University of Novi Sad
Faculty of Natural Sciences
Department of Mathematics and Informatics

Stundent
Tamara Krivokuca

Segmentation of Wheat Ears Using Deep Learning Models Based on U-Net Architecture

Master Thesis

Novi Sad, March 2022

I would like to express my sincere gratitude to dr Oskar Marko and PhD student Zeljana Grbovic for their advice, help and support during the writing of this thesis. I would also like to thank professor dr Dusan Jakovetic, dr Sanja Brdar, and all professors who taught me at Faculty of Natural Sciences, each and every one of them contributed greatly to my academic,professional and personal development. Finally, I would not be here without the support of my family and friends, for which I will be forever grateful.

Contents

List of Figures

Project title: Segmentation of Wheat Ears Using Deep Learning Models Based on U-Net Architecture
Student: Tamara Krivokuca
Supervisor: dr Oskar Marko
Novi Sad, 28 February 2022

Abstract

With the rise of human population but limited fertile land for wheat cultivation, farmers need creative solutions for making decisions that will result in best crop yield. One of the techniques farmers use to estimate wheat crop yield is counting wheat ears - spikes atop the wheat plant containing grain - per unit area. Doing this manually requires time and great effort, which is why it makes sense to try to automate this process. One idea for tackling this problem is to train machine learning algorithms to detect wheat ears on images of wheat, and developing methods for counting labeled wheat ears. The purpose of this thesis is explore effectiveness of UNet-based deep learning algorithms for segmentation of wheat ears performed on RGB images, using Python programming language. Dataset is owned by BioSense institute, Novi Sad, and consists of 64 images of grown wheat in a field, in two different stages of maturity, and the same number of images with labels. In total 4 algorithms were tested: UNet, UNet++, ResUNet and ResUNet++. We explain the architecture of these models and how they were developed from each other. We show that ResUNet++ gives the best performance when comparing accuracy, precision, recall, f1 and Jaccard scores of all four trained models applied to unseen data, and provide caparison between scores and runtime. Finally, we investigate and illustrate the process of feature extraction done by the ResUNet++ model, and provide ideas for model improvement and possible wheat ear counting methods.

1 Introduction

Wheat is one of the most important crops in the world. In 2017 alone, 772 million tonnes of wheat was produced, making it the second most-produced cereal after maize [7]. With the growing number of humans on the planet, there is a rising demand for increased food production. More land area is devoted to wheat cultivation than any other crop, but further area expansion is limited [6]. Therefore, in order to improve production in the future, farmers will have to rely on research for further improving cultivars and enhancing cultural technology, in order to get a greater **output per area**. One of the most popular ways to predict this output is to count the number of wheat ears - spikes atop the wheat plant containing grain - per unit ground area, i.e. to measure **ear density**. Using this metric, farmers are able to better monitor the efficiency of their crop management practices and establish an early prediction of grain yield. Traditionally, this metric was measured by a field worker, who would isolate a unit area in a field and count the ears manually [8]. With larger fields, this procedure would have to be repeated multiple times in order to get more accurate measurements. This method is time consuming and tedious, it is prone to human error, and becomes increasingly difficult to practice as field size grows.

Figure 1: Farmer isolating wheat ears which will be used for yield estimation in Punjab, Pakistan. Source [1].

With the rapid growth in technology over the years came great advancements in both image capturing technology and image processing algorithms. At the time of writing of this thesis, even mid-range smartphones have cameras which produce great quality pictures, and mobile phone manufacturers are constantly working on improving them. Seeing as smartphones have become a necessity in this day and age, it is safe to assume that most farmers own or have access to these devices. It is worth

mentioning that devices and gadgets like camera drones and smartphone lenses are becoming more popular and affordable. On the other hand, there has been a plethora of research on deep learning technology for image analysis, which demonstrated great success in locating and isolating objects in images. In digital image processing, this process is called **segmentation**, and is defined as a process of partitioning images into multiple segments by assigning a label to every pixel in an image so that pixels with the same label share certain characteristics (Figure 2). Technologies for image segmentation have been used successfully in many real life settings, such as self driving cars, face recognition, medical imaging, etc. With all this in mind, it is easy to see that manual counting of wheat ears can be automated so that farmers can simply take pictures of patches of their field, and get the number of wheat ears instantly, which would not only speed up the process of acquiring ear density measurement, but could also significantly reduce the incidence of errors in measurements.

Figure 2: Image of wheat (left) and segmented image (right). Blue parts indicate pixels where the wheat ears are located. Labels were made by hand, and the goal is to develop an algorithm that will be able to do that on its own.

Before deep learning methods were available, digital image segmentation and classification were performed using what is now called "classical methods". Segmentation methods applied edge detection, region detection and thresholding operations (among others), combined with morphological operations. Classification employed traditional statistical and ML techniques, such as Nearest Neighbour and Bayesian classification [10]. However, with the rapid development of deep learning techniques, image classification and segmentation approaches changed. This was due to the outstanding performance of Convolutional Neural Networks (CNNs) in different high level computer vision tasks. There are many advantages of applying deep learning to image segmentation. To give an outstanding performance, deep learning only needs data and it does not need any traditional handcrafted feature engineering techniques. Also, traditional machine learning algorithm cannot adjust itself for a wrong prediction, as opposed to

deep learning algorithms which have that capability. One of the first deep learning models developed for this problem was FCN (Fully Convolutional Network) [22]. As the name suggests, it includes only convolutional layers, which enables it to take an image of arbitrary size and produce a segmentation map of the same size (Figure 3). It inspired a lot of more complex models, combining it with different approaches such as the ones used in our research, which use encoder-decoder architecture. Other models used combinations of CNNs with CRFs (Conditional Random Fields), R-CNNs(Regional CNNs), RNNs(Recurrent Neural Networks), GANs(Generative Adversarial Networks), etc. Due to the growing complexity and variability of architectures used for image segmentation, it would be difficult to go over all possible approaches in this section. Detailed survey based on network architecture is provided in [23], and yet another based on specific task is given in [31].

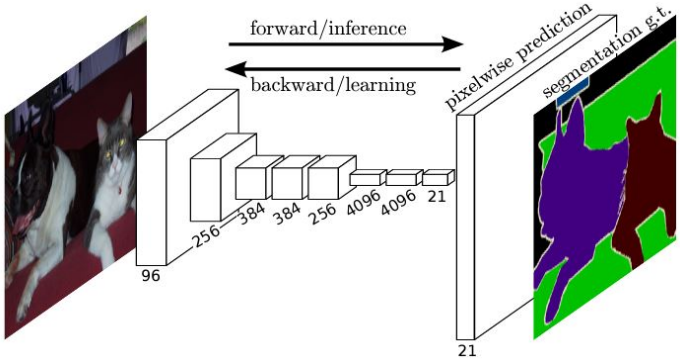


Figure 3: Architecture of FCN. Source [23].

Methods specifically for isolation of wheat ears from images have been developing for over a decade. This idea was first introduced in [5] in 2008, where authors used classical methods of image segmentation and classification, combined with morphological information about wheat ears to get the wheat ear count. In [8] a combination of Laplacian frequency filter and a Median filter was used to remove low and high frequency elements appearing in an image after which segmentation was performed using Find Maxima segmentation technique, where ears detection was determined by local peaks found within the image. In [3], authors applied a Gabor filter, PCA for dimensionality reduction, and K-means clustering for classification. Another example of application of classical methods comes from [36], where TWSVM algorithm was applied in combined with feature extraction. Some of these methods used RGB images, while others opted for their greyscale versions. While the greyscale route makes sense for their approach, converting images to greyscale eliminates important color information that helps distinguish ears from the rest.

Moving on to the deep learning era, there are quite a few interesting approaches.

