testing different model architectures, hyper parameters, class weights and image sizes, we selected a U-Net architecture with a Resnet-34 backbone for this purpose. We applied transfer learning to increase training efficiency and model performance. We considered *four irrigation systems* as well as urban and background areas as land use/cover classes; and applied the model to 8,600 very high resolution (1 meter) images, labeled with ground-truth observations of irrigation types, in a case study from Idaho, USA. Images were obtained from the US Department of Agriculture's National Agriculture Imagery Program. Our model achieved state-of-the-art performance for segmentation of different classes on the train data (85% to 94%), validation data (72% to 86%), and test data (70% to 86%), which attests to the efficacy of the model for the segmentation of images based on spatial features. Aside from leveraging deep learning and remote sensing for resolving the standing real-world problem of multiple irrigation type segmentation, this study develops and publicly shares labeled data, as well as a trained deep learning model, for irrigation type segmentation that can be applied/transferred to other regions globally. Furthermore, this study offers novel information about the impacts of transfer learning, imbalanced training data, and efficacy of various model structures for multiple irrigation type segmentation.

Keywords: irrigation systems, remote sensing, deep learning, U-Net architecture, image segmentation

Highlights:

A deep learning model was developed for irrigation type segmentation

High resolution imagery can inform irrigation system classification

Provided labeled training data and model can be transferred to other regions

Irrigation system shifts can be monitored regionally and globally

1. Introduction

Food security is closely tied to the efficiency of freshwater consumption by the agricultural sector in a warming climate with an ever-increasing dependence on irrigation (Crist et al., 2017; Sadegh et al. 2020). Irrigation systems modulate agricultural productivity (Tang et al., 2020a), and are associated with significant implications for the soil, water resources, sustainable development and the environment (Zhang et al., 2018; Torabi Haghighi et al. 2020; de Albuquerque et al., 2021b). Efficient irrigation systems lead to a substantial decrease in labor and water needs in comparison with traditional surface irrigation methods (Zhang et al., 2018). Hence, mapping farmlands with different irrigation systems is essential for understanding regional water needs, agricultural production, water resources consumption and vulnerability to climatic extremes (Tang et al., 2020c; Zhou et al. 2021; Loddo et al. 2021). Detailed information on the distribution of irrigation systems and how they shift in response to socio-environmental stressors are also essential for water and food planning, water resources allocation, and land use/cover change investigations (Zhang et al., 2018; Perea et al. 2021; Rad et al. 2022).

Most farmlands are remote and are characterized by different shapes, area, crop types, spatial features, and irrigation system types, making it infeasible to monitor farmlands at regional to global scales using ground observation due to excessive costs, especially given the long-term transient nature of farming characteristics (Islam et al., 2017). Remote sensing allows for cost- and time-effective land use monitoring at local to global scales (Kamilaris and Prenafeta-Boldú, 2018), especially in the face of advances in cloud computing and geospatial analysis that have resolved challenges of large data and computational limits (Gorelick et al., 2017; Rad et al. 2021a,b; Alizadeh et al. 2021).

Availability of very high resolution remote sensing imagery, such as the United States Department of Agriculture's (USDA) National Agriculture Imagery Program (NAIP) dataset, as well as the constellation of commercial very high resolution satellites, offer a great opportunity to efficiently monitor irrigation systems at large scales. NAIP imagery is free to the public and offers very high resolution (1 meter, although different in various years) imagery with optical (Red, Green, and Blue) and infrared bands, which is acquired during growing seasons by aircraft (Powell, 2009). Various studies have leveraged this remotely sensed data for investigations of cropland and urban areas, such as Patel et al. (2015), Johansen et al. (2015), Dong and et al. (2016), Liu et al. (2018), Tang et al. (2021b), and Teluguntla et al. (2018). In recent years, advances in deep learning algorithms have also presented unprecedented opportunities to extract geospatial features more efficiently, precisely and effectively than ever from remote sensing images. Deep learning has been applied in different fields, including, for example, land cover/use classification (Scott et al., 2017; Helber et al., 2018; Cheng et al., 2018a; Zhou et al., 2019; Cerron et al., 2020; and Tong et al., 2020), soil moisture content estimation (Song et al., 2016; Lee et al. 2019), plant disease detection (Ferentinos, 2018; Saleem et al., 2020; Tassis et al. 2021), crop type classification (Kussul et al., 2017; Wang et al., 2020; Hasan et al. 2021; Li et al., 2020; Zhou et al., 2019; Teimouri et al., 2019; Crisóstomo et al., 2020), and crop yield estimation (Kuwata and Shibasaki, 2015; Wang et al., 2018; Elavarasan and Vincent, 2020). Recently, a few studies focused on mapping irrigation systems using deep learning (Zhang et al., 2018; de Albuquerque et al., 2020; Saraiva et al., 2020; and Tang et al., 2020a), but they mainly investigated center pivot (CP) irrigation system (Carvalho et al., 2020).

Despite the significant milestones achieved in application of Machine Learning and remote sensing in the field of agriculture, developing effective and accurate remote sensing-based Machine Learning models to distinguish various irrigation systems beyond the traditional focus on center pivot irrigation remains a challenge, especially in areas in which wildland and urban areas intermingle with agricultural fields. The goal of this research was to develop a novel deep learning model for the segmentation of four agricultural irrigation systems, as well as background/wildland and urban areas, based on very high resolution remote sensing imagery. The main contributions of this study are:

1. Leveraging advances in Machine Learning and remote sensing to classify different irrigation types, beyond merely center pivot, at a large scale.

2. Developing a machine learning-ready dataset for use in irrigation classification/segmentation studies. This includes airborne imagery with a 1-meter resolution labeled with four irrigation types as well as urban and wildland areas.
3. Investigating the role of Transfer Learning, imbalanced training data, and various deep learning model structures on the efficacy and efficiency of segmentation of different irrigation systems.

The outcomes of this research are essential for policy-makers and agricultural managers in support of land and water management decisions. In the following, Section 2 introduces the study area and the architecture of the proposed model. In Section 3, we discuss the results and Section 4 summarizes the concluding remarks.

2. Data and Analytical Steps

In this section we briefly discuss our modeling approach, data and study area. Background information about neural networks and convolutional models are provided in Sections S1-S2, Table S1 and Figs. S1-S11 in the Supplementary Information.

2.1. Study Area

The study area is located between 43.45° and 43.81° N latitude, and 62.99° and 63.81° W longitude, in Ada and Canyon counties in Idaho, USA. We considered six different classes of land use/cover and irrigation types in the study area (also see Fig. S12 in the Supplementary Information), as follows:

Class 1: Irrigated areas with a Center-Pivot (CP) irrigation system
Class 2: Irrigated areas with a Linear sprinkler Irrigation (LI) system
Class 3: Irrigated areas with a Center Pivot with Swing Arm (CPSA) irrigation system
Class 4: Irrigated areas with a furrow Surface Irrigation (SI) system
Class 5: Urban Area (UA)
Class 6: Wildland/Background area (BG)

2.2. Data Preparation

In this study, we used high resolution (1 meter) imagery from the National Agriculture Imagery Program (NAIP), which acquires aerial imagery during the agricultural growing seasons in the continental U.S. every three years (used to be a five-year cycle before 2008). The images contain 4 bands of Red, Green, Blue, and Near-infrared. To develop a Machine Learning-ready dataset, we first randomly generated more than 8,600 coordinates within the study region to serve as center points of NAIP image patches. The size of each image patch was set at 400×400 pixels (400x400 meters). We then employed Arcpy under ArcGIS python engine to label each pixel in each image using geo-referenced shapefiles of irrigation types that were acquired through on-the-ground field investigation in 2011 by the U.S. Department of Agriculture’s Agricultural Research Service in Boise, ID, U.S. We also used a 2011 version of NAIP imagery from the study area.

We then divided the imagery collection into three sets, with 5,700 images for training (66%), 1,450 images for validation (17%) and 1,450 images for testing (17%), as presented in Fig. S13 in the Supplementary Information. The data split was designed to ensure a similar proportion of irrigation classes in the train, validation, and test datasets. Table 1 enlists the area of each of the six classes. As an example, Fig. S14 in the Supplementary Information shows four samples of the original images compared to the labeled images.

We discarded all images with missing pixels. We ensured that the overlap between the various images was minimal (mostly <10 m) to reduce the redundant information. This minimal overlap enabled capturing information from image edges and it prevents the loss of information for borders (de Albuquerque et al., 2020a). Finally, we used original channel values (RGB) without preprocessing, noting that the model included a batch normalization component.

Table 1. Area (m^2) of each class in the train, validation, and test datasets

	Area ($\times$ 384 $\times$ 384 m^2) in each class					
	BG	SI	LI	CP	CPSA	UA
Train	1611	1820	300	661	622	709
Validation	458	493	78	150	120	202
Test	340	510	70	141	127	195

2.3. Model Architecture

For the segmentation model, we employed a U-Net architecture proposed by Ronneberger et al. (2015). The model consists of an encoder and a decoder built by Convolutional blocks. The U-Net architecture offers a unique property (i.e., skip connections), which sets it apart from other encoder-decoder models. At each spatial resolution level of the U-Net model, the skip connection copies the activation outputs to the same resolution level in the decoder (Fig. 1). The skip connections prevent a common problem in image-to-image segmentation called “loss-of-resolution”, where the input image is encoded to a coarser level and loses important spatial information in the upsampling process.

We note that increasing the number of layers (deepening the model) increase the number of parameters, i.e., degrees of freedom, enabling the model to fit to more complex problems (He et al., 2016). This, however, prompts a “vanishing gradient” in a gradient-based learning framework after a certain number of layers, and in turn, impedes the model from learning from the data (Glorot and Bengio, 2010). In such situations, neural network’s weights that are tuned proportionally to the partial derivative of the loss function cannot be updated effectively (He et al., 2016). In other words, as the gradient flows are updated from the last layers to the first, their values diminish. Thus, the earlier layers’ weights do not update, and the model’s performance does not improve by iteration (He et al., 2016; Pedamonti, 2018). Therefore, it is not possible to deepen a model beyond a certain level, because it loses its generalization capability. Skip (residual) connection, a solution used in Residual Networks (Resnet) to deal with the vanishing gradient problem (He et al., 2016), was employed in our model. For more details, refer to the Supplementary Information, Section S2 and Fig. S11. Finally, the parameters of our network were tuned using a gradient-based backpropagation algorithm (Section S1 in the Supplementary Information).

We employed Keras and TensorFlow (Abadi et al., 2016) python libraries to implement the deep learning-based image segmentation model (https://github.com/qubvel/segmentation_models). Fig. 1 presents a schematic view of the model architecture. First, the model decreased the image size from 384×384 to 192×192 pixels and increased the depth from 3 to 64. This process was followed by a max-pooling layer to decrease the image size from 192×192 to 96×96 pixels, keeping the depth at 64. Subsequently, a series of consecutive functions, including convolution layers, batch normalization, ReLU activation function and zero-padding, were repeated several times while decreasing the image size by 50 percent and doubling its depth in each step. Each pair of convolution layers was connected by a skip connection to minimize the vanishing gradient problem. At the end of the encoding path, the image size reached 12×12 pixels with a depth of 512.

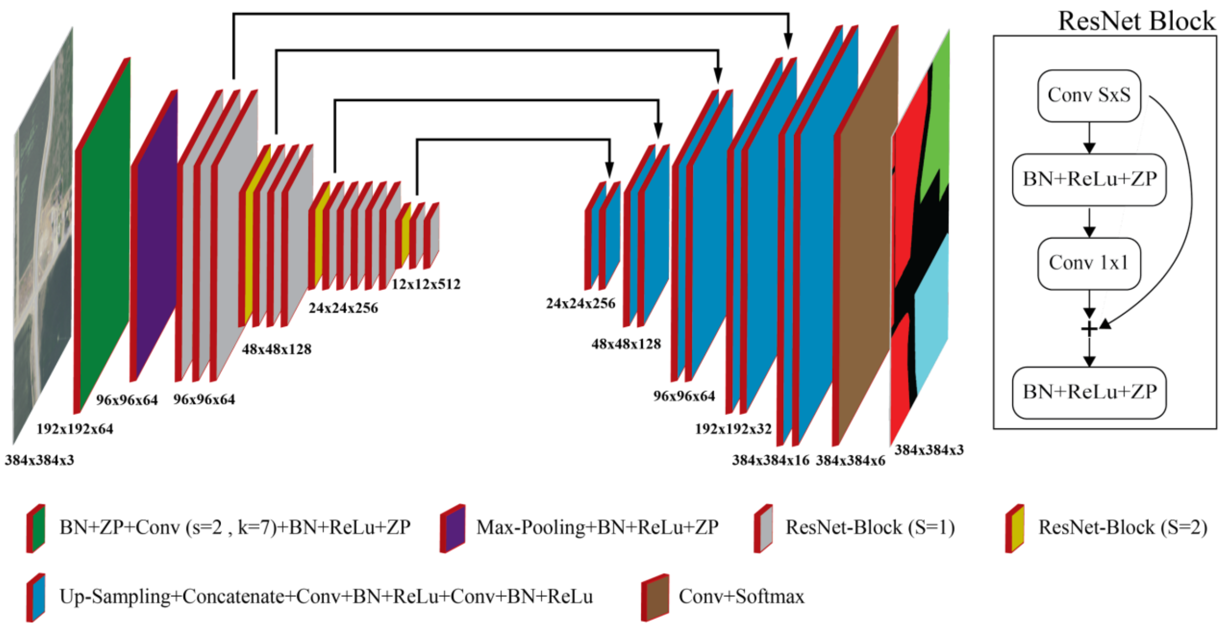


Fig. 1. The U-net architecture used in this study

Several stages follow on the right-side of the U-net for decoding, similar to encoding stages, to return encoded vectors (12×12×512) to image labels (384×384×6). Each stage consisted of two convolution layers, while decreasing the image depth by 50 percent and doubling the image size through 2-D convolution layers, batch normalization and ReLU activation function. At the end of this path, the image size reached 384×384 pixels with a depth of 16. The process was followed by a convolution and a Softmax activation to decrease the image depth from 16 to 6, considering the six different classes for image segmentation.

We leveraged adaptive learning and early-stopping techniques to improve learning and avoid overfitting. If the model performance did not improve after 5 epochs, the learning rate was divided by 10. The maximum epoch number was set to 50, but the model was terminated after 30 epochs if there was no improvement. We employed the Adadelta optimization algorithm to tune model weights, as it outperformed Adam and Stochastic Gradient Descent optimizers (Fig. S15 in the Supplemental Information). The initial and minimum learning rates were set as 1.0 and 0.00001, respectively. We used categorical cross-entropy for the loss function calculation. Since there was a higher proportion of BG and SI classes compared to other classes (Table 1), we used a weighting scheme, with lower weights for BG and SI, to account for the class distribution imbalance in the dataset. Furthermore, we used a Transfer Learning (TL) method based on a pre-trained model on "ImageNet" (Deng et al. 2009) to improve the training process of our deep model. ImageNet is a large database that includes more than 1 million images, including instances of people, animals, plants, and vehicles, among many more. This database has been used to train different models. Using a pre-trained model leads to a significant reduction in the validation loss (Huh et al. 2016; Chen et al. 2020; Cheng et al. 2018b).