Pound et al. used a network based upon a stacked hourglass network, which itself is an evolution of FCNs and residual networks, in [27]. In their work they also provided a new dataset they named ACID (Annotated Crop Image Dataset), which is publically-available and contains 520 images of wheat plants exhibiting a wide range of canopy and spike phenotypes. In another work [13], Hasan et al. used R-CNN achitecture combined with a pre-trained CNN network, with great success (average F1-score was 0.95). They also contributed with a new dataset, named SPIKE, of 300 labeled images at 3 different growth stages. In [12], Grbovic et al. used CNNs on both RGB and thermal images. Due to compact shape, ears have a greater thermal capacity than the rest of the plant, which is why they appear very bright in thermal images and can be easily distinguished from the background, even by human eye. However, as it is explained in [8], using thermal images has limitations because of low resolution and a high cost of thermal cameras. Recently, in [34] Wang et al. proposed an improved EfficientDet-D0 object detection model for wheat ear counting, and focuses on solving occlusion (when two or more wheat ears come too close and seemingly merge or combine with each other), by adding a random cutout method where some rectangles are selected and erased according to the number and size of the wheat ears in the images to simulate occlusion in real wheat images, forcing the algorithm to pay more attention to wheat ears. Sadeghi-Tehran et al. combine U-Net with SLIC (simple linear iterative clustering) method in [29]. They also feed the network with two types of patches of wheat images, one representing the ears and other the background. This paper demonstrated the great potential U-Net has for tackling this issue. Another great approach is given in [19], where authors developed WheatNet, which uses a truncated MobileNetV2 and two parallel sub-networks for simultaneous density-based counting and localization which mutually strengthen each other for improving prediction accuracy.

kurtis je pisao o leba u [6]

The main goal of this thesis is to propose a method for isolating locations of wheat ears on a given image of wheat in outdoor conditions. With an algorithm that is precise enough, and further development of techniques for counting wheat ears after they have been located, this could prove to be a very useful tool for farmers. The thesis paper is divided into five sections, including this one. In section 2 we will explain experimental setup, and it is divided into two parts. Section 2.1 will cover basic information about the dataset, and explain the data acquisition process, preprocessing techniques and practical implementation of the data pipeline. In section 2.2 we will explore basic deep learning concepts and describe four algorithms that were used for the purpose of wheat ear detection: Unet [28], Unet++ [37], Deep ResUnet (in further text denoted only by ResUnet) [35] and ResUnet++ [18], as well as details about their implementation. Here we will also describe the metrics used for evaluating the performance of our models. With all technical information covered,

in section 3 we will show the results of our work. First we will explain the network training process, and illustrate post-processing techniques used to get the best possible outcome from the models, and then we will provide concrete numerical results, with visual demonstrations. Section 4 is dedicated for discussion of the results, as well as providing ideas for further research. Finally, in section 5 we will recapitulate the entire project and give our final thoughts on the subject.

2 Materials and Methods

2.1 Data

The dataset we used is comprised of two parts: one part is a set of RGB images in JPG format of wheat in outdoor conditions, and the other is a set of PNG images of labeled locations of wheat ears. Images contain a white frame of size $0.5m \times 0.5m$, that guarantees the observed area within the frame will be representative for the measurement (Figure 4a). Labels are provided only for wheat ears within the white frame (Figure 4b). On each label image, there are white strokes on the locations of wheat ears, while the rest is black. Original resolution of images is 4036×3024.

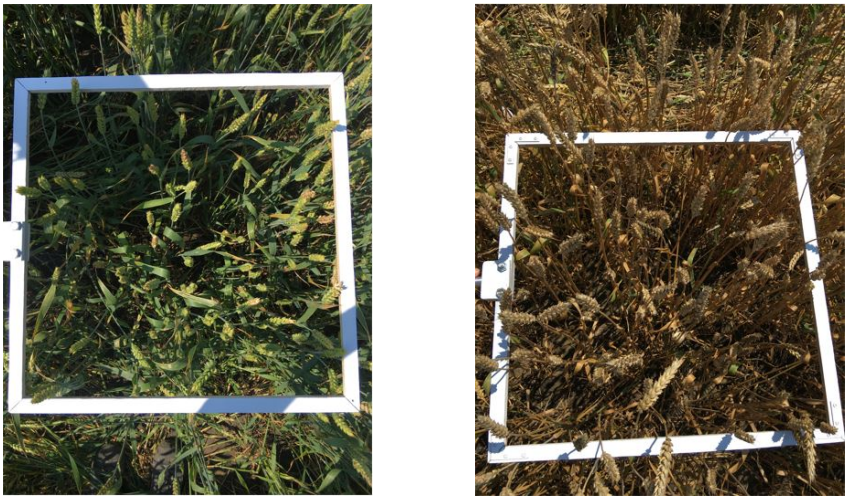
(a) One image from the dataset.

(b) Label image for image 4a

Figure 4: A pair image/label image from the dataset.

2.1.1 Data Source

Images were provided by the BioSense institute from Novi Sad, Serbia. Researchers went on a field in Ravno Selo, a village located approximately 25km northeast from Novi Sad, to gather the data. They went on two separate dates: 10/06/2019 when wheat blooming period was ending and it was starting to mature, and 26.06.2019. when the wheat was mature and harvest ready (Figure 5). Images captured on the first date showed green wheat ears, grass and leaves, while the color of ones taken on the second date was yellow/orange. It was important to get images form different stages of wheat development to make sure algorithms will be able to recognise wheat ears regardless of the color of wheat. Time and date of capturing images were carefully chosen to avoid clouds and shadows. For the experiment they used an iPhone 8 (with 12 MP camera, f/1.8 aperture, digital image stabilization, optical image stabilization). In order to capture the wanted area within the frame, images were taken approximately 0.5m height above the crop. BioSense provided almost 400 images, however only 64 were used for this project. This was due to the fact that in order to get better results, images needed to be relabeled more precisely, and this is very time consuming. Another reason for this is hardware limitations - using all images would prolong the training time significantly, and considering the fact that variability of images could be introduced through image enhancement (flipping, rotation, change of brightness), it made sense to choose a smaller subset of images.



(a) Image taken on 10/06/2019 (b) Image taken on 26/06/2019

Figure 5: Difference between images taken on separate dates.

2.1.2 Data Preprocessing

Main preprocessing steps included (Figure 6):

- **Frame expansion**. Because of the way wheat ears are counted, only those inside the frame needed to be isolated, disregarding all ears outside of the frame. This was achieved by developing a simple algorithm for detection of the frame, and automatically making all pixels outside of the frame white. This accomplished two things: first, undesirable wheat ears were being discarded, and second, frames were being naturally expanded, not adding any redundant information. As for why this was necessary, images were taken at an angle, which means that the frame did not form a perfect square in the image, making it impossible to simply crop the pictures around the frame without losing or adding wheat ears. We should note that some of the images had too many wheat ears going over the frame for it to be detected successfully in this manner. Those images were processed manually.

- **Cropping**. Next step in the preprocessing was to crop images so that the main focus of the resulting image was the wheat. The goal was also to crop images into a square shape, to simplify further processing. Since image size is 4032x3024, and frames on most images go from the left edge to the right (along the shorter dimension), images were cropped into size 3024x3024. This process was automated by loading images into Python, detecting the first non-white pixel from the top of the image, noting its y coordinate, and then cropping from starting pixel $(0, y - 100)$ if possible, otherwise starting from $(0, 0)$, with desired crop size. Again, some of the images had to be cropped manually, because of the way the images were taken.

- **Resizing**. Even though the second step reduced the size of the images, their size would still make further processing slower, and because of the good quality of images, they could still be reduced in size without significant loss of information. It was also beneficial to make the size a degree of 2, to simplify the following steps. For these reasons, we resized the the images to 2048×2048 resolution.

Final dataset consisted of 64 2048×2048 RGB images and 64 labels.

2.1.3 Data Pipeline

For the process of loading and preparing images for network training two Python libraries were used: OpenCV and PyTorch. OpenCV [4] is a library of programming functions mainly aimed at real-time computer vision. It was built to provide a common infrastructure for computer vision applications and to accelerate the use

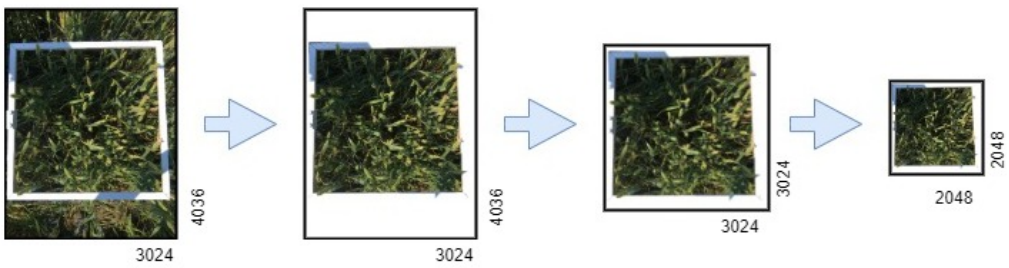


Figure 6: Preprocessing steps.

of machine perception in the commercial products. It has a rich collection of methods for transforming and improving images. PyTorch [25] is an open source machine learning library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab. It provides methods for generating data pipelines, neural network architectures, network training, etc.

After loading the images, they were shuffled and divided into training, validation and test datasets. Traditionally in machine learning, training set is used for the training process, to adjust the network *weights and biases* based on this data alone. Validation set is also used during training, to check how the network is performing on unseen data, and to tune network *hyperparameters* to improve the results. This can also skew the network towards performing better on this dataset than it actually would on unseen data, which is why there is a test dataset, which is used to check the network performance after all parameters and hyperparameters are fixed [11]. There is no fixed rule on how to determine data split ratio. A general rule of thumb is to dedicate most of the dataset to training, and then leave equal smaller sizes for validation and testing. We used a 0.6 / 0.2 / 0.2 ratio, i.e. 38 images for training, and 13 each for validation and testing.

In order to prevent the code for data loading and processing from becoming messy, we used PyTorch's *Dataset* and *DataLoader* modules. *Dataset* stores the images and their corresponding labels, and *DataLoader* wraps an iterable around the Dataset to enable easy access to the samples. We made our own class titled *WheatDataset* built on PyTorch's *Dataset*, adding methods for image augmentation. In order to help our networks perform better, the following augmentation techniques were applied on the images and the labels:

- **Histogram equalization**. Image histogram is a histogram representing the distribution of intensity values of a digital image. Histogram equalization is the process of transforming its histogram, so that all intensity values have similar incidence in the image. It is essentially taking an image and increasing the contrast between the image's relative highs and lows in order to bring out subtle

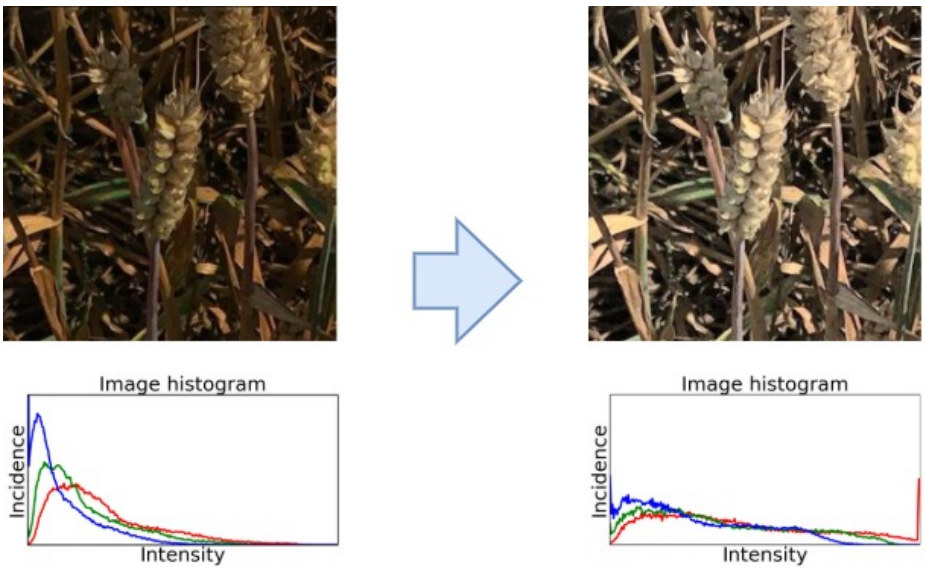


Figure 7: Image histogram equalization illustrated on a part of an image of wheat.

differences in shade and create a higher contrast image [10]. The process is straightforward with greyscale images. With color images (which consist of three channels - red, green, and blue), simply equalizing every channel does not result in desired outcome. Instead, RGB images are converted to another color space containing the luminance or brightness channel, which is equalized and then the entire image is converted back to RGB. In the case of our dataset, there are a lot of green or yellow wheat ears with similar colors to leaves, stems or grass around them. After equalizing, it was visibly easier to discern wheat ears from the rest (Figure 7).

- **Image flipping and rotation**. A common technique for enriching the dataset is to perform random flip or rotation operations. With flipping, images are mapped symmetrically either along x axis, or y axis or both, with given random chance. With rotations, images are rotated around their center by the random angle. This both provides more images to train on, but can also help expose our networks to a wider variety of possible positions of wheat, which makes the algorithms more robust. Both images and labels are mapped with the same transformation, ensuring everything remains labeled properly (Figure 8).

- **Label blurring**. Because the labels were made manually by humans, and because humans cannot tell the precise location of the edge of the wheat, every

Figure 8: Image flipping and rotating.

label of a wheat ear probably captured parts of the background as well[1]. Labels are binary (1 where wheat ears are present and 0 where they are not), which means that when they get passed to the algorithm, the algorithm will get the message "Locations of white segments of the labels give a 100% chance of existence of wheat ear on those pixel locations on the image". This is not desirable, since there could be objects near the edges of the ears that algorithm could mislabel (e.g. it could learn that leaves are also wheat ears). To avoid this, label images were transformed using the Gaussian Blur (Figure 9). Gaussian blur is the result of blurring an image by convolving it with a Gaussian kernel. This kernel is a matrix obtained by sampling a 2D Gaussian function so that the resulting matrix has the center element with the highest value of the Gaussian. Performing Gaussian blur resulted in labels having blurry edges but keeping the core white, giving the algorithm a chance to be unsure along the edges of the wheat. This idea is a slightly modified approach from [32], where the authors described introducing noise to labels as a regularization technique.

We used OpenCV functions for the implementation of these transformations, and they were added as methods in *WheatDataset* class. For the training of the networks, images needed to be divided into smaller pieces, due to memory limitations [2]. For this reason, an additional method was added, dividing the images and labels. Every image and label was divided into 64 smaller images of size 256x256, resulting in a set of total of 4096 smaller images, i.e. 2432 images for training and 832 each for

[1]It is very difficult to label these images precisely in a short amount of time. Even after relabeling, there were still errors that could affect the training.

[2]This issue will be addressed in detail in chapter 2.2

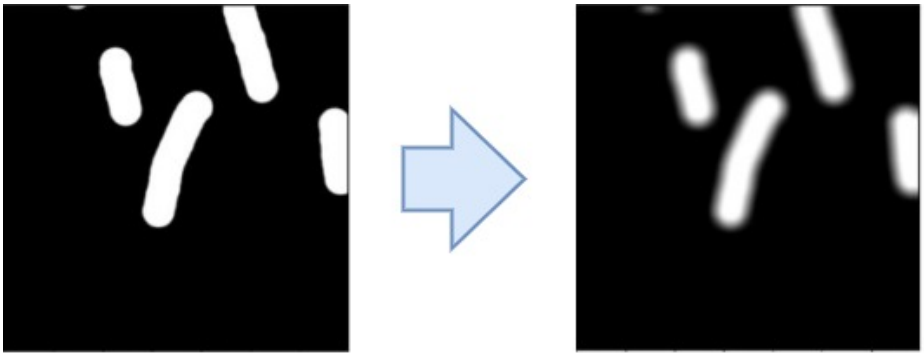


Figure 9: Blurring the labels

validation and testing[3]. Data is then loaded into the *DataLoader*, which divides it into batches of predefined size. This batch size is another hyperparameter that can be tuned during training. This is also where random flip and random rotation are performed. The entire pipeline is illustrated in Figure 10.

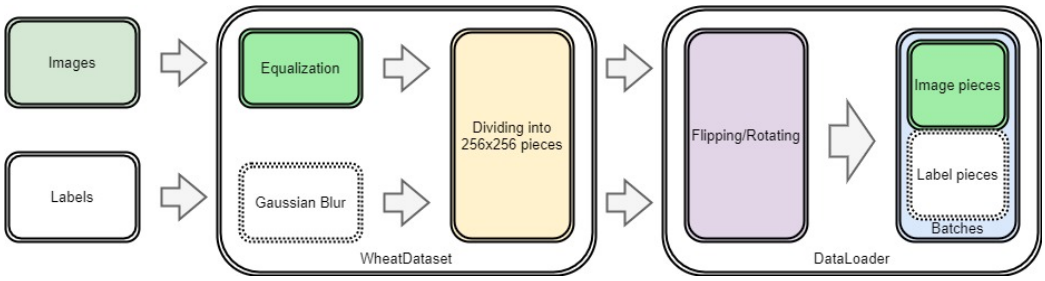


Figure 10: Data pipeline.

2.2 Methods

Image segmentation is a pixel-level vision task, which aims to extract meaningful information for easier analysis. In these tasks, the image pixels are labeled in such a way that every pixel in an image shares certain characteristics such as color, intensity, texture, etc. with pixels from the same class. Mainly, image segmentation tasks can be divided into two types: semantic segmentation and instance segmentation [31]. Semantic segmentation is defined as simply assigning a class label to each pixel in an image. For example, in an image of traffic, semantic segmentation task would be to label cars with one label, the road with a second label, people with third, and so on (Figure 11 (b)). In contrast, with instance segmentation the goal

[3]This is without counting amount of images added by enrichment

is to detect each object and delineate it with a bounding box or segmentation mask, respectively. In the previous example, in an instance segmentation task cars would all be labeled with separate labels (Figure 11 (c)). Recently, another type called panoptic segmentation [20] has become popular, which is the unified version of two basic segmentation processes (Figure 11 (d)).