2.4. Methodological Choices

Several factors control the proposed model's accuracy, including the type of residual network, the image size, and the class distribution, which are discussed in this section.

Model Selection: We compared our proposed residual network model's performance to other models such as Xception, Linknet, PSPNet and Unet with different backbones.

Determining the Most Efficient Residual Network: We compared four types of Residual Networks (Resnet), including Resnet 18, Resnet 34, Resnet 50, and Resnet 101, which have 8, 16, 16, and 33 skip connections, respectively. For more information regarding the structure, layers, parameters, and other details refer to He et al., 2016.

Investigation of the Image Size: Image size had a significant effect on model performance. Increasing image size enhanced its information content and feature richness, but also increased the computational time, mainly because the number of parameters in the model increased (de Bem et al., 2020). We compared the model performance for three different image sizes (128×128, 256×256, and 384×384 pixels). We cropped the original 400×400 pixels images, rather than resizing them, to preserve the spatial information content of the images.

Controlling for an Imbalanced Dataset: We investigated the effect of assigning different weights to the background and surface irrigation classes, since they covered larger areas than other classes (table 1) and thereby deviations in model classification from their original classes resulted in disproportionately larger impacts on the loss function (Table 2). The weights are applied using "class_weight" parameter in the Keras' fit function.

Table 2. Six scenarios assigning different weights to Background (BG) and Surface Irrigation (SI) classes

	BG	SI	LI	CP	CPSA	UA
No. 1	1	1	1	1	1	1
No. 2	<u>0.1</u>	1	1	1	1	1
No. 3	1	<u>0.1</u>	1	1	1	1
No. 4	<u>0.01</u>	1	1	1	1	1
No. 5	1	<u>0.01</u>	1	1	1	1
No. 6	<u>0.1</u>	<u>0.1</u>	1	1	1	1

***Investigation of Model Hyperparameters*:** We compared three optimizers, including Adam, SGD and Adadelta. Also, we optimized batch sizes, learning rates and momentum for this model through trial and error.

3. Results and Discussion

In this section, we investigate the effect of each factor mentioned in section 2.4. on the model's performance.

Determining the Most Efficient Residual Network: As shown in Fig. 2, Resnet-34 provided the lowest validation loss compared to other Resnet models, although it converged more slowly. Fig S15 shows a comparison between different Resnet structures for two samples, as an example. Table 3 presents four performance metrics, namely intersection over union (IOU), F1, precision and recall for different Resnet structures. As shown here, Resnet-34 offered best performance metrics in all categories. The results of Resnet 34 are presented in the remainder of this paper.

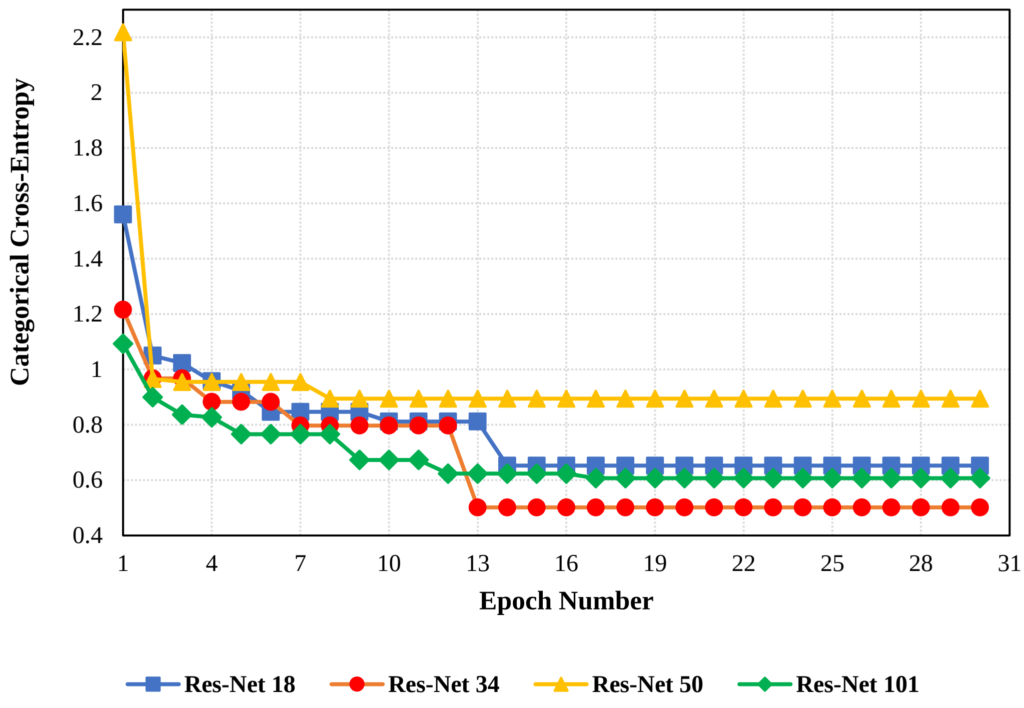


Fig. 2. Comparing different Resnet models in the U-Net architecture

Table 3: Comparing four metrics for different Resnet structures

Backbone	F1-Score	Precision	recall	IOU
Resnet-18	0.79	0.87	0.78	0.72
Resnet-34	0.82	0.88	0.80	0.74
Resnet-50	0.73	0.85	0.70	0.64
Resnet-101	0.76	0.86	0.74	0.68

Investigation of the Image Size: As shown in Fig. 3, the image size of 384×384 pixels resulted in the lowest validation loss compared to that of images with 128×128 and 256×256 pixels, and although it had a higher computational time, it was selected to present the geospatial information needed for the model to perform accurately. Fig S16 shows a comparison between three image sizes and predictions. By increasing the image size, the covered area by a single image increases, which informs the model to outperform when dealing with large farms – specifically by better capturing the information from farm boundaries and corners. Table 4 shows that the model's performance improved significantly by increasing image size.

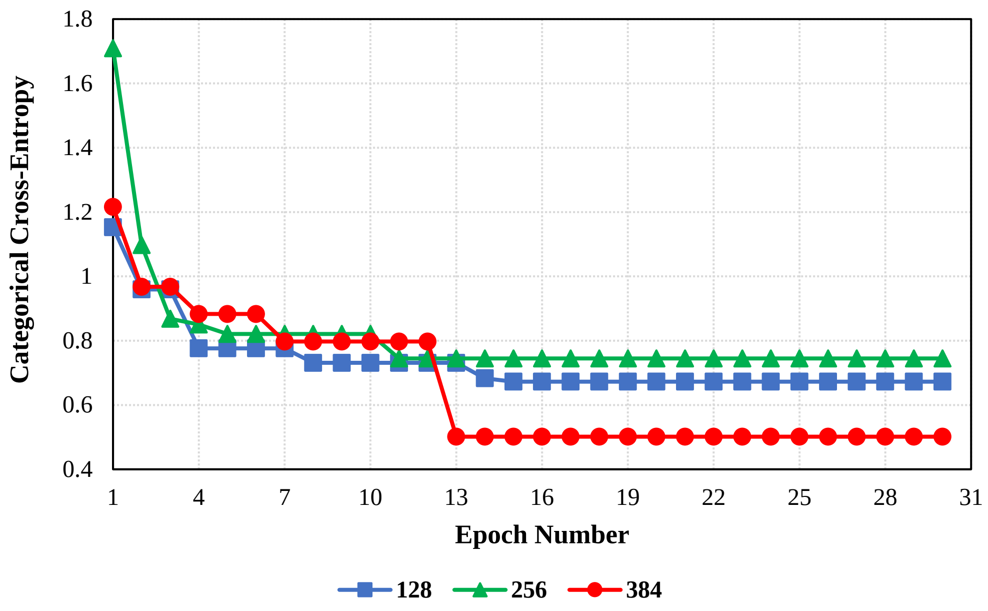
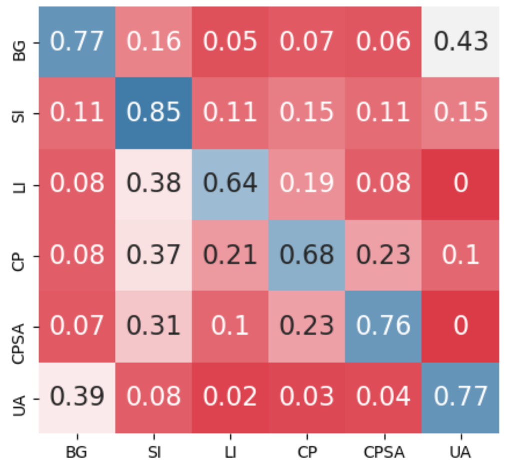


Fig. 3. Effect of image size on the model performance

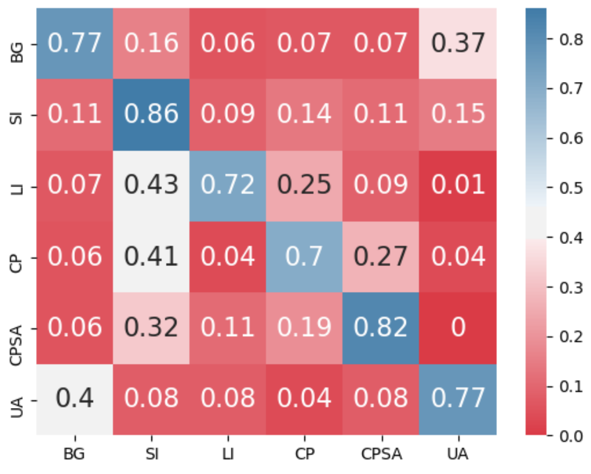
Table 4: The effect of image size on the model performance

Input Size	F1-Score	Precision	recall	IOU
Size 128x128	0.627	0.76	0.61	0.57
Size 256x256	0.72	0.83	0.7	0.65
Size 384x384	0.82	0.88	0.80	0.74

Controlling for an Imbalanced Dataset: Validation loss obtained a minimum value when we assigned the weight of 0.1 to SI and 1.0 to all other classes. Fig. 4 compares the confusion matrix in a model that ignored an imbalanced distribution (i.e., equal weights were assigned to all classes; Fig. 4a), with a model that assigned the weight of 0.1 to the SI class and 1 to all others (Fig. 4b). Each element of the confusion matrix "a_{ij}" presents the percentage of correct model prediction, where "i" and "j" refer to "the actual class in the image" and "the predicted class by the model", respectively. These results represented averaged values from 1,450 images of the test data. A comparison of the elements on the diagonal of the confusion matrices in Fig. 4, indicated that the best performance of the model was obtained by assigning a weight of 0.1 to the SI class. As an example, the prediction of the LI class increased from 64% to 72% after controlling for class distribution by tuning the weights of different classes. Table 5 compares the model performance for different combination of weights.



(a)



(b)

Fig. 4. Confusion matrices for (a) assigning equal weights to all classes (i.e., ignoring the imbalanced distribution), (b) assigning a weight of 0.1 to the Surface Irrigation (SI) class and 1 to all other classes

Table 5: Comparing the model performance for different combination of weights.

Class Weights	F1-Score	Precision	recall	IOU
No. 1	0.80	0.87	0.78	0.72
No. 2	0.80	0.87	0.79	0.73
No. 3	0.82	0.88	0.80	0.74
No. 4	0.8	0.87	0.78	0.73
No. 5	0.77	0.87	0.74	0.68
No. 6	0.80	0.87	0.79	0.73

Fig S17 shows a comparison between three weights of NO. 1, 2, and 3 on four samples.

Investigation of Hyperparameters: A batch size of 25 was selected in this study, based on available GPU's memory. For the Adadelta optimizer, a learning rate of 1 was chosen through trial and error. For other optimizers, the default values of the learning rate and hyperparameters were selected. Table 6 shows model's performance for different optimizers. The best results are for the Adadelta optimizer with learning rate of 1 and rho of 0.95. The model of Unet with Resnet-34 as backbone with Adadelta optimizer is presented in the remainder of this paper. Fig S18, shows validation loss for different optimizers.

Table 6: Model's performance for different optimizers.

Optimizer	F1-Score	Precision	recall	IOU
SGD	0.69	0.80	0.67	0.59
Adam	0.76	0.85	0.74	0.68
Adadelta (lr=1)	0.82	0.88	0.80	0.74

Comparing Different Models: Table 7 compares four metrics between our proposed model (Unet with Resnet-34) and other models including Xception, Linknet, PSPNet, and Unet with different backbones. The input image size (384x384), optimizer (Adadelta), hyperparameters (batch size of 25, learning rate of 1), and class weights are set to be similar for all models. The Unet structure with Resnet-34 provided similar results to Xception, and they outperform other models.

Table 7: Comparing performance metrics among different models

Architecture (Backbone)	F1-Score	Precision	recall	IOU
Linknet (Resnet-34)	0.79	0.87	0.77	0.71
Linknet (efficientnetb3)	0.77	0.86	0.76	0.70
PSPNet (Resnet-34)	0.75	0.82	0.75	0.66
PSPNet (efficientnetb3)	0.74	0.85	0.73	0.66
DeepLabV3 (Resnet-34)*	0.76	0.86	0.75	0.69
Unet (densenet121)	0.80	0.88	0.78	0.72
Unet (Inception)	0.80	0.87	0.79	0.74
Xception**	0.81	0.88	0.79	0.74
Unet (Resnet-34)***	0.82	0.88	0.80	0.74

* https://keras.io/examples/vision/deeplabv3_plus/#building-the-deeplabv3-model
** https://www.kaggle.com/code/meaninglesslives/unet-xception-keras-for-pneumothorax-segmentation/notebook
*** The proposed optimized model

Using a Transfer Learning (TL) Approach: Fig. 5 compares the TL-based pre-trained model with a similar model without pre-training. The TL-based pre-trained model led to a 37.8% reduction in the validation loss (categorical cross-entropy). Although it converged more slowly, TL-based model was selected for this study.