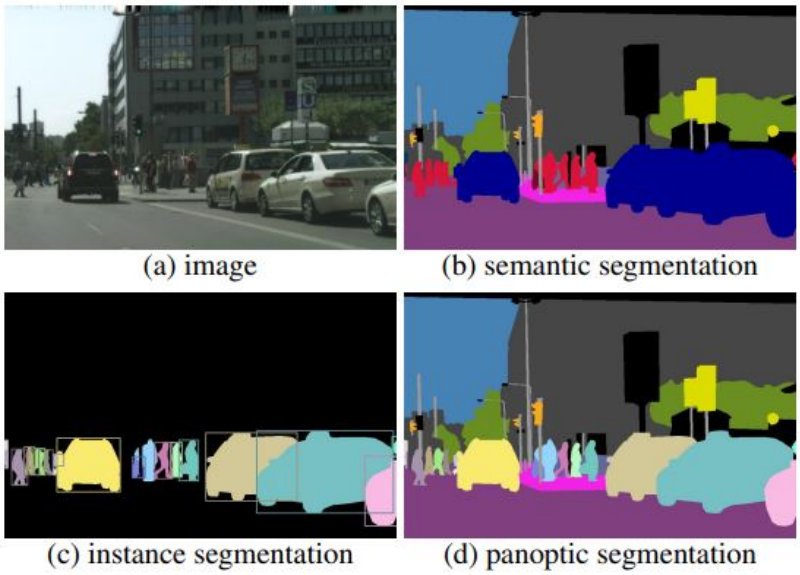


Figure 11: Types of image segmentation tasks. Source [20].

Because of the nature of the dataset and labels provided, we opted for semantic segmentation approach. However, there are image processing techniques for isolating separate instances from labels obtained in this manner, which will be discussed in section 4.2, that could be applied for the task of counting the wheat ears.

2.2.1 Neural Networks

Artificial Neural Networks (ANNs, or just NNs) are machine learning models whose design was inspired by neuroscience. Their purpose, like with other ML algorithms, is to approximate some function f^*, that maps an input of data X into an output $y = f^*(X)$, with a function $f(X;\theta)$ by *learning* the value of parameters θ which minimizes some predefined *loss function* that measures how similar functions f and f^* are. These algorithms are called networks because they are typically represented by composing together many different functions. The model is associated with a directed acyclic graph describing how the functions are composed together (Chapter 6 in [11]) (Figure 12). These functions are called *layers*, and in most neural networks there are 3 types of layers: input layer (first layer, usualy consists of a sample of

our data), output layer (final layer, function that results with approximation of the output data y), and hidden layers(everything in between). Number of hidden layers indicates the *depth* of the model, which is the reason for the popular "deep learning" terminology. All layers in the network are vector-valued, and each element of the vector may be interpreted as playing a role analogous to a neuron, in the sense that it receives input from many other units and computes its own activation value. This is why the networks are called "neural". These algorithms could be used for both unsupervised and supervised tasks, however we are only interested in the latter, and from now on will only be talking about NNs designed for these tasks.

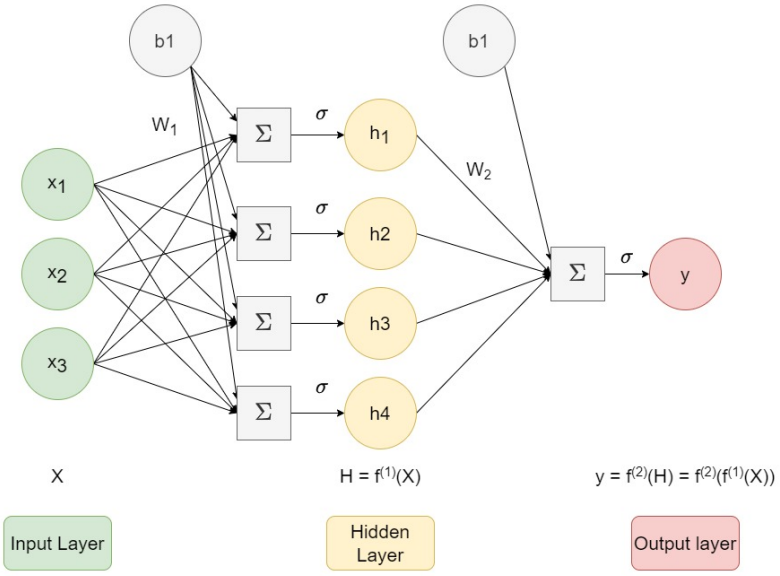


Figure 12: Schematic illustration of a simple neural network.

If information in the network flows from X through the hidden layers and to the output y, Neural Networks are called *feedforward* neural networks - there is no feedback connections in which outputs of the model are fed back into itself. When feedback connections are included, these are called *recurrent* neural networks. This flow is usually achieved by some operation like matrix multiplication or convolution, followed by optional addition of bias term, and passed through an *activation function*. Activation functions are used in order to introduce non linearity to NNs, enabling them to model more complex relationships. These functions help decide which "neurons" are useful and which ones are not - they achieve this usually by giving a near-zero value to the less useful neurons, and higher values to more useful ones. Some of the most commonly used activation functions are rectified linear unit (ReLU) and its variants, sigmoid, hyperbolic tangent, softmax, etc.

To illustrate how a NN algorithm works, let us look at Figure 12. In the input layer, we have our data vector $X = [x_1, x_2, x_3]^T$. This vector gets multiplied by a weight matrix W_1, summed with a bias term b_1 and passed through an activation function σ_1 to get the values of the hidden layer $H = [h_1, h_2, h_3, h_4]^T$. In vector form, this process can be written as:

$$H = f^{(1)}(X) = \sigma_1(W_1 X + b_1).$$

In order to get the output y, H is multiplied by another weight matrix W_2, summed with a new bias term b_2 and passed through another activation function σ_2. So, in the end we have:

$$y = f(X) = f^{(2)}(H) = \sigma_2(W_2 H + b_2) = \sigma_2(W_2(\sigma_1(W_1 X + b_1)) + b_2)$$

Parameters $\theta = (W_1, W_2, b_1, b_2)$ are the ones that need to be adjusted in order to get a good approximation of f^*. The values set for these parameters initially can determine how well the algorithm is able to perform. For feedforward neural networks, it is important to initialize all weights to small random values. The biases may be initialized to zero or to small positive values (Chapter 6.2 in [11]).

In order to obtain the values for parameters θ resulting in the best approximation of function $f^*(X) \approx f(X; \theta)$, NN goes through a *training* process. During the process, values of θ are adjusted usually by using iterative, gradient-based optimizers that drive the loss function to a very low value. In order to compute the gradient, NNs *back-propagate* the information on the loss function from the output layer to the input layer using the well-known gradient chain rule. Then the learning is performed using the optimizer. Gradient-based optimizers add the negative gradients scaled by a parameter α (called *learning rate*) to the current values of the network parameters, possibly with some additional calculations. Some of the most commonly used optimizers are: stochastic gradient descent (SGD), SGD with momentum, AdaGrad, RMSProp and Adam (Chapter 8 in [11].)

In order to be able to test how the network performs, the dataset is split into two subsets: training and test set. Additionally, another subset called validation set may be isolated, to help with the training process. During the training, loss is calculated on subsets of the training set called *batches* (or minibatches). Essentially, batches of equal size are sampled uniformly from the training set, one by one they go through the network, and with each pass loss is calculated between the output of the network and original labels. One passing of all data is called an *epoch*, and the length of the learning process is measured both in the time needed to complete the training and the number of epochs needed to achieve good results.

Sometimes the algorithm performs much better on the training set than on the validation/test set (overfitting), and to avoid this, techniques commonly referred to as

regularization techniques are applied. Here are some popular regularization technique (and the ones we used in our research):

- **Early stopping**. It is one of the simplest regularization techniques. It is common that the algorithm starts overfitting only after a certain number of epochs have passed. With early stopping, we simply stop the training process before this occurs, by setting an early stopping criterion - how many epochs will we allow the network to continue learning without improving validation loss.
- **Dropout**. This approach randomly "turns off" a subset of neurons during training. This prevents units from co-adapting too much. It was proven to be a quite effective regularization technique, that also reduces training time [30].
- **Data augmentation**. This technique is very useful with image recognition algorithms, since it provides a simple way to increase the number of data points (images) for the training, making it easier for the algorithm to generalise to unseen data. We already mentioned some of the techniques we used: image rotation, image flipping, blurring labels, equalization, etc.

While the training set is used to tune the parameters of the network, validation set is used to tune the hyper-parameters - predetermined variables that affect the training process. Some examples of hyper-parameters include number of epochs, batch size, learning rate, early stopping criterion, dropout rate, etc. After obtaining satisfactory results on the validation set, the entire network can be re-trained using both training and validation sets (this step is not necessary, but could be beneficial). Final analysis is then performed on the test set, which the model has never seen before.