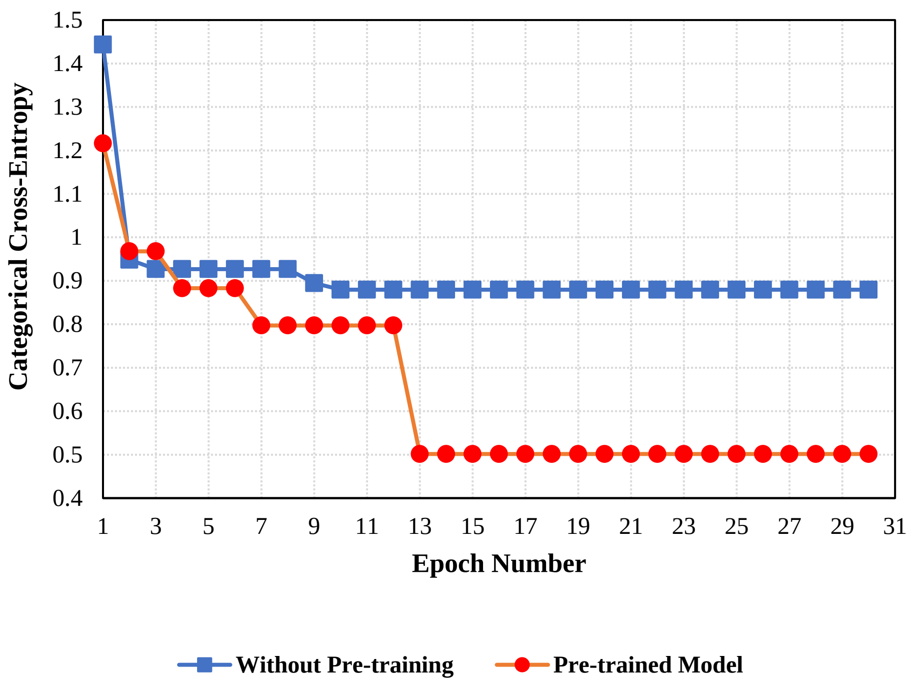
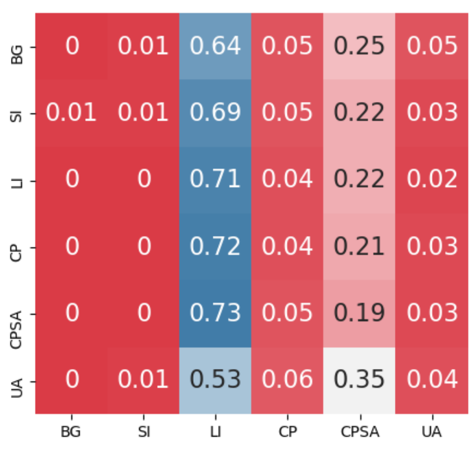
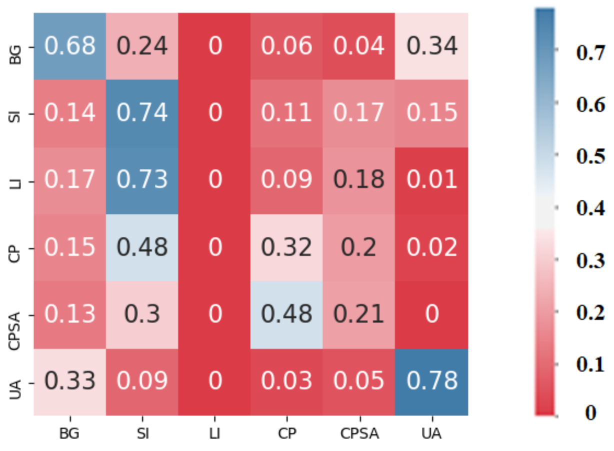


Fig. 5. Comparing the TL-based pre-trained model with a similar model without pre-training

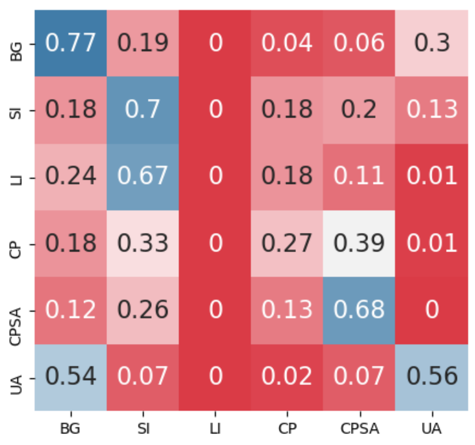
Performance of the Proposed Modeling Approach: Confusion matrices were developed for the test data in different stages: during the pre-training stage and after epochs 1, 2, 4, 7, and 13 (Fig. 6). The elements on the diagonal of the confusion matrices in Fig. 6 indicated that the model performance for the prediction of each class improved as the training progressed (i.e., epoch number increased). As shown in Fig. 3, the training converged to an optimum value at epoch 13, and the loss function (cross entropy) in the steps presented in Fig. 6 adopted values of 1.22, 0.97, 0.88, 0.80, and 0.50, respectively.



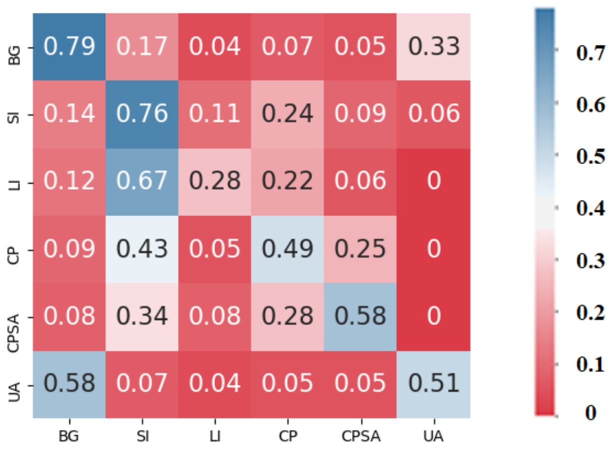
a) Pre-training stage



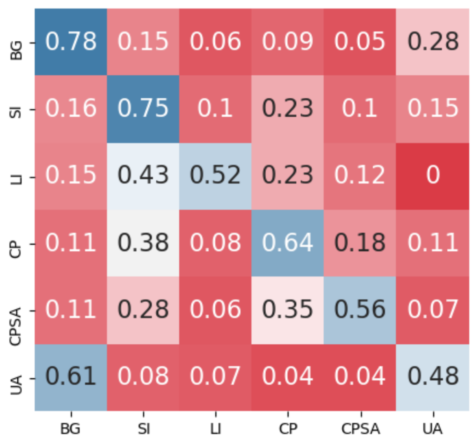
b) Epoch 1



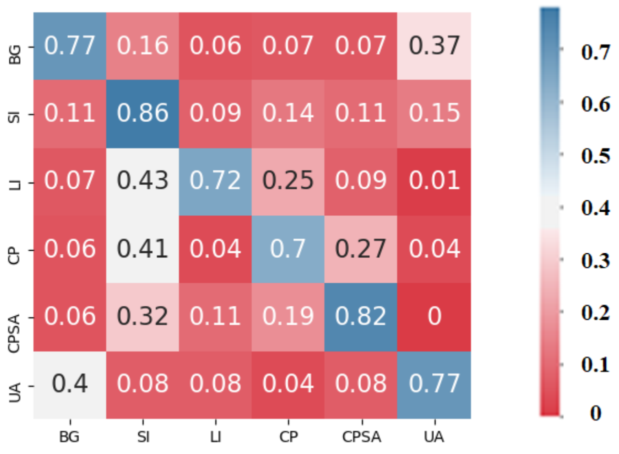
c) Epoch 2



d) Epoch 4



e) Epoch 7

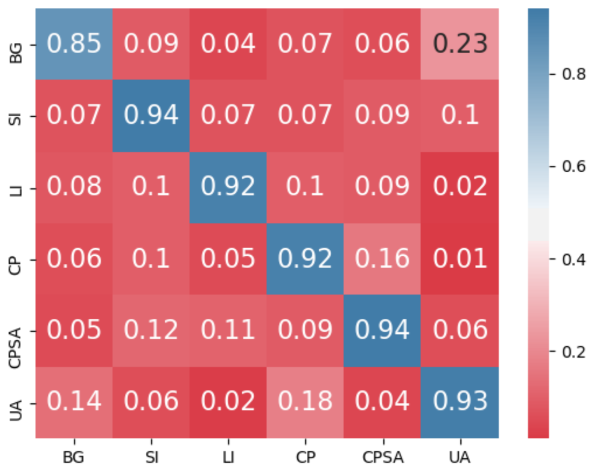


f) Epoch 13

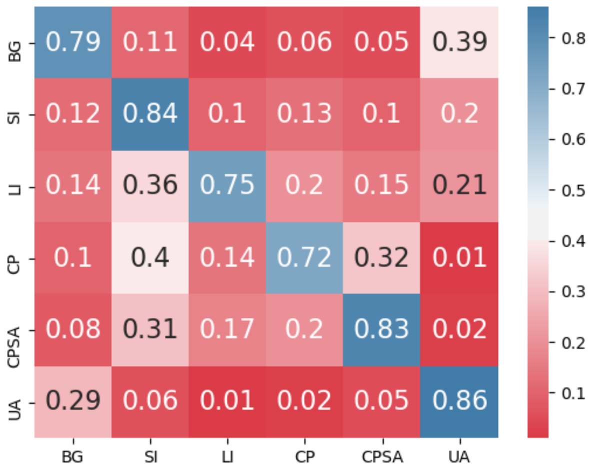
Fig. 6. Confusion matrices in different stages based on the test data

In the pre-training stage, the model recognized almost all features as LI and CPSA classes (the third and the fifth columns in Fig. 6a). After the first iteration (Fig. 6b), the prediction of BG, SI and UA classes improved significantly, implying that the model was capable of detecting these classes. However, the model's ability to distinguish LI, CP and CPSA classes from the SI class was poor at this stage (Fig. 6b). It started to improve in subsequent epochs.

Fig. 7 displays the final confusion matrix for the train and validation datasets, corresponding to the same epoch number (13) as for the test data in Fig. 6f. The developed deep network correctly classified irrigation systems (SI, LI, CP and CPSA) for at least 92% and 72% of cases in the train and validation data, respectively. Table 8 compares four metrics between train, validation and test datasets.



a) Train dataset



b) Validation dataset

Fig. 7. Final confusion matrix after training the deep learning algorithm

Table 8. Performance metrics for the Resnet 34 irrigation type classification model

	IOU	F1	Precision	Recall
Train	0.91	0.95	0.95	0.95
Validation	0.82	0.83	0.88	0.82
Test	0.74	0.82	0.88	0.8

Figs. 8-9 show how the model classification evolved in three arbitrarily selected images in epochs 1, 2, 4, 7, and 13 (corresponding to those in Fig. 5). The confusion matrix in Fig. 6e showed that the model could not recognize the CP class properly until epoch 7 and predicted it as SI and CPSA. Figs. 8c,d,e illustrate this false detection of CP (blue), which was classified (see field on the left) mostly as SI (red) and CPSA (cyan) until epoch 7 when detection of CP started to improve, as shown in Fig. 6e and Fig. 8f. Only in epoch 13 (Fig. 8g) did the model accurately identify boundaries for this class.

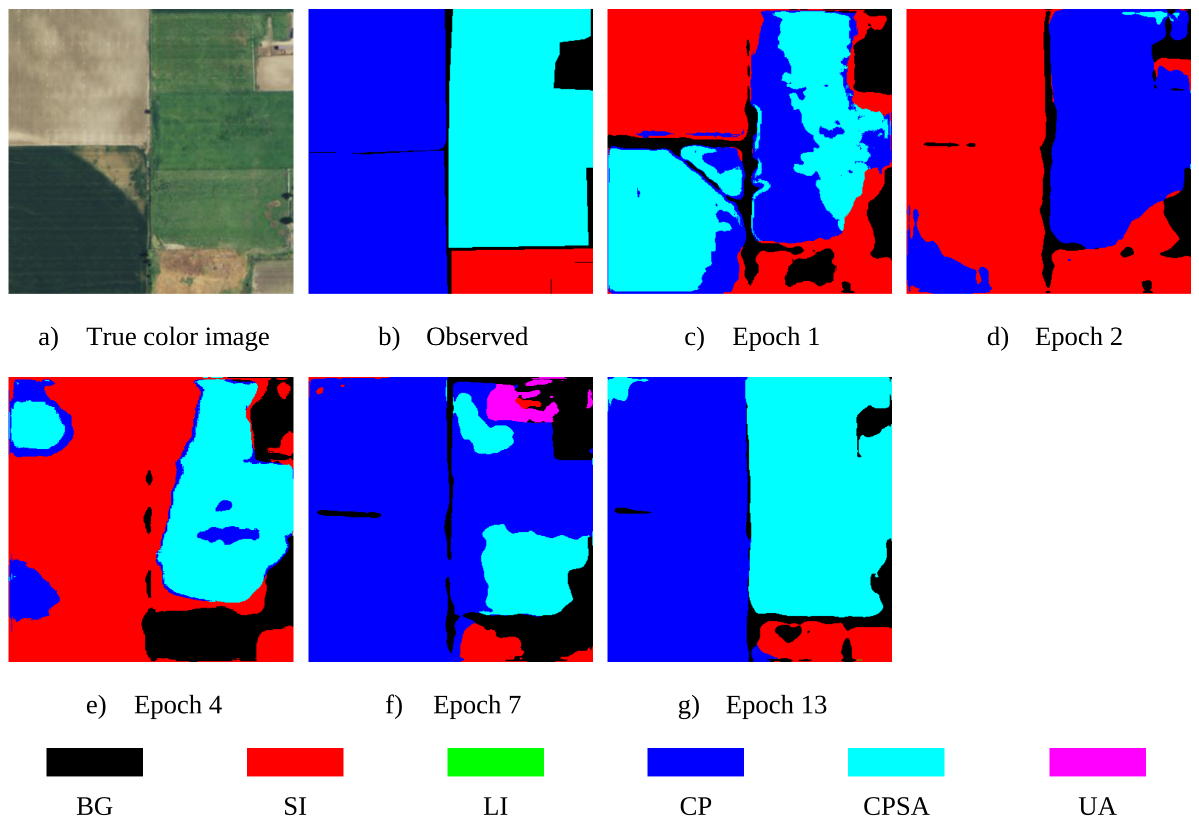


Fig. 8. Evolution of the performance of the Resnet-34 model for the detection of different classes

Fig. 9 shows that the model detected class boundaries very accurately even at epoch 1, and was effective in detecting the BG class. However, it took until epoch 7, corresponding to the confusion matrix in Fig. 6e, for the model to learn to correctly detect the LI class (green) (Fig 9f). At this stage, the model was still not able to distinguish between LI (green) and SI (red) classes. Finally, at epoch 13, the model reached its best performance.

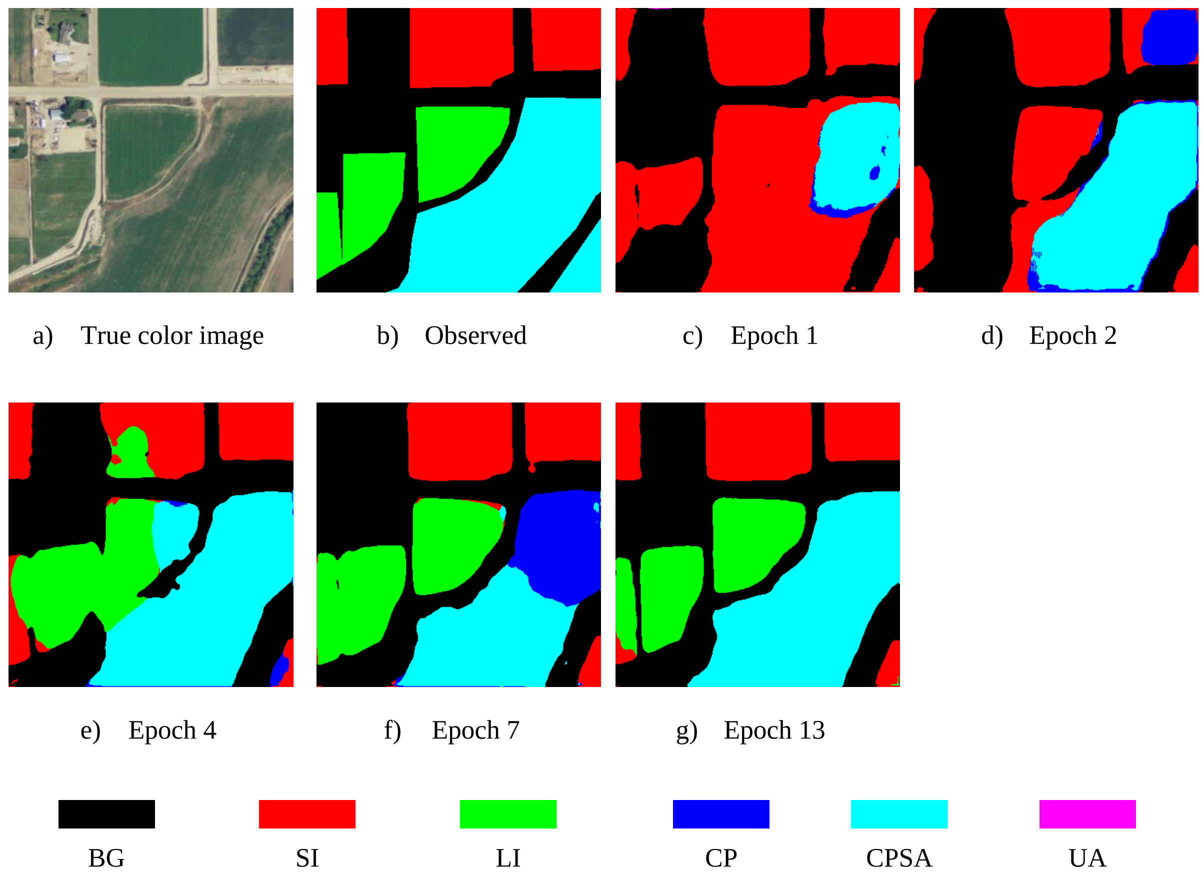


Fig. 9. Evolution of the performance of the Resnet-34 model for the detection of different classes

Limitations of the Proposed Model: As shown in Fig. 6f, there was some confusion in the model prediction between BG and UA classes. This was mainly because large portions of both classes included trees, which confused the model when trying to distinguish the two. However, our purpose was mostly to distinguish irrigation systems, and differentiating between background and urban areas was not a major focus of this study. If the latter was important, we would tune class weights to improve this classification. Fig. 10 provides an example of how UA and BG classes were not accurately distinguished from one another, noting that the model was at least 72% accurate in detecting various classes.

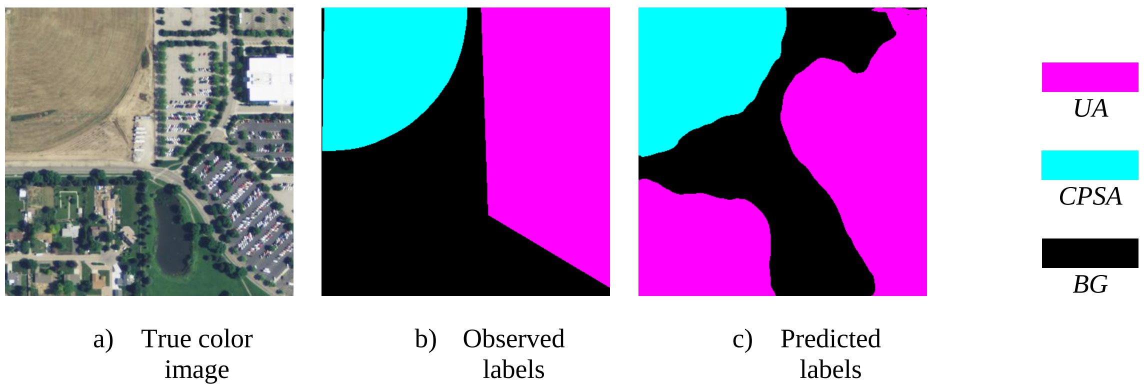


Fig. 10. Limitations of the proposed model in differentiating between Background (BG) and Urban Area (UA) classes

4. Conclusion

In this paper, we developed a deep neural network with a U-Net architecture with Resnet-34 as its backbone to classify irrigation systems in agricultural fields. We used very high resolution (1 meter) airborne imagery from the USDA's National Agriculture Imagery Program (NAIP). Our main goal was to use image segmentation to distinguish four different irrigation systems as well as Background and Urban Areas in a study region located in

Idaho, USA. We proposed this model as a proof-of-concept to enable regional to global monitoring of irrigation systems, which have different water consumption footprints. Such information is critical for management and planning of food and water resources in a warming climate and with an ever-increasing population pressure on natural resources.

We assessed the impacts of different settings on model performance. We tested various image sizes and selected a 384x384 pixel (384x384 meters) image size as the most informative. Increasing image size generally improves model accuracy as more information is provided, but it also increases the computational time due to the amplified number of model parameters. To decrease the misclassification of classes due to imbalanced train data (i.e., disproportionately larger areas covered by furrow Surface Irrigation than by any individual irrigation system), we applied a reductive weight to the furrow Surface Irrigation class. Additionally, we employed transfer learning using models trained on the ImageNet dataset, which showed better performance compared to similar models without the pre-training.

Finally, we observed that even one epoch of training significantly improved detection of several classes, but the confusion among irrigation classes was minimized in epoch 13. The final model accuracy reached 85%, 94%, 92%, 92%, 94%, and 93% for the train data, 79%, 84%, 75%, 72%, 83%, and 86% for the validation data and 77%, 86%, 72%, 70%, 82%, and 77% for the test data for the Background, furrow Surface Irrigation, Linear Irrigation, Center Pivot, Center Pivot with Swing Arm, and Urban Area classes, respectively. The proposed deep segmentation model has the potential to classify multiple irrigation systems and can be applied at regional to global scales using very high resolution remote sensing imagery, for example from commercial satellites.

5. Data and Code Availability

All data and codes for this study are available at: https://github.com/ehsanraei/Irrigation

6. Competing Interests

The authors declare no competing or conflicting interests.

7. Acknowledgments:

The authors thank Dr. Jodi Brandt of Boise State University for providing the shapefiles of various irrigation systems and her input to this study. MS appreciates partial financial support from the U.S. Bureau of Reclamation (award number R21AP10404).

8. References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

Alizadeh, M. R., Abatzoglou, J. T., Luce, C. H., Adamowski, J. F., Farid, A., & Sadegh, M. (2021). Warming enabled upslope advance in western US forest fires. *Proceedings of the National Academy of Sciences*, *118*(22).

Bjorck, N., Gomes, C. P., Selman, B., & Weinberger, K. Q. (2018). Understanding batch normalization. In *Advances in Neural Information Processing Systems* (pp. 7694-7705).

Cerron, B., Bazan, C., & Coronado, A. (2020) Detection of housing and agriculture areas on dry-riverbeds for the evaluation of risk by landslides using low-resolution satellite imagery based on deep learning. Study zone: Lima, Peru. Published as a conference paper at ICLR 2020, National University of Engineering.

Chen, J., Chen, J., Zhang, D., Sun, Y., & Nanehkaran, Y. A. (2020). Using deep transfer learning for image-based plant disease identification. Computers and Electronics in Agriculture, 173, 105393.

Cheng, G., Li, Z., Han, J., Yao, X., & Guo, L. (2018a). Exploring hierarchical convolutional features for hyperspectral image classification. IEEE Transactions on Geoscience and Remote Sensing, 56(11), 6712-6722.

Cheng, G ., Yang, C., Yao, X., Guo, L., & Han, J. (2018b). When deep learning meets metric learning: Remote sensing image scene classification via learning discriminative CNNs. IEEE transactions on geoscience and remote sensing, 56(5), 2811-2821.

Crist, E., Mora, C., & Engelman, R. (2017). The interaction of human population, food production, and biodiversity protection. *Science*, *356*(6335), 260-264.

De Albuquerque, A. O., de Carvalho Júnior, O. A., Carvalho, O. L. F. D., de Bem, P. P., Ferreira, P. H. G., de Moura, R. D. S., ... & Fontes Guimarães, R. (2020). Deep semantic segmentation of center pivot irrigation systems from remotely sensed data. *Remote Sensing*, *12*(13), 2159.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248–255).

Dong, J., Xiao, X., Menarguez, M. A., Zhang, G., Qin, Y., Thau, D., ... & Moore III, B. (2016). Mapping paddy rice planting area in northeastern Asia with Landsat 8 images, phenology-based algorithm and Google Earth Engine. *Remote sensing of environment*, *185*, 142-154.

Elavarasan, D., & Vincent, P. D. (2020). Crop Yield Prediction Using Deep Reinforcement Learning Model for Sustainable Agrarian Applications. *IEEE Access, 8,* 86886-86901.

Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, *145*, 311-318.

Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote sensing of Environment*, *202*, 18-27.

Torabi Haghighi, A., Sadegh, M., Behrooz-Koohenjani, S., Hekmatzadeh, A. A., Karimi, A., & Kløve, B. (2020). The mirage water concept and an index-based approach to quantify causes of hydrological changes in semi-arid regions. *Hydrological Sciences Journal*, *65*(2), 311-324.

Hasan, A. M., Sohel, F., Diepeveen, D., Laga, H., & Jones, M. G. (2021). A survey of deep learning techniques for weed detection from images. *Computers and Electronics in Agriculture*, *184*, 106067.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Helber, P., Bischke, B., Dengel, A., & Borth, D. (2018). Introducing eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. In IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium (pp. 204-207). IEEE.

Huh, M., Agrawal, P., & Efros, A. A. (2016). What makes ImageNet good for transfer learning?. arXiv preprint arXiv:1608.08614.

Islam, M., Dinh, A., Wahid, K., & Bhowmik, P. (2017). Detection of potato diseases using image segmentation and multiclass support vector machine. In *2017 IEEE 30th canadian conference on electrical and computer engineering (CCECE)* (pp. 1-4). IEEE.

Johansen, K., Phinn, S., & Taylor, M. (2015). Mapping woody vegetation clearing in Queensland, Australia from Landsat imagery using the Google Earth Engine. *Remote Sensing Applications: Society and Environment*, *1*, 36-49.

Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. Computers and electronics in agriculture, 147, 70-90.

Kussul, N., Lavreniuk, M., Skakun, S., & Shelestov, A. (2017). Deep learning classification of land cover and crop types using remote sensing data. *IEEE Geoscience and Remote Sensing Letters*, *14*(5), 778-782.

Kuwata, K., & Shibasaki, R. (2015). Estimating crop yields with deep learning and remotely sensed data. In *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)* (pp. 858-861). IEEE.

Lee, C. S., Sohn, E., Park, J. D., & Jang, J. D. (2019). Estimation of soil moisture using deep learning based on satellite data: a case study of South Korea. *GIScience & Remote Sensing*, *56*(1), 43-67.

Liu, X., Hu, G., Chen, Y., Li, X., Xu, X., Li, S., ... & Wang, S. (2018). High-resolution multi-temporal mapping of global urban land using Landsat images based on the Google Earth Engine Platform. *Remote sensing of environment*, *209*, 227-239.

Loddo, A., Loddo, M., & Di Ruberto, C. (2021). A novel deep learning based approach for seed image classification and retrieval. *Computers and Electronics in Agriculture*, *187*, 106269.

Patel, N. N., Angiuli, E., Gamba, P., Gaughan, A., Lisini, G., Stevens, F. R., ... & Trianni, G. (2015). Multitemporal settlement and population mapping from Landsat using Google Earth Engine. *International Journal of Applied Earth Observation and Geoinformation*, *35*, 199-208.

Pedamonti, D. (2018). Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *arXiv preprint arXiv:1804.02763*.

Perea, R. G., Ballesteros, R., Ortega, J. F., & Moreno, M. Á. (2021). Water and energy demand forecasting in large-scale water distribution networks for irrigation using open data and machine learning algorithms. *Computers and Electronics in Agriculture*, *188*, 106327.

Powell, W. G. (2009). Identifying land use/land cover (LULC) using National Agriculture Imagery Program (NAIP) data as a hydrologic model input for local flood plain management.

Rad, A. M., Kreitler, J., & Sadegh, M. (2021a). Augmented Normalized Difference Water Index for improved surface water monitoring. *Environmental Modelling & Software, 140,* 105030.

Rad, A. M., AghaKouchak, A., Navari, M., & Sadegh, M. (2021b). Progress, Challenges, and Opportunities in Remote Sensing of Drought. *Global Drought and Flood: Observation, Modeling, and Prediction*, 1-28.

Rad, A. M., Kreitler, J., Abatzoglou, J. T., Fallon, K., Roche, K., & Sadegh, M. (2022). Anthropogenic stressors compound climate impacts on inland lake dynamics: The case of Hamun Lakes. *Science of The Total Environment*, 154419.

Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.

Sadegh, M., AghaKouchak, A., Mallakpour, I., Huning, L. S., Mazdiyasni, O., Niknejad, M., ... & Davis, S. J. (2020). Data and analysis toolbox for modeling the nexus of food, energy, and water. *Sustainable Cities and Society*, *61*, 102281.

Saleem, M. H., Potgieter, J., & Arif, K. M. (2020). Plant Disease Classification: A Comparative Evaluation of Convolutional Neural Networks and Deep Learning Optimizers. *Plants*, *9*(10), 1319.

Saraiva, M., Protas, É., Salgado, M., & Souza Jr, C. (2020). Automatic Mapping of Center Pivot Irrigation Systems from Satellite Images Using Deep Learning. *Remote Sensing*, *12*(3), 558.

Scott, G. J., England, M. R., Starms, W. A., Marcum, R. A., & Davis, C. H. (2017). Training deep convolutional neural networks for land–cover classification of high-resolution imagery. *IEEE Geoscience and Remote Sensing Letters*, *14*(4), 549-553.

Sharma, S., & Mehra, R. (2019). Implications of Pooling Strategies in Convolutional Neural Networks: A Deep Insight. *Foundations of Computing and Decision Sciences*, *44*(3), 303-330.

Song, X., Zhang, G., Liu, F., Li, D., Zhao, Y., & Yang, J. (2016). Modeling spatio-temporal distribution of soil moisture by deep learning-based cellular automata model. *Journal of Arid Land*, *8*(5), 734-748.

Tang, J. W., Arvor, D., Corpetti, T., & Tang, P. (2020a). PVANET-HOUGH: DETECTION AND LOCATION OF CENTER PIVOT IRRIGATION SYSTEMS FROM SENTINEL-2 IMAGES. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, *5*(3).

Tassis, L. M., de Souza, J. E. T., & Krohling, R. A. (2021). A deep learning approach combining instance and semantic segmentation to identify diseases and pests of coffee leaves from in-field images. *Computers and Electronics in Agriculture*, *186*, 106191.

Teluguntla, P., Thenkabail, P. S., Oliphant, A., Xiong, J., Gumma, M. K., Congalton, R. G., ... & Huete, A. (2018). A 30-m landsat-derived cropland extent product of Australia and China using random forest machine learning algorithm on Google Earth Engine cloud computing platform. *ISPRS Journal of Photogrammetry and Remote Sensing*, *144*, 325-340.