This subsection covered all basic factors required for understanding NNs and their training process. There are many different NN models that are used for different purposes, and that apply different optimization techniques, neuron layout, operations on neurons, loss functions, activation functions etc. The focus of this paper is the use of CNNs for image segmentation, which is our next topic.

2.2.2 Convolutional Neural Networks

Among different deep learning algorithms, CNNs got tremendous success in different fields of computer vision as well as the area of image segmentation. The original concept was proposed by Fukushima in 1980 [9], and was significantly improved upon by LeCun et al [21] in 1998. At the core of these algorithms is the operation of *convolution* - an operation that expresses the amount of overlap of one function g as it is shifted over another function f. In the case when the functions are discrete and two-dimensional (i.e. when we are working with 2d images), if we denote the first

function (often called *kernel*) with K and the second with I (for "image"), it can be defined as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n),$$

where i and j indicate the coordinates of the image pixels. CNNs use this operation in place of the traditional matrix multiplication. They mainly consist of three types of layers [23]:

1. **Convolutional layer**. In this layer, a kernel (or filter) of weights is convolved with the input in order to extract features from the image; these kernels are matrices that are usually small in size (3×3, 5×5 or 7×7, although they do not have to be square matrices) and they can be randomly generated or predefined when creating the network; often another matrix is added to the convolution as a bias term; these weights and biases are modified during training to improve the performance of the network;

2. **Nonlinear layers**. which apply an activation function on feature maps (usually element wise) in order to enable the modeling of non-linear functions by the network; among the most commonly used activation functions are sigmoid, hyperbolic tangent, ReLU (rectified linear unit) and its variants and softmax [24];

3. **Pooling layers**. which replace a small neighborhood of a feature map with some statistical information (mean, max, etc.) about the neighborhood and reduce spatial resolution; neighbourhood is defined by pool size parameter, which determines the shape and size of the sliding window matrix which will gather the local information from the image.

In Figure 13 we can see how simple convolution layer with 3×3 convolution kernel and 2x2 maxpooling function acts when applied to our wheat image. The units in layers are locally connected, that is, each unit receives weighted inputs from a small neighborhood, known as the receptive field, of units in the previous layer. By stacking layers to form multi-resolution pyramids, the higher-level layers learn features from increasingly wider receptive fields (Figure 14). The main computational advantage of CNNs is that all the receptive fields in a layer share weights, resulting in a significantly smaller number of parameters than fully-connected neural networks. Another useful advantage of CNNs is *translational equivariance*, which is a property that assures that the position of the object in the image does not affect its ability to be detected by the CNN. This simply means that if the input changes, the output also changes. This is useful for our case since wheat ears appear in different positions throughout

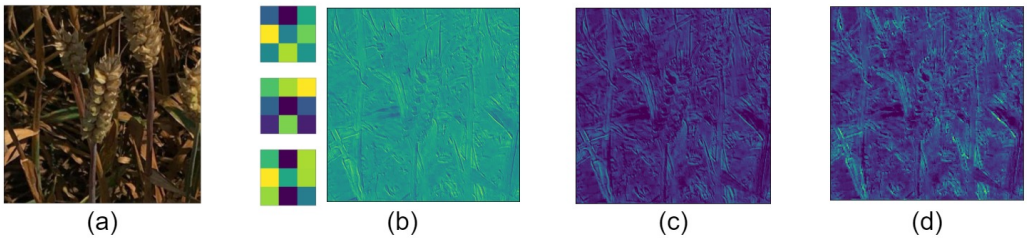


Figure 13: Demonstration of how convolutional layer affects an image. (a) original RGB input image; (b) three kernels convolved with red, green and blue channels of the input image (randomly generated) and the resulting image; (c) convolved image after being passed through activation function - we can see that darker parts from (b) got "deactivated", and only the lighter ones remained; (d) result of maxpooling operation - the image has halved in size and brighter parts have become more pronounced

images, and we want to be able to capture them regardless of their position and orientation. The property of translational equivariance is achieved in CNN's by the concept of weight sharing - since the same weights are shared across the images, if an object occurs in any image it will be detected irrespective of its position in the image. However, CNNs are not naturally equivariant to some other transformations such as changes in the scale or rotation of the image, which is why we added rotation and flipping of the images in the pipeline, ensuring that the network will learn to adapt to these transformations as well.

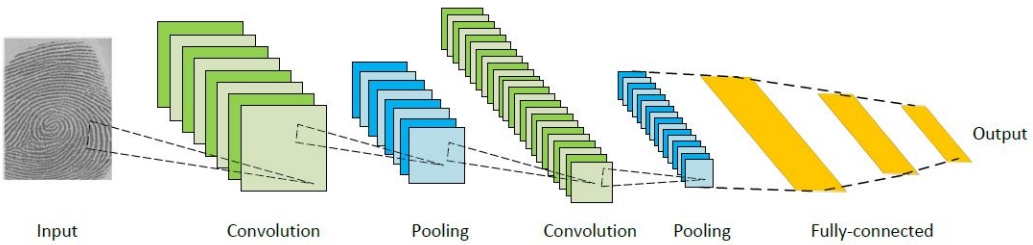


Figure 14: Illustration of how information gets passed through a convolutional network. Source [23]

2.2.3 U-Net

U-Net architecture was introduced by Ronneberger et al. in [28]. They modified and extended the FCN architecture so that it works with very few training images (the network relies on the use of data augmentation) and yields more precise segmentations. The U-Net architecture (Figure 15) comprises two parts, a contracting

path to capture context, and a symmetric expanding path that enables precise localization. The down-sampling or contracting part has a FCN-like architecture that extracts features with 3×3 convolutions. The up-sampling or expanding part uses up-convolution, reducing the number of feature maps while increasing their dimensions. This up-convolution is an operation that convolves a kernel with a "diluted" version of the original image. The dilution is achieved by padding the original image until the required size is achieved, while possibly spreading apart individual pixels. Feature maps from the down-sampling part of the network are copied and concatenated the up-sampling part to avoid losing pattern information. Finally, a 1×1 convolution processes the feature maps to generate a segmentation map that categorizes each pixel of the input image.

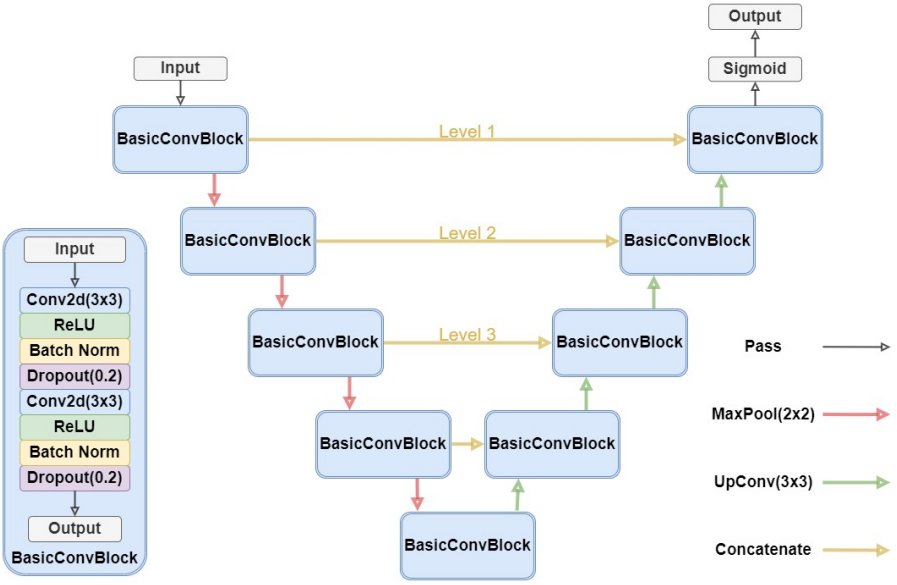


Figure 15: Architecture of U-Net.

We implemented a few minor modifications to the original U-Net structure. In the original paper, researches used 64 filters in the first level, and multiplied the number by 2 for every consecutive level[4]. We used the same pattern, but started with 16 filters. This was done because there was not a significant improvement in results, while the training time increased rapidly with the higher number of filters. We also added one layer of zero padding (adding a border of pixels all with value zero around the edges of the input images) to convolutions, so that original images would

[4]By "levels", we refer to layers with the same number of filters, descending and ascending. See Figure 15

not decrease in size with every convolution, and a sigmoid function at the output. Additionally, we inserted a dropout function after every convolution. This functions randomly zeroes some of the elements of the input filter with predefined probability p, and was shown to be an effective regularization technique [15]. Through trial and error it was established that all these modifications improved the results.

The network was implemented using PyTorch torch.nn.Module class. This approach can be used to wrap parameters, functions, and layers in a single class, making the code very elegant. Any deep learning model can simply be defined using predefined functions and layers from the torch.nn library and applied using a forward(input) method which passes the input through the network and returns the output. This also enables users to create nested structures in order to further simplify the code. In our case we created a separate class for BasicConvBlock (Figure 15) which is a mini-network that applies two convolutional layers consisting of a convolutional operation, relu activation function, batch norm operation and a dropout function to the input. This network was then used in the Unet class, where all the functions were applied following the schema from Figure 15. Each function illustrated in the Figure has an existing implementation in PyTorch.

This architecture for U-Net was originally designed for semantic segmentation of biomedical microscopic images. It quickly became apparent that the model can be expanded and applied to other fields as well, such as medical diagnostic images (x-rays, endoscopy and colonoscopy images, MRI-s etc.) and road segmentation from aerial images, for which the "upgraded" models used for this thesis were developed and applied.

2.2.4 U-Net++, ResUnet, ResUnet++

In the beginning of this project, only U-Net was used to isolate the wheat ears. While the results were promising, similarity between the ears and leaves sometimes proved challenging to distinguish for the algorithm. We therefore turned to a couple of upgraded versions of original U-Net architecture, to see if results could be improved and if ears could be segmented more precisely. The following models were also implemented:

- **U-Net++**. Created by Zhou et al. [37], U-Net++ (or Nested UNet) introduced additional layers in the skip connections of the U-Net architecture. The underlying hypothesis behind the idea is that the model can more effectively capture fine-grained details of the foreground objects when high-resolution feature maps from the encoder network are gradually enriched prior to fusion with the corresponding semantically rich feature maps from the decoder network (Figure 16). This is in contrast to the plain skip connections commonly used

in U-Net, which directly fast-forward high-resolution feature maps from the encoder to the decoder network, resulting in the fusion of semantically dissimilar feature maps. This model was developed for medical image segmentation, where anomalies need to be very precisely located. In their experiments, Zhou et al. showed U-Net++ to be superior to regular U-net, when applied to their field.

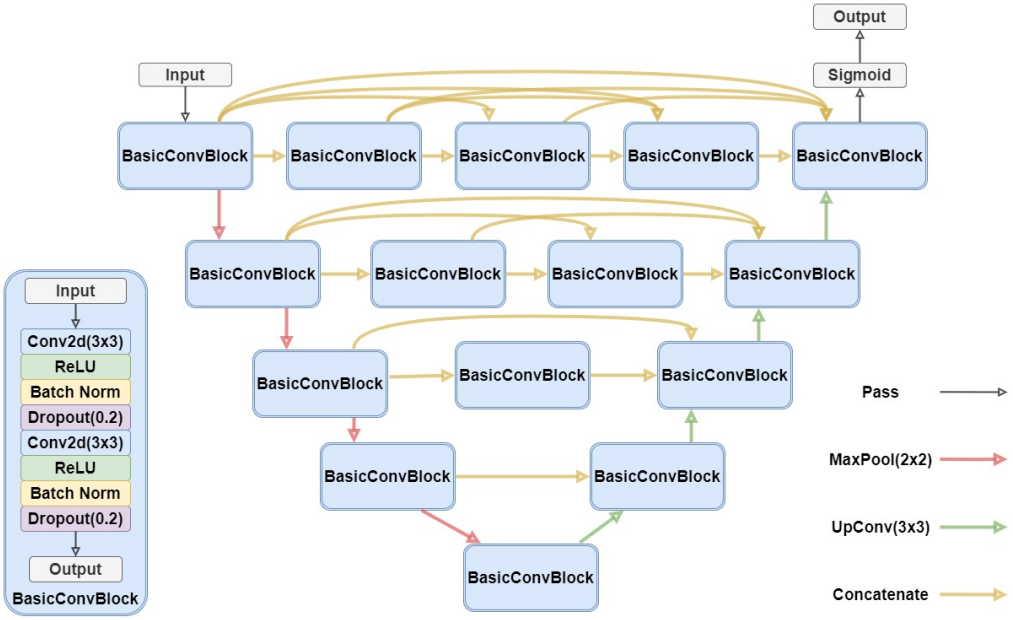


Figure 16: Architecture of U-Net++.

In our implementation, just like in [37], we took the code from U-Net and added convolutional blocks to the skip connections. Everything else remained the same.

- **ResUnet**. In 2015, the same year U-Net paper was published, He et al. proposed the residual neural network to address the degradation problem (phenomenon where with the network depth increasing, accuracy gets saturated and then degrades rapidly) [14]. Instead of using convolutional units, they propose *residual* units where each unit can be illustrated in the general form:

$$\mathbf{y}_l = h(\mathbf{x}_l) + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l), \tag{1}$$

$$\mathbf{x}_{l+1} = f(\mathbf{y}_l) \tag{2}$$

Here $\mathbf{x}_l$ and $\mathbf{x}_{l+1}$ are the input and output of the l-th residual unit, $\mathcal{F}(\cdot)$ is the residual function, $f(\mathbf{y}_l)$ is activation function and $h(\mathbf{x}_l)$ is a identity mapping function, a typical one is $h(\mathbf{x}_l) = \mathbf{x}_l$. A few years later, in 2018, Zhang et al. [35] created their own architecture named Deep Residual U-Net, where they replaced convolutional with residual blocks (Figure 17). This combination accomplished two things: first, the residual unit eases the training of the network, and second, the skip connections within a residual unit and between low levels and high levels of the network will facilitate information propagation without degradation, making it possible to design a neural network with much fewer parameters however could achieve comparable, or even better performance on semantic segmentation. Additionally, instead of using pooling operation to downsample the feature map size, a stride of 2 is applied to the first convolution block to reduce the feature map by half.

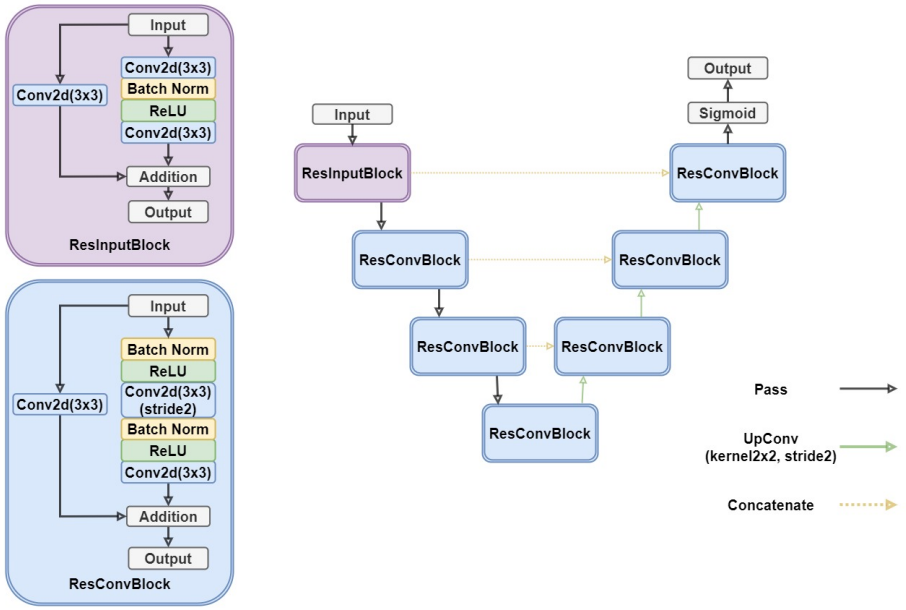


Figure 17: Architecture of ResUnet.

In implementation we used, the network was designed as is written in [35], the only difference being, like with the U-Net, that instead of using 64 filters in the first level, we used 16 for the same reasons as before, and doubled it in every following level. Separate nn.Module class instances were made again for convolutional blocks, here being named ResConvBlock because of the residual element, with addition of ResInputBlock for the first convolutional block, because it has different architecture then the rest of them (Figure 17).

- **ResUnet++**. With good results that ResUnet demonstrated, another upgrade was implemented in [18], with the new architecture named ResUnet++ (Figure 18). Jha et al. based their model on ResUnet, and added three new block units:

  1. **Squeeze and excitation units**: they boosts the representative power of the network by re-calibrating the features responses employing precise modeling inter-dependencies between the channels;
  2. **ASPP units**: The idea of ASPP (Atrous Spatial Pyramidal Pooling) comes from spatial pyramidal pooling, which is successful at re-sampling features at multiple scales. In ASPP, the contextual information is captured at various scales and many parallel atrous convolutions with different rates in the input feature map are fused. Atrous convolution allows controlling the field-of-view for capturing multi-scale information precisely;
  3. **Attention units**: They give attention to the subset of their input. The attention mechanism determines which parts of the network require more attention in the neural network. The main advantage of this mechanism is that it is simple, can be applied to any input size, enhance the quality of features that boosts the results.