Tong, X. Y., Xia, G. S., Lu, Q., Shen, H., Li, S., You, S., & Zhang, L. (2020). Land-cover classification with high-resolution remote sensing images using transferable deep models. *Remote Sensing of Environment*, *237*, 111322.

Wang, A. X., Tran, C., Desai, N., Lobell, D., & Ermon, S. (2018). Deep transfer learning for crop yield prediction with remote sensing data. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies* (pp. 1-5).

Wang, S., Di Tommaso, S., Faulkner, J., Friedel, T., Kennepohl, A., Strey, R., & Lobell, D. B. (2020). Mapping crop types in southeast india with smartphone crowdsourcing and deep learning. *Remote Sensing*, *12*(18), 2957.

Wang, Y., Li, Y., Song, Y., & Rong, X. (2020). The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition. *Applied Sciences*, *10*(5), 1897.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, *9*(4), 611-629.

Zhang, C., Yue, P., Di, L., & Wu, Z. (2018). Automatic identification of center pivot irrigation systems from landsat images using convolutional neural networks. *Agriculture*, *8*(10), 147.

Zhou, Z., Majeed, Y., Naranjo, G. D., & Gambacorta, E. M. (2021). Assessment for crop water stress with infrared thermal imagery in precision agriculture: A review and future prospects for deep learning applications. *Computers and Electronics in Agriculture*, *182*, 106019.

Zhou, P., Han, J., Cheng, G., & Zhang, B. (2019). Learning compact and discriminative stacked autoencoder for hyperspectral image classification. IEEE Transactions on Geoscience and Remote Sensing, 57(7), 4823-4833.

New References

Zhou, Y. N., Luo, J., Feng, L., Yang, Y., Chen, Y., & Wu, W. (2019). Long-short-term-memory-based crop classification using high-resolution optical images and multi-temporal SAR data. *GIScience & Remote Sensing, 56(8),* 1170-1191.

Li, Z., Chen, G., & Zhang, T. (2020). A CNN-transformer hybrid approach for crop classification using multitemporal multisensor images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 13*, 847-858.

de Albuquerque, A. O., de Carvalho, O. L. F., e Silva, C. R., de Bem, P. P., Gomes, R. A. T., Borges, D. L., ... & de Carvalho Júnior, O. A. (2021a). Instance segmentation of center pivot irrigation systems using multi-temporal SENTINEL-1 SAR images. *Remote Sensing Applications: Society and Environment, 23*, 100537.

de Albuquerque, A. O., de Carvalho, O. L. F., e Silva, C. R., Luiz, A. S., Pablo, P., Gomes, R. A. T., ... & de Carvalho Júnior, O. A. (2021b). Dealing With Clouds and Seasonal Changes for Center Pivot Irrigation Systems Detection Using Instance Segmentation in Sentinel-2 Time Series. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 14*, 8447-8457.

Carvalho, O. L. F. D., de Carvalho Junior, O. A., Albuquerque, A. O. D., Bem, P. P. D., Silva, C. R., Ferreira, P. H. G., ... & Borges, D. L. (2020). Instance segmentation for large, multi-channel remote sensing imagery using mask-RCNN and a mosaicking approach. *Remote Sensing, 13(1),* 39.

Tang, J., Arvor, D., Corpetti, T., & Tang, P. (2021b). Mapping center pivot irrigation systems in the southern Amazon from Sentinel-2 images. *Water, 13(3),* 298.

Tang, J., Zhang, Z., Zhao, L., & Tang, P. (2021c). Increasing shape bias to improve the precision of center pivot irrigation system detection. *Remote Sensing, 13(4),* 612.

de Bem, P. P., de Carvalho Júnior, O. A., de Carvalho, O. L. F., Gomes, R. A. T., & Fontes Guimarães, R. (2020). Performance analysis of deep convolutional autoencoders with different patch sizes for change detection from burnt areas. *Remote Sensing, 12(16),* 2576.

Teimouri, N., Dyrmann, M., & Jørgensen, R. N. (2019). A novel spatio-temporal FCN-LSTM network for recognizing various crop types using multi-temporal radar images. *Remote Sensing, 11(8),* 990.

Crisóstomo de Castro Filho, H., Abílio de Carvalho Júnior, O., Ferreira de Carvalho, O. L., Pozzobon de Bem, P., dos Santos de Moura, R., Olino de Albuquerque, A., ... & Trancoso Gomes, R. A. (2020). Rice crop detection using LSTM, Bi-LSTM, and machine learning models from Sentinel-1 time series. *Remote Sensing, 12(16),* 2655.

Supplementary Information for: A Deep Learning-Based Image Segmentation Model for Agricultural Irrigation System Classification

S1. How Does a Neural Network Work?

A neural network aims to find the best values of model parameters (weights and biases) so as to have the closest prediction(s) to the observed value(s). In this process, the input(s) are transformed to the output(s) using a matrix of weights in each layer. This path is called the forward propagation process.

First, these weights are randomly assigned initial values, then the output(s) is/are calculated by a forward propagation process, and finally, the loss value is computed by comparing the output with the observed value(s). Depending on the loss value, an optimizer algorithm will change the values of parameters (weights and biases) slightly and perform the forward propagation process again to see if the loss decreases or not. This path in which parameters are modified is named the backpropagation process (Wang et al., 2020). Depending on the problem, a neural network can change from a simple neural network (with only one hidden layer) to a deep one (with many layers). Fig. S1 shows a simple neural network with four input variables (X_i), one hidden layer with two neurons, and two outputs (O_i).

As shown in Fig. S1, the input variables pass through hidden layers, neurons, and activation functions, and result in predicted values. This process is called the forward propagation path (blue arrows in Fig. S1). In the opposite direction, the path moving from the outputs towards the weights and biases is called the backpropagation process (red arrows in Fig. S1). Mathematically, the relationships between the input variables (X_i), weights (W), and biases (b) are as follows:

$$Z_n^1 = \sum_{i=1}^{4} X_i \times W_{in}^1 + b^1 \tag{S1}$$

$$f_n = \sigma\left(Z_n^1\right) \tag{S2}$$

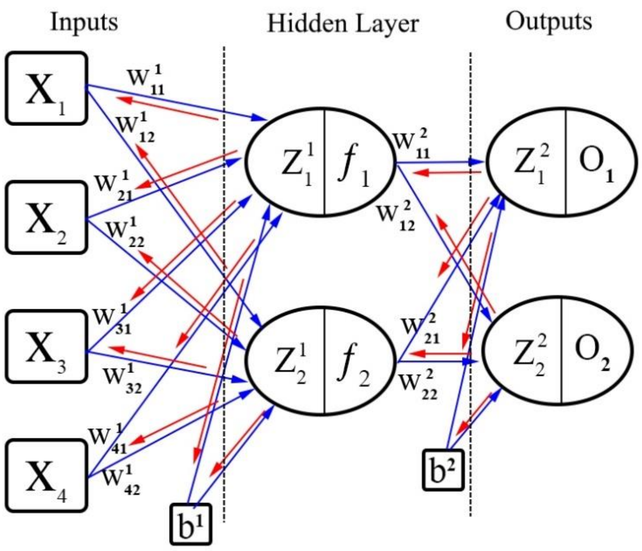


Fig. S1. Schematic view of a simple neural network.
The forward propagation process is shown by blue arrows and the backpropagation process by red arrows.

where, σ is a nonlinear activation function, and n is the neuron index. As an example, the sigmoid activation function (Eq. S3) and its derivative are displayed in Fig. S2. In Eq. S3, e and x are the exponential constant and the input values, respectively.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{S3}$$

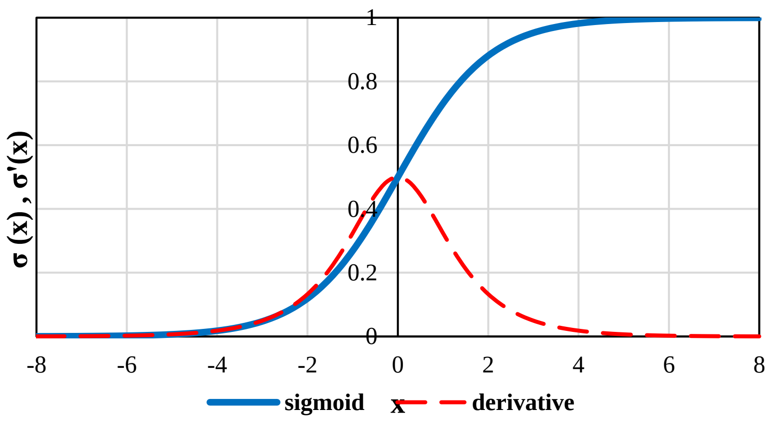


Fig. S2. Sigmoid function and its derivative

The output values (o_i) are obtained as follows:

$$o_i = \sigma(Z_n^2) \tag{S4}$$

$$Z_n^2 = \sum_{i=1}^{2} f_i \times W_{in}^2 + b^2 \tag{S5}$$

$$E_{total} = \frac{1}{2}\left(o_{1\,(Observed)} - o_{1(Predicted)}\right)^2 + \frac{1}{2}\left(o_{2\,(Observed)} - o_{2\,(Predicted)}\right)^2 \tag{S6}$$

In Eq. S6, E_{total} is the error or the loss value that shows the difference between the predicted values of the network outputs and the actual (observed) values. In the backpropagation process, the model will try to minimize the loss value by moving in the opposite direction of the forward propagation path and modifying the network's weights and biases. The level of adjustment is determined by calculating the gradients of the loss function for those parameters (weights and biases) through the chain rule, which multiply all derivatives one by one (Eq. S7 and Eq. S11). The gradient value shows how much the parameter needs to change (in a positive or negative direction) to minimize the loss. The backpropagation process is calculated by the following equations (Eq. S7 to Eq. S10) (Sibi et al., 2013).

As an example, W_{11}^2 (Fig. S1) is optimized as follows:

$$\frac{\partial E_{total}}{\partial W_{11}^2} = \frac{\partial E_{total}}{\partial o_1} \times \frac{\partial o_1}{\partial Z_1^2} \times \frac{\partial Z_1^2}{\partial W_{11}^2} \tag{S7}$$

$$\frac{\partial E_{total}}{\partial o_1} = 2 \times \frac{1}{2}\left(o_{1\,(Observed)} - o_{1(Predicted)}\right)^{2-1} \times -1 + 0 \tag{S8}$$

$$\frac{\partial o_1}{\partial Z_1^2} = \sigma'(x) = \text{derivation of the sigmoid function} \tag{S9}$$

$$\frac{\partial Z_1^2}{\partial W_{11}^2} = \frac{\partial\left(f_1 \times W_{11}^2 + f_2 \times W_{21}^2\right)}{\partial W_{11}^2} = f_1 \tag{S10}$$

And for modifying W_{11}^1:

$$\frac{\partial E_{total}}{\partial W_{11}^1} = \left(\frac{\partial E_{total}}{\partial o_1} \times \frac{\partial o_1}{\partial Z_1^2} \times \frac{\partial Z_1^2}{\partial f_1} + \frac{\partial E_{total}}{\partial o_2} \times \frac{\partial o_2}{\partial Z_2^2} \times \frac{\partial Z_2^2}{\partial f_1}\right) \times \frac{\partial f_1}{\partial Z_1^1} \times \frac{\partial Z_1^1}{\partial W_{11}^1} \tag{S11}$$

$$\frac{\partial f_1}{\partial Z_1^1} = \sigma'(x) \tag{S12}$$

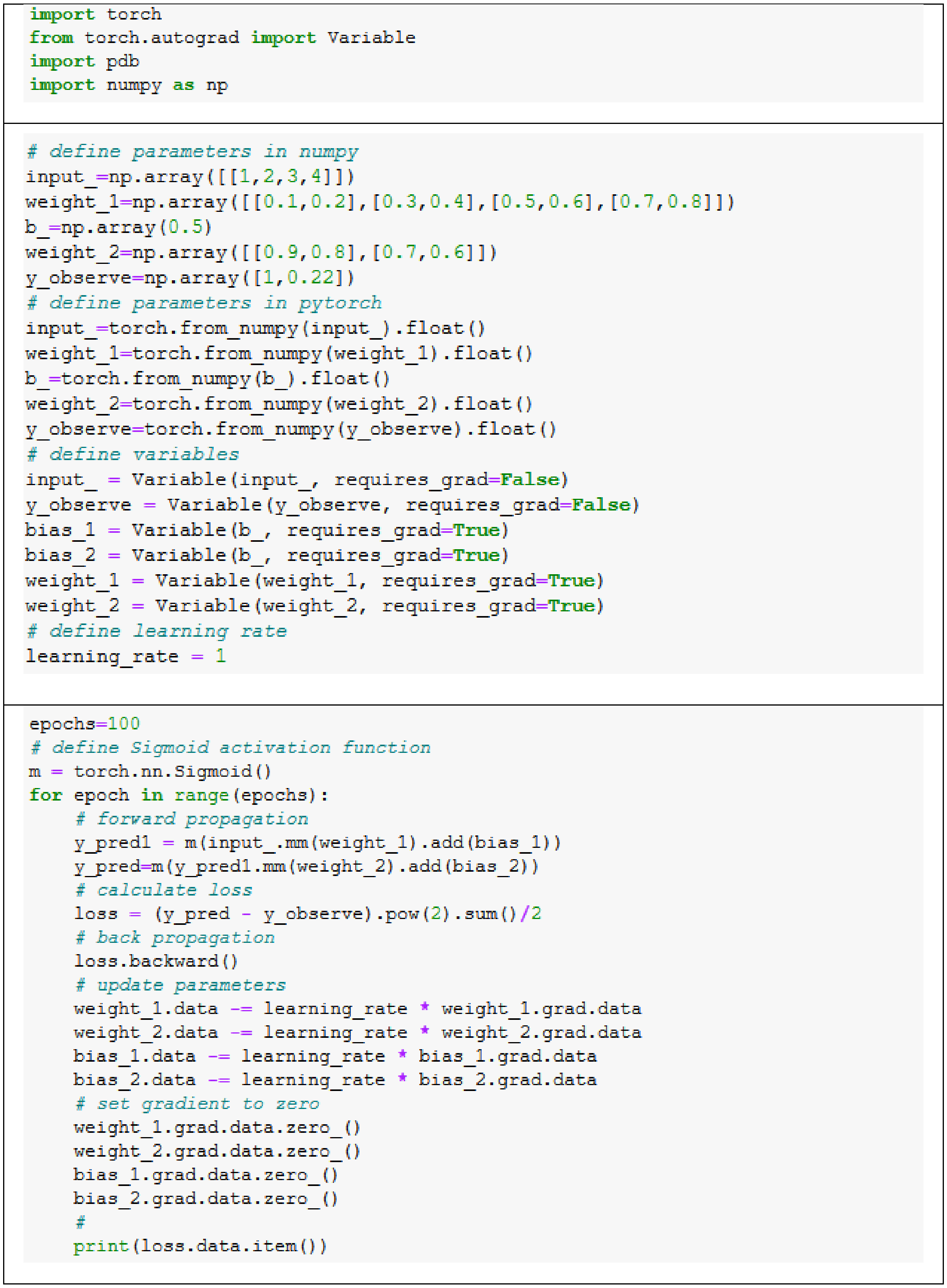
$$\frac{\partial Z_1^1}{\partial W_{11}^1} = \frac{X_1 \times W_{11}^1 + X_2 \times W_{21}^1 + X_3 \times W_{31}^1 + X_4 \times W_{41}^1}{\partial W_{11}^1} = X_1 \tag{S13}$$

<u>Example</u>

In this example, we consider a simple neural network, as shown in Fig. S1. The values of the input variables (X_1, X_2, X_3, X_4) are assumed to be 1, 2, 3, and 4, respectively. The observed values of the outputs (o_1, o_2) are 1 and 0.22. Considering these values and the sigmoid activation function, Table S1 shows the model parameters (weights and biases) and the obtained loss values in 100 training epochs. Epoch No. 0 presents the values initially assigned to the model parameters and the corresponding loss values. The gray row in epoch No. 0 displays the gradient value of the model's error corresponding to each parameter. In the next epoch (No. 1), the model parameters (weights and biases) are obtained by subtracting the former values of weights and biases in epoch No. 0 and the corresponding gradient of each parameter. In this example, for simplicity, we consider the learning rate to be 1. As shown in Table S1, the initial model's loss value is 0.217. After 10 training epochs, the loss value reaches 0.03. Finally, in epoch No. 99, the predicted values will be 0.9508 and 0.2252, corresponding to the observed ones (1 and 0.22), and the loss value reaches 0.001. Fig. S3 shows the python code in PyTorch for model implementation in this example.

Table S1. Model parameters (weights and biases) and the loss values in 100 training epochs for the example above

Epoch	W^1_{11}	W^1_{12}	W^1_{21}	W^1_{22}	W^1_{31}	W^1_{32}	W^1_{41}	W^1_{42}	W^2_{11}	W^2_{12}	W^2_{21}	W^2_{22}	$\boldsymbol{b}^1$	$\boldsymbol{b}^2$	loss
0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.8	0.7	0.6	0.5	0.5	0.217
	2.0E-4	5.51E-5	4.0E-04	1.1E-04	6.0E-04	1.65E-4	8.0E-04	2.2E-04	-1.06E-2	7.34E-02	-1.07E-2	7.36E-2	2.55E-4	6.3E-2	
1	0.1	0.2	0.3	0.4	0.499	0.6	0.699	0.8	0.911	0.727	0.711	0.526	0.437	0.437	0.201
	2.14E-4	5.62E-5	4.28E-4	1.12E-4	6.42E-4	1.69E-4	8.57E-4	2.25E-4	-1.14E-2	8.19E-02	-1.15E-2	8.21E-2	2.7E-04	7.08E-2	
2	0.1	0.2	0.299	0.4	0.499	0.6	0.698	0.8	0.922	0.645	0.722	0.444	0.366	0.366	0.181
	2.21E-4	5.39E-5	4.41E-4	1.08E-4	6.62E-4	1.62E-4	8.83E-4	2.16E-4	-1.24E-2	9.03E-02	-1.24E-2	9.06E-2	2.75E-4	7.83E-2	
3	0.099	0.2	0.299	0.4	0.498	0.6	0.697	0.799	0.934	0.554	0.735	0.354	0.287	0.287	0.157
	2.12E-4	4.58E-5	4.23E-4	9.16E-5	6.35E-4	1.37E-4	8.47E-4	1.83E-4	-1.35E-2	9.73E-02	-1.36E-2	9.76E-2	2.57E-4	8.42E-2	
4	0.099	0.2	0.298	0.4	0.497	0.599	0.697	0.799	0.948	0.457	0.748	0.256	0.203	0.203	0.13
	1.78E-4	2.98E-5	3.57E-4	5.95E-5	5.35E-4	8.93E-5	7.14E-4	1.19E-4	-1.48E-2	1.0E-01	-1.49E-2	1.01E-1	2.08E-4	8.60E-2	
5	0.099	0.2	0.298	0.4	0.497	0.599	0.696	0.799	0.963	0.357	0.763	0.155	0.116	0.116	0.103
	1.17E-4	6.09E-6	2.35E-4	1.22E-5	3.52E-4	1.83E-5	4.70E-4	2.43E-5	-1.62E-2	9.74E-02	-1.63E-2	9.78E-2	1.23E-4	8.16E-2	
6	0.099	0.2	0.298	0.4	0.497	0.599	0.695	0.799	0.979	0.259	0.779	0.058	0.035	0.035	0.078
	3.81E-5	-2.08E-5	7.62E-5	-4.16E-5	1.14E-4	-6.24E-5	1.52E-4	-8.32E-5	-1.76E-2	8.83E-02	-1.76E-2	8.87E-2	1.73E-5	7.12E-2	
7	0.099	0.2	0.298	0.4	0.496	0.599	0.695	0.799	0.997	0.171	0.797	-0.031	-0.037	-0.037	0.058
	-4.03E-5	-4.45E-5	-8.05E-5	-8.91E-5	-1.21E-4	-1.34E-4	-1.61E-4	-1.78E-4	-1.86E-2	7.56E-02	-1.87E-2	7.60E-2	-8.48E-5	5.75E-2	
8	0.099	0.2	0.298	0.4	0.497	0.599	0.695	0.799	1.015	0.095	0.816	-0.107	-0.094	-0.094	0.044
	-1.03E-4	-6.14E-5	-2.05E-4	-1.23E-4	-3.08E-4	-1.84E-4	-4.11E-4	-2.46E-4	-1.92E-2	6.27E-02	-1.93E-2	6.30E-2	-1.64E-4	4.39E-2	
9	0.099	0.2	0.298	0.4	0.497	0.6	0.696	0.8	1.034	0.033	0.835	-0.17	-0.138	-0.138	0.034
	-1.45E-4	-7.14E-5	-2.90E-4	-1.43E-4	-4.36E-4	-2.14E-4	-5.81E-4	-2.85E-4	-1.94E-2	5.15E-02	-1.95E-2	5.18E-2	-2.17E-4	3.24E-2	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	0.106	0.202	0.311	0.405	0.517	0.607	0.722	0.810	1.617	-0.486	1.421	-0.691	-0.064	-0.064	0.001
	-2.50E-5	-9.54E-6	-5.00E-5	-1.91E-5	-7.50E-5	-2.86E-5	-1.00E-4	-3.82E-5	-2.29E-3	9.03E-04	-2.30E-3	9.07E-4	-3.45E-5	-1.39E-3	



```
import torch
from torch.autograd import Variable
import pdb
import numpy as np
```

```
# define parameters in numpy
input_=np.array([[1,2,3,4]])
weight_1=np.array([[0.1,0.2],[0.3,0.4],[0.5,0.6],[0.7,0.8]])
b_=np.array(0.5)
weight_2=np.array([[0.9,0.8],[0.7,0.6]])
y_observe=np.array([1,0.22])
# define parameters in pytorch
input_=torch.from_numpy(input_).float()
weight_1=torch.from_numpy(weight_1).float()
b_=torch.from_numpy(b_).float()
weight_2=torch.from_numpy(weight_2).float()
y_observe=torch.from_numpy(y_observe).float()
# define variables
input_ = Variable(input_, requires_grad=False)
y_observe = Variable(y_observe, requires_grad=False)
bias_1 = Variable(b_, requires_grad=True)
bias_2 = Variable(b_, requires_grad=True)
weight_1 = Variable(weight_1, requires_grad=True)
weight_2 = Variable(weight_2, requires_grad=True)
# define learning rate
learning_rate = 1
```

```
epochs=100
# define Sigmoid activation function
m = torch.nn.Sigmoid()
for epoch in range(epochs):
    # forward propagation
    y_pred1 = m(input_.mm(weight_1).add(bias_1))
    y_pred=m(y_pred1.mm(weight_2).add(bias_2))
    # calculate loss
    loss = (y_pred - y_observe).pow(2).sum()/2
    # back propagation
    loss.backward()
    # update parameters
    weight_1.data -= learning_rate * weight_1.grad.data
    weight_2.data -= learning_rate * weight_2.grad.data
    bias_1.data -= learning_rate * bias_1.grad.data
    bias_2.data -= learning_rate * bias_2.grad.data
    # set gradient to zero
    weight_1.grad.data.zero_()
    weight_2.grad.data.zero_()
    bias_1.grad.data.zero_()
    bias_2.grad.data.zero_()
    #
    print(loss.data.item())
```

Fig. S3. The python code for model implementation in PyTorch for the example above

S2. Convolutional Block

One of the main building blocks of a Convolutional Neural Network (CNN) model is the convolution layer, which consists of several functions (Yamashita et al., 2018) explained below:

Activation Function: A neural network is the sum of products between inputs and their corresponding weights (W) (Eq. S1, S5). The output of each layer is the input of the subsequent one linked to the other neurons of the next layers through weighted connections and multiplications between them. We can summarize all these multiplications as a simple linear function by condensing multiple transformations into a single one (Equations. S1 to S6 can be simplified by discarding Eq. S2 and Eq. S4 or assuming them to be linear). However, this simplification makes the multi-layer network less effective (Wang et al., 2020) and causes the following two main problems:

1) The model can only perform a linear regression. As a result, it cannot learn complex functions because it is a one-degree of freedom model. Fig. S4 shows a classification example with different linear lines to classify two classes. As shown in this figure, none of the linear black lines can separate red circles and blue rectangles, except a non-linear green curve (Nwankpa et al., 2018).
2) A neural network must be able to optimize the weights and biases and minimize the loss value in order to learn. This learning process is done by derivation of the loss function relative to each parameter (the backpropagation process) (Eq. S7 and Eq. S11). Therefore, the loss function should be differentiable. Since the power of the linear multiplication is one, its derivative will be a constant value. Thus, the gradient has no relationship with the parameters, and consequently, the model parameters will not change in the backpropagation process (Goodfellow et al., 2016; Nwankpa et al., 2018).

Based on the above issues, a model needs to have non-linear properties using the activation function to solve complex problems (Fig. S2). In addition to introducing the non-linear properties to the network, most of the activation functions can transfer values between $[-\infty, +\infty]$ into a smaller range (as an example, the sigmoid function (Fig. S2) will transfer all inputs to [0, 1]). Two of the most famous non-linear activation functions are Softmax (Eq. S14) and Rectified Linear Unit (RELU) functions (Eq. S15).

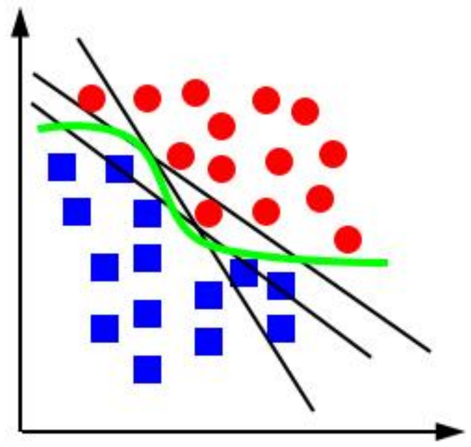


Fig. S4. The role of a non-linear activation function in problems with more than one degree of freedom

$$\sigma(z_i) = \frac{e^{Z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{S14}$$

$$RELU=\begin{cases} x & x>0 \\ 0 & x\le 0 \end{cases} \tag{S15}$$

The Softmax activation function performs well in classification problems. It is usually applied in the last layer of the network for the classification of more than two classes. It determines the probability that the input relates to a particular class by transferring the values into the range of [0, 1].

As mentioned, the model parameters are optimized through the backpropagation process based on the chain rule. This means that when a network has an appropriate number of hidden layers (almost deep), more derivation multiplications are happening for earlier layers rather than the last ones based on the chain rule. Therefore, the gradients tend to get smaller and smaller as if moving backward. Since the gradient values in the earlier layers become very small, the neurons in these layers will learn very slowly compared to those in the last layers. This problem is called a vanishing gradient and usually happens in the Tanh and Sigmoid activation functions. In contrast, an exploding gradient problem will happen when the gradient value is more than one. (Pedamonti, 2018).

RELU is a non-linear activation function and its derivation is equal to one (for inputs more than zero) or zero (for inputs less than or equal to zero) (Nair and Hinton, 2010), as shown in Fig. S5 and Eq. S15. The RELU activation function is used for the prevention of vanishing/exploding gradient issues in deep networks. It also needs less computation time compared to other activation functions, and therefore is not time-consuming in deep networks.

As mentioned, the RELU activation function will filter all negative values and transform them into zero. These weights will not adjust during the backpropagation process because the derivations are zero. Therefore, these neurons will not have any role in the model performance. This is called dying RELU, which can cause problems in some models (Maas et al., 2013; Pedamonti, 2018).

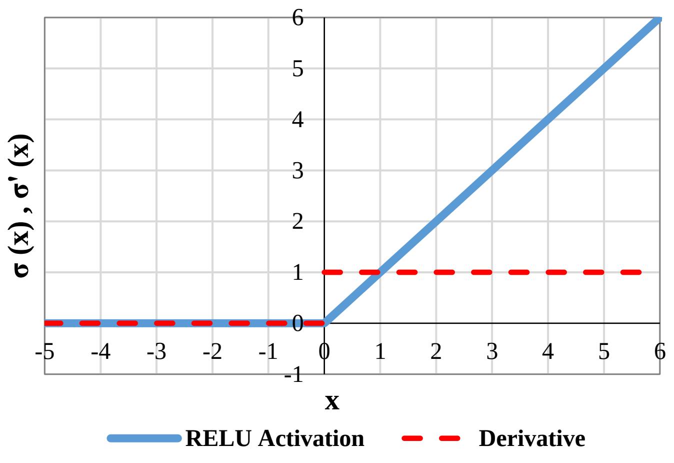
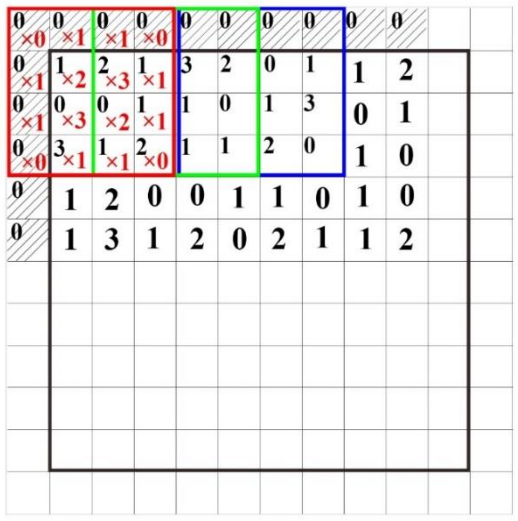
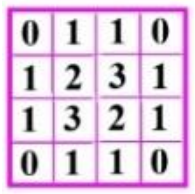