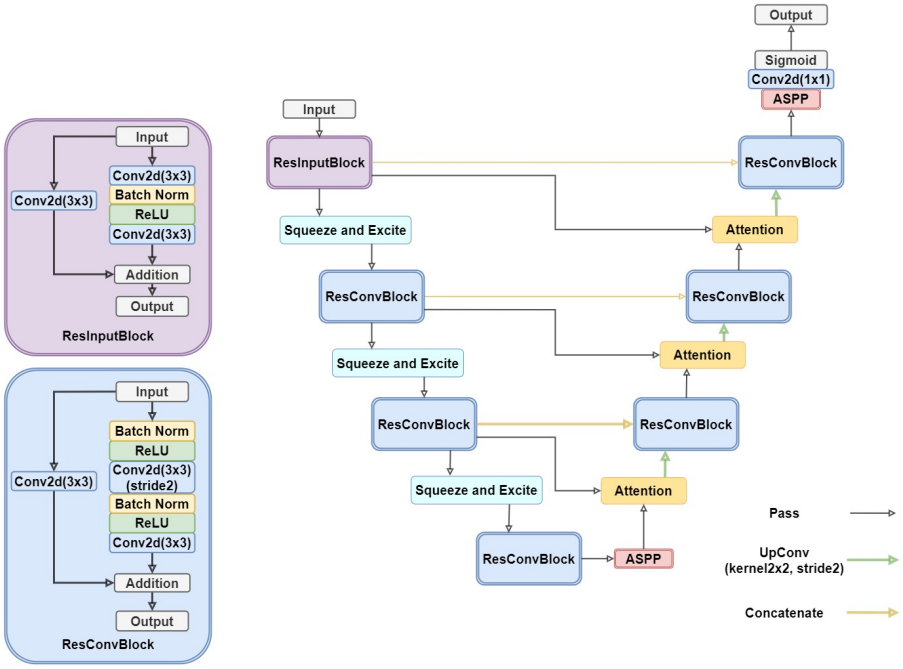


Figure 18: Architecture of ResUnet++.

The new blocks are illustrated in Figure 19. This model was created in order to achieve an accurate segmentation of images of colorectal polyps, and authors demonstrated that it outperformed UNet and ResUnet in this task. In implementation we used, the model was built on the ResUnet model as previously described, due to the similarities of the two networks. Separate nn.Module classes were created for the new special blocks, and they were inserted into the ResUnet class according to Figure 18 schema, with additional change of "UpConv" function - instead of using a trainable "ConvTranspose2d" pytorch module, a faster "Upsample" function was used, to reduce training time, seeing as the training time would be significantly increased by addition of special blocks.

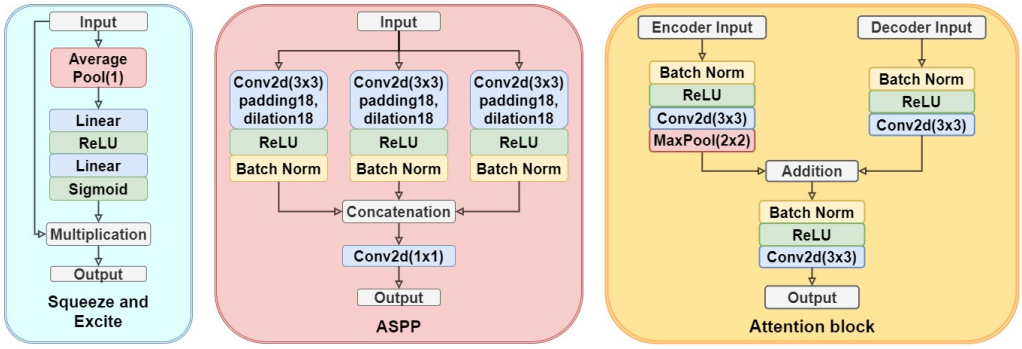


Figure 19: Special blocks used in ResUnet++

2.3 Metrics

There are two types of performance metrics used when training a deep learning algorithm: a loss function used for optimizing the algorithm (with the help of an optimizer) and metrics for evaluating different aspects of how the model actually performs on the data. With classification problems, the latter is the main indicator of how well the model performed. Here we will list the functions that we used and explain their purpose.

2.3.1 Loss Function

Machine learning algorithms are able to learn with the use of a loss function. It's a method of evaluating how well specific algorithm models the given data. If predictions deviate too much from actual results, that would be indicated by a very large value of a loss function. Gradually, with the help of some optimization function, loss function

learns to reduce the error in prediction [11]. The choice of a loss function depends on the type of problem that algorithm is supposed to help solve. We experimented during the training of the models with two loss function:

- **Binary Cross Entropy**. Also known as Log Loss, BCE is one of the most commonly used functions for binary classification tasks. We take flattened label image and flattened predicted label image and calculate the score using the following formula:

$$BCE(y, \hat{y}) = -w\big[y \cdot \mathsf{log}\hat{y} + (1 - y) \cdot \mathsf{log}(1 - \hat{y}))\big], \tag{3}$$

  where y are true labels, $\hat{y}$ are predicted labels, and w is the value of weights that could be added if there is a class imbalance or certain features need to be penalized more. This function essentially represents the log of actual predicted probability for the ground truth class, because if $y = 1$, second half of the function disappears, and if $y = 0$ the first half vanishes. An important aspect of this function is that it penalizes heavily the predictions that are confident but wrong.

- **Combined BCE with Dice score**. With this loss, we used BCE as defined previously, and added additional term which calculates Dice or F1 score, which in this case is defined as:

$$DL(y, \hat{y}) = 1 - \frac{1y\hat{y} + 1}{y + \hat{y} + 1} \tag{4}$$

  Here 1 is added to the numerator and denominator to ensure that the function is not undefined in edge case scenarios such as when $y = \hat{y} = 0$ [17]. This loss attempts to leverage the flexibility of Dice loss of class imbalance and at same time use cross-entropy for curve smoothing. Seeing as wheat ears make up around 10% of our images, it makes sense to try an approach that tackles the class imbalance issue.

2.3.2 Performance Metrics

An important tool for understanding model performance in classification tasks is the confusion matrix. It is a 2×2 matrix whose elements represent number of correctly or incorrectly classified data points (Figure 20).

Based on values of this matrix, and the priorities of classification, different metrics have been developed [16]. During training and testing, we monitored the following metrics:

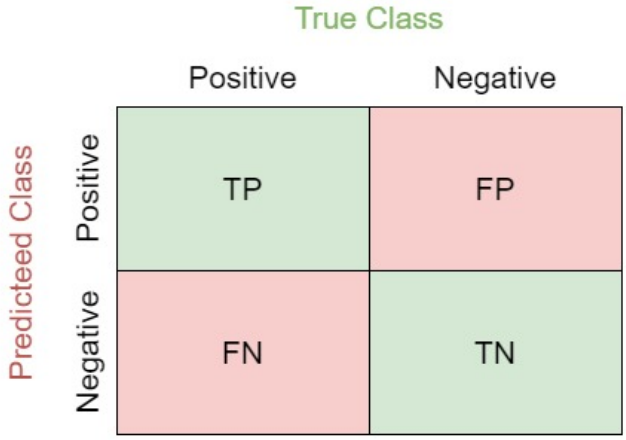


Figure 20: Confusion matrix.

- **Accuracy**. Accuracy is one of the most popular metrics in classification tasks. It is used as a statistical measure of how well a binary classification test correctly identifies or excludes a condition. The formula is:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

- **Precision**. Precision is used to measure the positive patterns that are correctly predicted from the total predicted patterns in a positive class, with formula:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

- **Recall**. Recall is used to measure the fraction of positive patterns that are correctly classified, with formula:

$$Recall = \frac{TP}{TP + TN} \tag{7}$$

- **F1 score**. Also known as Dice loss, this metric represents the harmonic mean between recall and precision values. As precision grows recall usually declines, and vice versa, meaning that high F1 score indicates good balance between the two. Formula is:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{8}$$

- **Jaccard index**. Jaccard index (also known as the Jaccard similarity coefficient or Intersection over Union), is a statistic used for gauging the similarity and diversity of label sets. It is defined as the size of the intersection divided by the size of the union of true labels and predicted labels, or, using the confusion matrix values, as:

$$Jaccard = \frac{TP}{TP + FP + FN} \tag{9}$$

In our code, we used implementation of these metrics from Scikit-learn library [26]. Important thing to note here is that handmade labels in our dataset were not perfectly applied, and are not completely indicative of the precise location of the ears. This means that even if our algorithm perfectly isolates the wheat ears, all of these metrics will still not show perfect results. However, they are still very representative of the success of the algorithm.