Fig. S5. RELU activation function

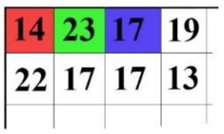
Kernel: In the convolution layer, a matrix multiplication is performed between a small matrix named a kernel (filter) and the matrix of the input of each layer and the resulted summation is transferred to the features' map. Features of an image are simple or complex. For example, in an image containing a cat, there are both simple features like lines and edges, and complex features like whiskers, nose, leather, and tail. Simple features like lines and edges are learned at earlier layers while complex features are learned at deeper layers (Zeiler and Fergus, 2014). The kernel is responsible for the identification of features that distinguish the objects from each other (such as the edges from the lines) in the input of each layer. As shown in Fig. S6, by sliding a 4x4 kernel (the pink matrix in Fig. S6 (b)) through the image and multiplying between them (red, green, and blue matrices), a matrix of features is produced (Fig. S6 (c)), which is the input of the next layer. The large black matrix is one channel of the image or the input of the current layer.



a) Sliding a 4x4 kernel through the image



b) A 4x4 kernel



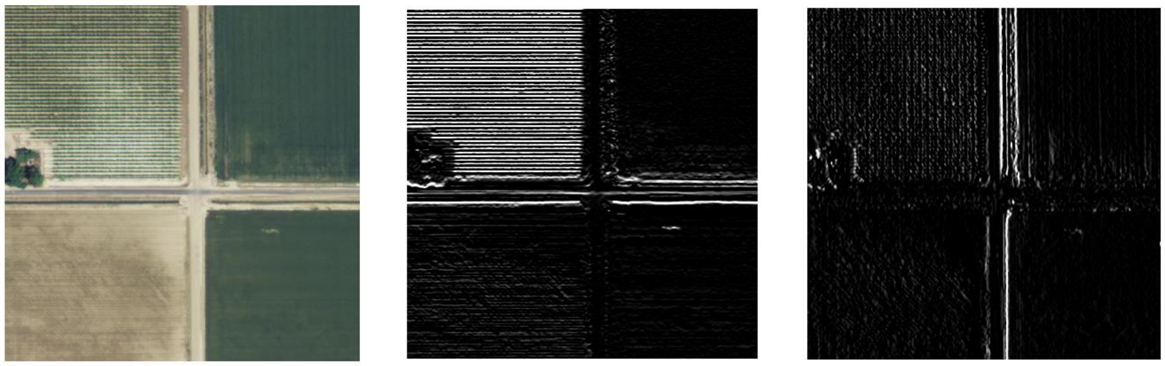
c) Result

Fig. S6. A convolutional layer with a 4x4 kernel

Fig. S7 presents the application of the horizontal and vertical 3x3 kernels (Eq. S16 and Eq. S17) on an image. These kernels are responsible for the extraction of the horizontal and vertical lines from the image and help the model to identify different features.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{Horizontal kernel} \tag{S16}$$

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \text{Vertical kernel} \tag{S17}$$



a) Image b) horizontal kernel c) Vertical kernel

Fig. S7. Extraction of the horizontal and vertical lines using kernels

Max-Pooling: Pooling is a process that will downsample the input while keeping the majority of information. This process will reduce the computation time by decreasing both the image size and the number of parameters of the model and will diminish the over-fitting problem as well (Sharma and Mehra, 2019). The most common form of Pooling is the max-pooling method that applies a max filter to non-overlapping subregions. Fig. S8 shows the application of a max-pooling with a size of 2x2 and a stride size of 2.

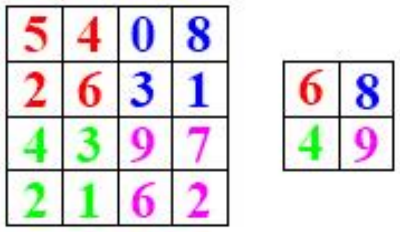


Fig. S8. A max-pooling with a size of 2x2 and a stride size of 2

Fig. S9 shows the application of two max-pooling functions with different sizes (2×2 and 4×4) on an image size of 400×400 pixels. As shown, the original image size will be decreased by 50% and 75%, respectively, while keeping most of the features.

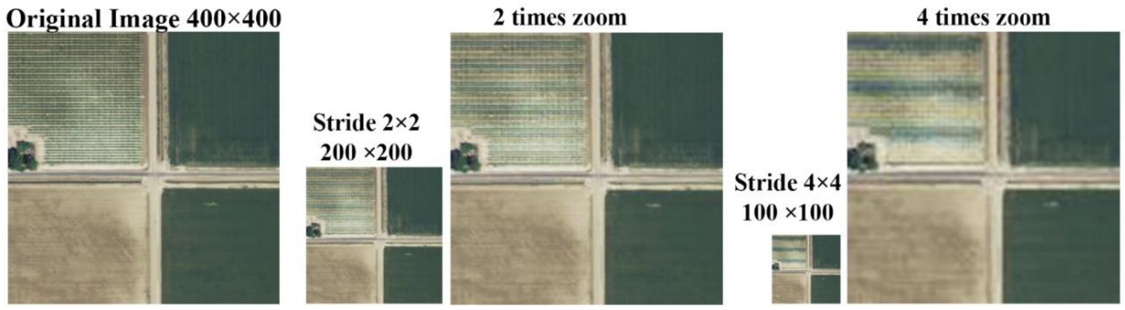


Fig. S9. Application of two max-pooling functions on an image

Learning Rate: The value of the learning rate shows the size of the steps taken to reach a local minimum. In other words, the learning rate determines the percentage of the gradient of the Error (E) with respect to the parameters (weights and biases) that should be applied in the next epoch. In the following equations, W_{t+1} is the weight at iteration *t+1*, η is the learning rate, and g_t is the gradient of the error with respect to W_t. In Table S1, we consider the learning rate to be equal to one, which means the whole value of the gradient is used in the next epoch.

$$W_{t+1} = W_t + \Delta x \tag{S18}$$

$$\Delta x = -\eta g_t \tag{S19}$$

In Fig. S10, arrows show the effect of learning rate value on finding the optimized weights. When the value of the learning rate is too large, the error (E(w)) will fluctuate around the minimum and it cannot converge (red arrows in Fig. S10), or will even diverge. On the other hand, when this value is too small (green arrows in Fig. S10), it will minimize the error (E(w)) continuously but in a slow convergence.

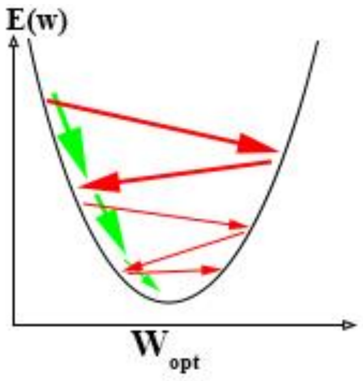


Fig. S10. Effect of learning rate value on finding the optimized weights.

Batch Normalization: Batch normalization is a method that normalizes the input values of each layer so that their mean and variance will be zero and one, respectively. By normalization (Ioffe and Szegedy, 2015):

1. The model accuracy and processing speed will increase (Bjorck et al., 2018).
2. The model can use higher values of learning rate because it will transfer the numbers in a range near zero (Bjorck et al., 2018).
3. The need for dropout and chance of overfitting is reduced (Garbin and Marques, 2020).
4. The model will be less dependent on initial parameters because the subsequent layers will use the normalized values (Bjorck et al., 2018).

Batch normalization will work better with numbers close to zero in some activation functions like Sigmoid and Tanh (as shown in Fig. S2, the change of the activation function is almost neglectable for values more than "5" and less than "-5"). Batch normalization can prevent dying out during training (exploding gradient problem) in the RELU activation function (Bjorck et al., 2018) and can also standardize input variables with different scales because the weights related to some inputs will be modified much faster than others.

Skip Connection: In an ordinary neural network (shallow network), the output of a layer ($f(x)$) is the input of the next layer ($f(x) = x$, Fig. S11). A skip connection will skip multiple layers and add information directly from previous layers ($H(x) = f(x) + x$). Therefore, when $f(x) = 0$, there is still some value and a zero gradient is avoided, as shown in Fig. S11.

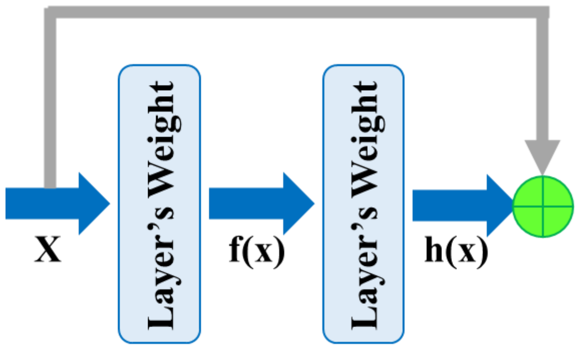
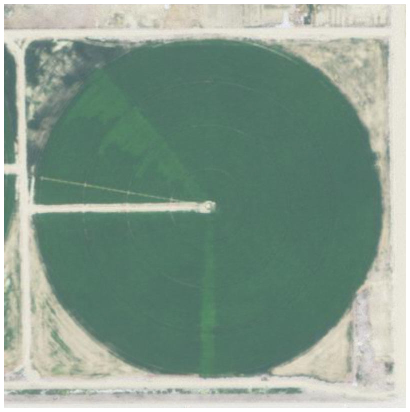
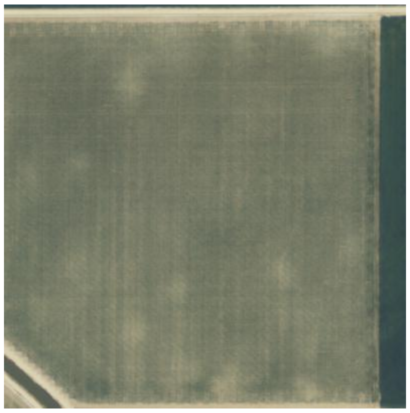
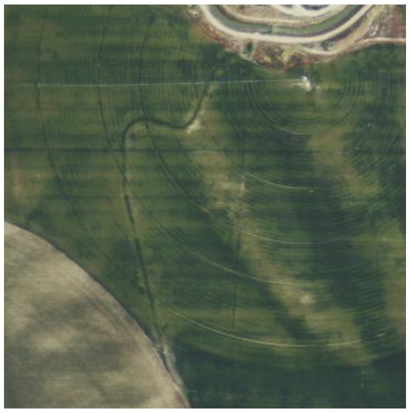


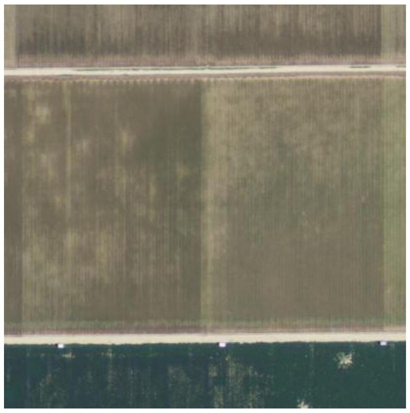
Fig. S11. A skip connection

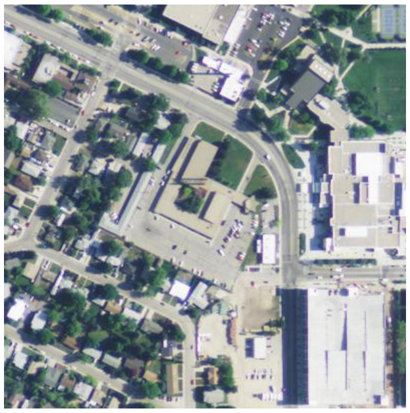
S3. Data and Different Land Use/Cover and Irrigation Type Classes


(a) Center Pivot (CP)


(b) Linear sprinkler Irrigation (LI)


(c) Center Pivot with Swing Arm (CPSA)


(d) furrow Surface Irrigation (SI)


(e) Urban areas (UA)

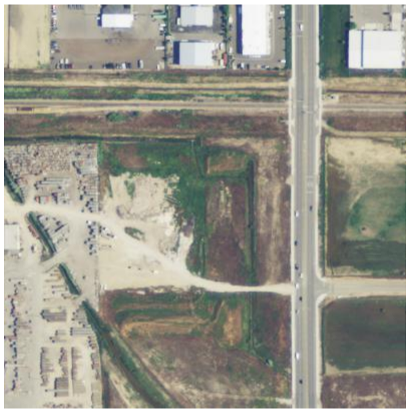

(f) Wildland/Background area (Background)

Fig. S12. One sample from each class in the study area

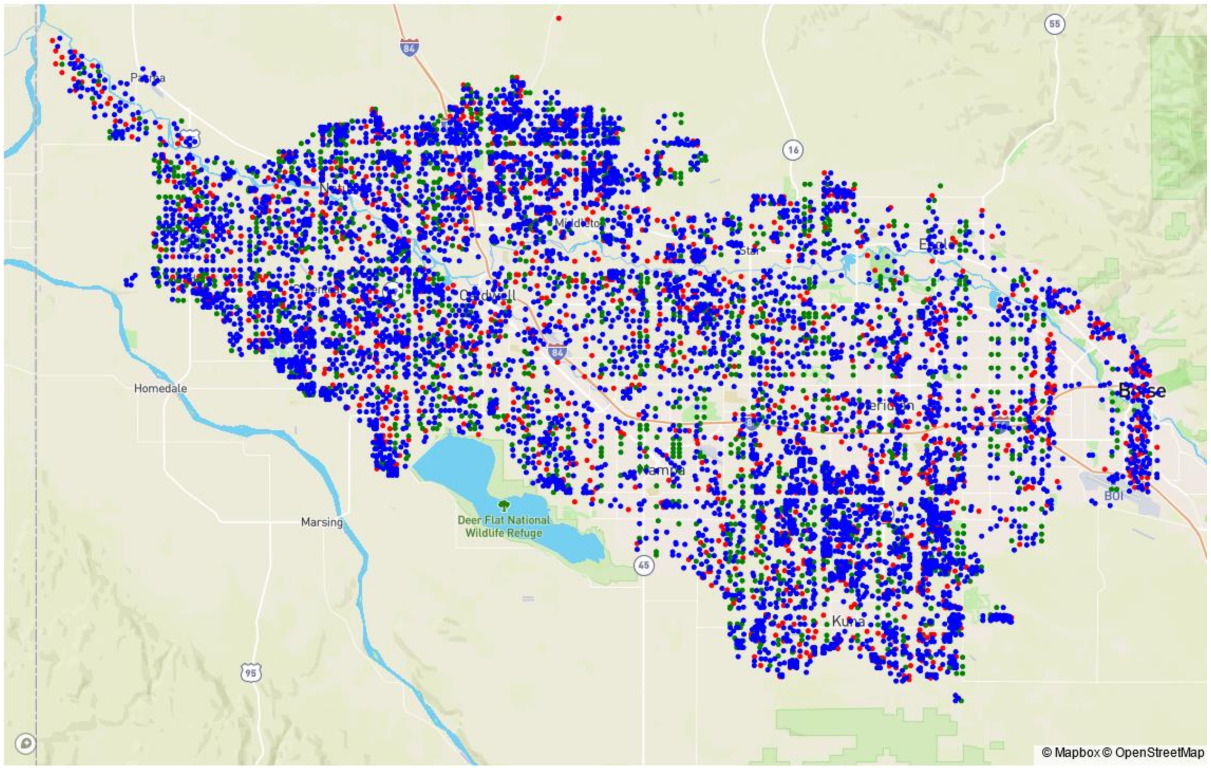


Fig. S13. Distribution of the training data (blue points), the validation data (red points) and the testing data (green points) across the study area

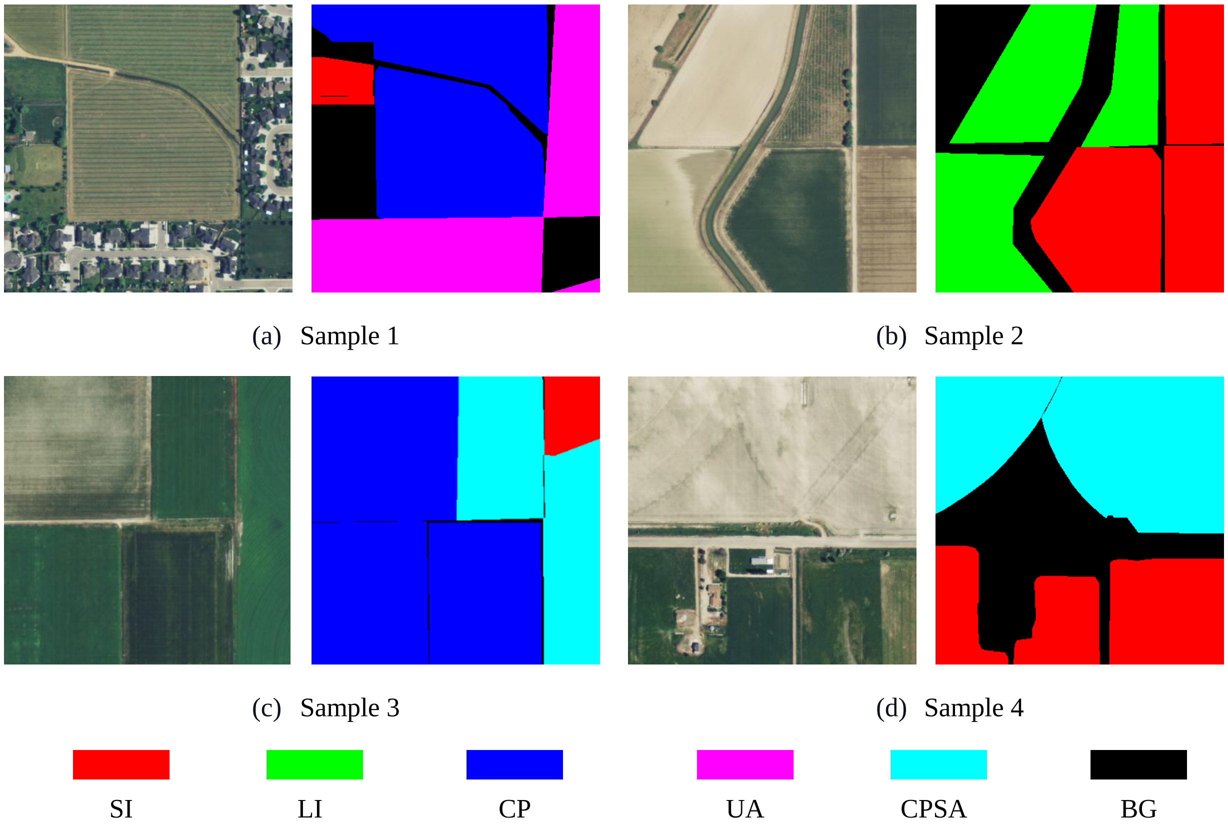


Fig. S14. Generating labels corresponding to different classes using ArcPy

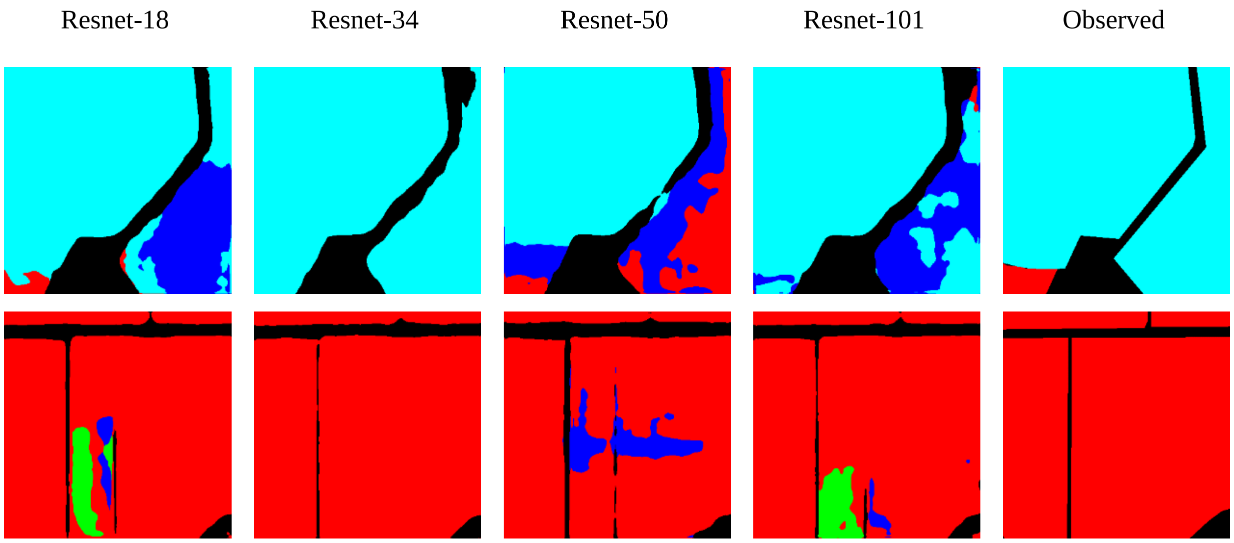
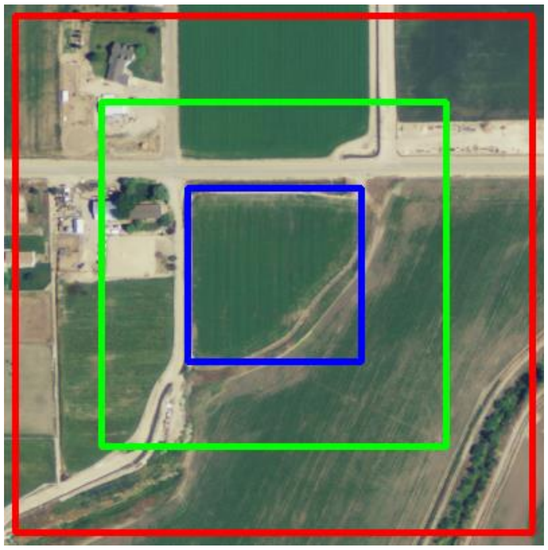
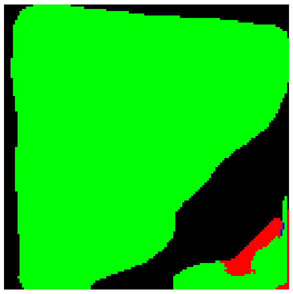


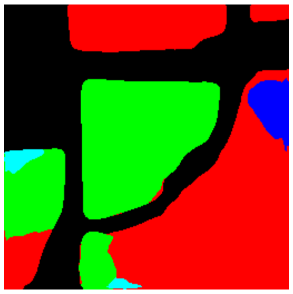
Fig S15. Comparison between different Res-Net structures



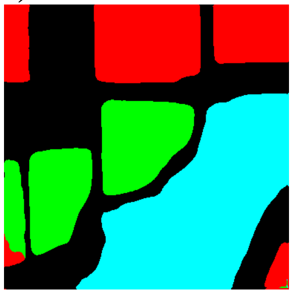
a) True color image-Blue Border (b) 128x128, Green (c) is 256x256 and red (d) 384x384 meters



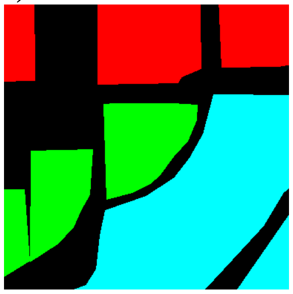
b) Size 128x128



c) Size 256x256



d) Size 384x38



e) Observed (384x384)

Fig S16. the effect of image size on area coverage and model performance.

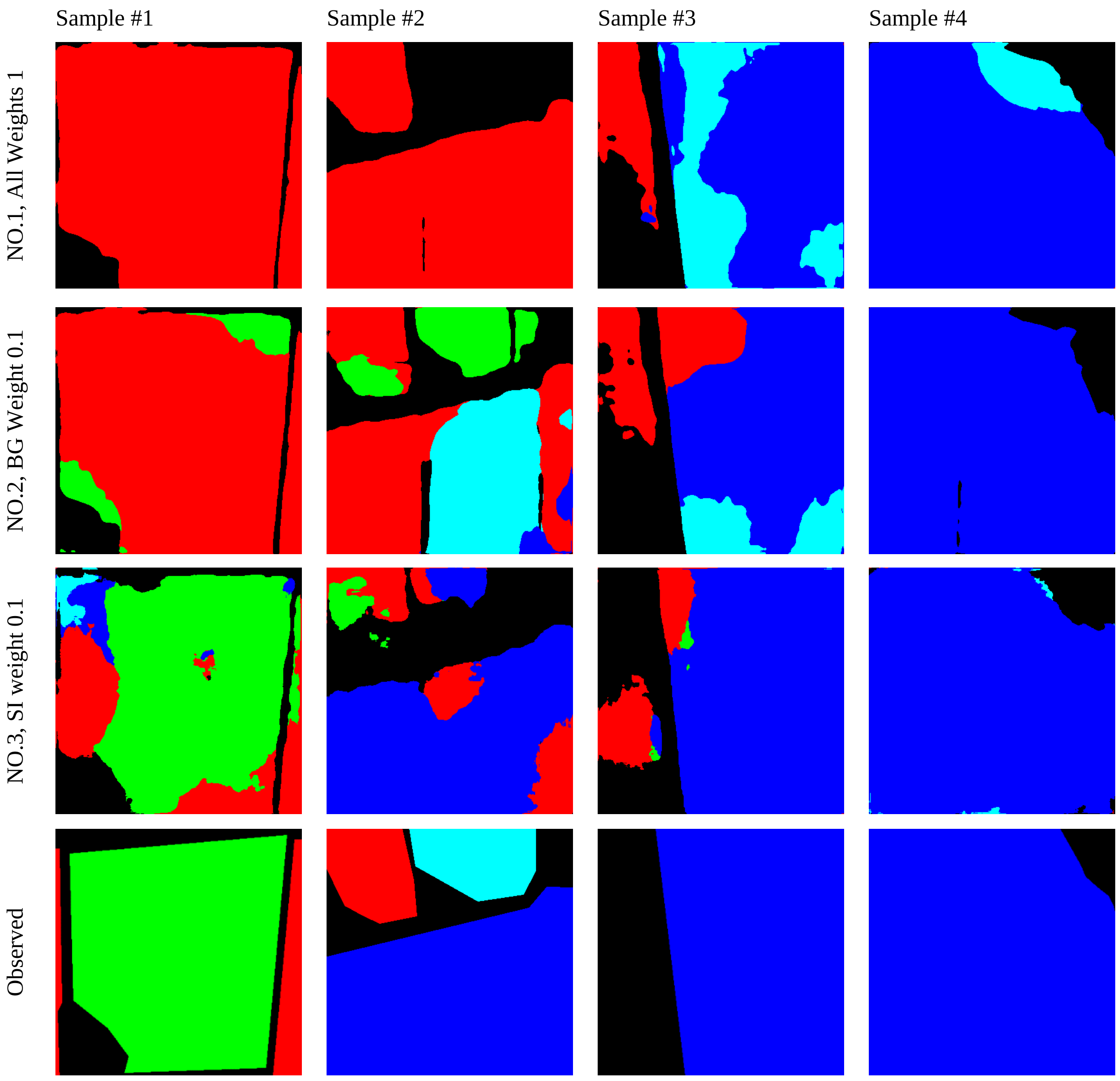


Fig S17. Comparison between different class weights

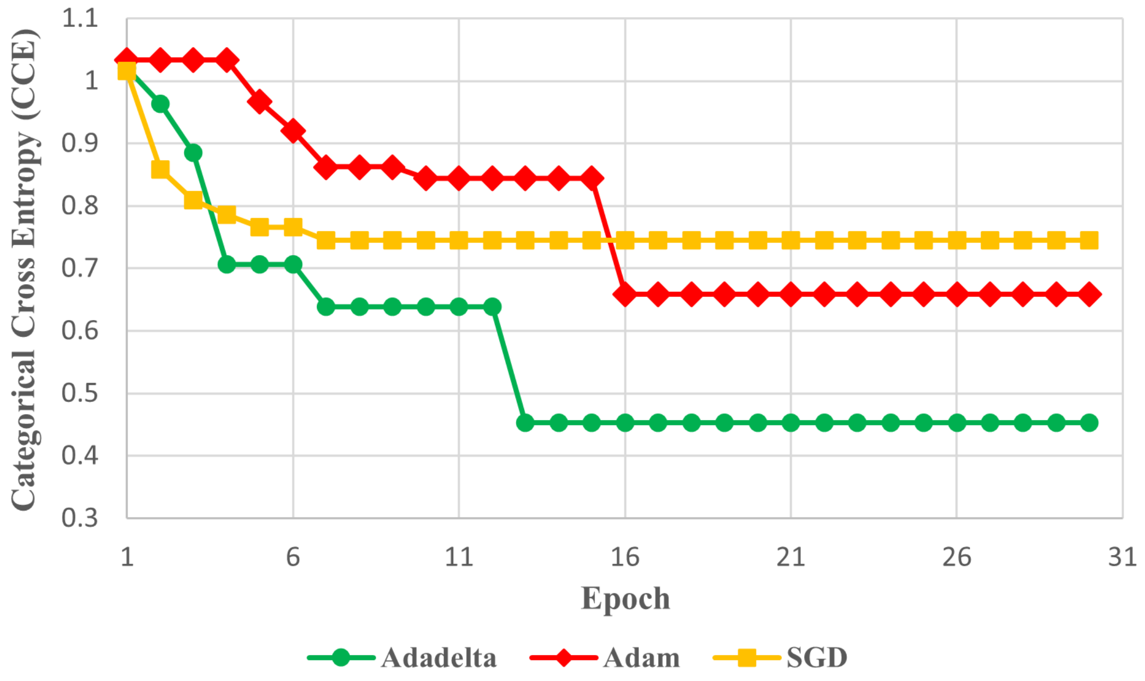


Fig. S18. Evolution of categorical cross-entropy for different gradient-based optimizers

S4. References

Garbin, C., Zhu, X., & Marques, O. (2020). Dropout vs. batch normalization: an empirical study of their impact to deep learning. *Multimedia Tools and Applications*, 1-39.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, p. 2). Cambridge: MIT press [Online]. Available: http://www.deeplearningbook.org.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML* (Vol. 30, No. 1, p. 3)

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *ICML*.

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Sibi, P., Jones, S. A., & Siddarth, P. (2013). Analysis of different activation functions using back propagation neural networks. *Journal of theoretical and applied information technology*, *47*(3), 1264-1268.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.