3 Experimental Results

Now that we have covered all the theoretical and practical preparations needed to perform our experiments, we are ready to describe how they were conducted and show and discuss obtained results.

3.1 Network Training

Experiments were executed using Python programming language, and specifically using Jupyter Notebook software. All four networks were trained on a computer with NVIDIA GeForce RTX 2060 graphics card with 6GB memory, AMD Ryzen 5 6300XT 6-Core processor and with 16GB RAM. NVIDIA is an American technology company that creates some of the best graphics cards currently available on the market, and they created CUDA - a parallel computing architecture that enables dramatic increases in computing performance by harnessing the power of the GPU [2]. Since our networks were created using PyTorch, which has an option of training the network on GPU using CUDA, we were able to significantly speed up the training process. Important thing to keep in mind is that computer vision tasks are computationally very demanding and often require much more sophisticated and powerful hardware and software in order to work effectively. During our experiments, we had problems with networks not being able to handle whole images as inputs, CUDA not having enough memory available to conduct a large amount of operations that come with greater image size. That is why, as previously mentioned, we cut the 2048×2048 images into smaller 256×256 pieces that the hardware could handle.

Because we were trying to compare the performance of different deep learning models, we made sure that experimental conditions were as similar as possible for all four networks. Training, validation and testing datasets were seeded during shuffling so that each network was learning on the same set of images. Batch sizes were set at 4, meaning that networks would take 4 images at a time. This is another point that was affected by the limited GPU - some models could not process batches larger than 4. Number of epochs was initially set to be 200, but we implemented an early stopping method, with the early stopping criterion of 15 epochs, forcing the models to stop the training if they do not detect significant improvement. In the end, none of the models needed more than 100 epochs (Figure 21).

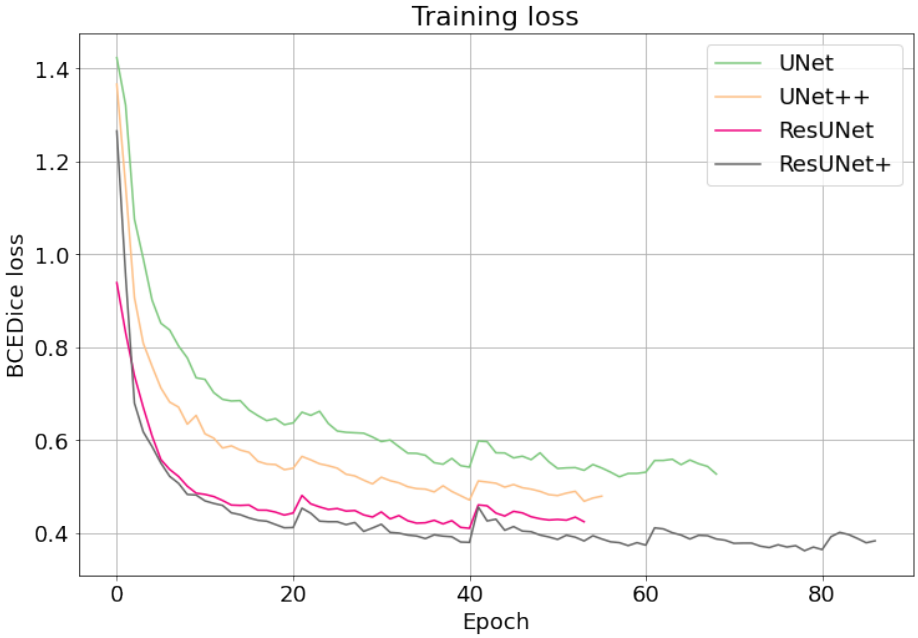


Figure 21: Training losses of all four models. Notice the different lengths of training for different models due to early stopping, and slight increases of loss after every 20 epochs due to learning rate adjustment.

For the loss function, after testing both BCE and BCEDice losses, we determined results were better with BCEDice loss. This was established by checking F1 and Jaccard scores on validation data, and noting the increased scores when using BCEDice loss. As for the optimizer, we tested both SGD and Adam optimizers and concluded that Adam was able to speed up training and reach lower values of the loss function. Initially, we set all learning rates to be fixed at 0.0001. However, we noticed that after a certain number of epochs pass, losses drop very slowly and early stopping would terminate the process after more than 100 epochs. For this reason, we added

a dynamic learning rate: all four models started with $\alpha = 0.0001$ and doubled every 20 epochs. This resulted in slight growth of the learning curve at these checkpoints, which can be seen in Figure 21. However, networks were able to recover and in the end less epochs were needed to complete the training, while results did not seem to worsen. This increase was not notable in the validation data, which can be seen in Figure 22.

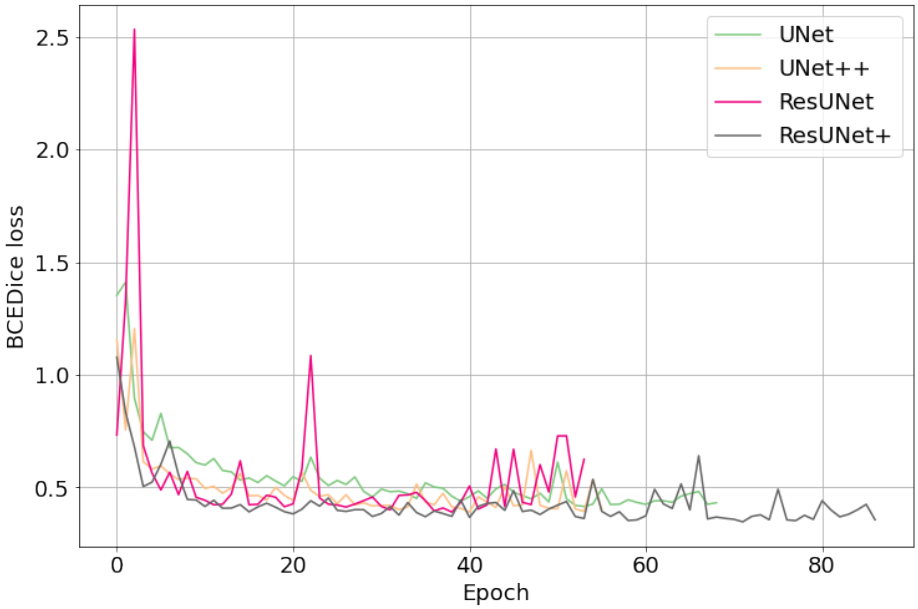


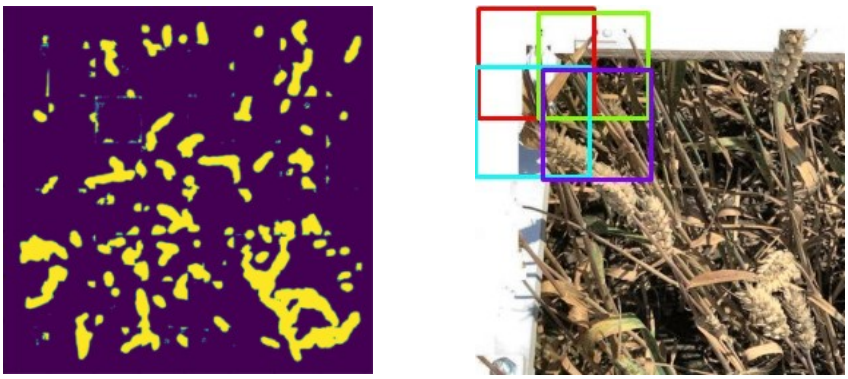
Figure 22: Validation losses of all four models. While there are obvious differences between training losses, validation losses showed much smaller gaps between performance of models.

Training times are available in Table 1. We can see that ResUnet++ had the longest training time, almost double the time it took Unet++ to train, while the fastest to learn was ResUnet. Appendix A provides a Figure comparing training and validation losses for each model individually.

3.2 Post-processing

None of the networks can handle taking an entire image of size 2048×2048 without GPU running out of memory, which is why the images were cut into smaller pieces for the training. On the test set however, we want to be able to have a segmentation map of the entire image. Sending smaller pieces of the images and concatenating them back together did not achieve satisfactory results - the resulting images had

clear marks of being cut and reconnected, as seen on Figure 23a. For this reason, we decided to cut the image by "sliding" a square 256×256 window along the image, with step size 128 along both axes. On Figure 23b this process is illustrated just for the upper left corner of one of the images. Different colored squares represent different image cuts, and after each of these cuts was forwarded to a model and got a prediction, we calculated the final prediction for overlapping areas of these squares as a pixel wise mean value of all available image pieces that contain it.



(a) (b)

Figure 23: Techniques for cutting test images: (a) after forwarding 64 non-overlapping pieces of an image to our models and putting them back together, some parts of resulting image had square-like structure; (b) cutting the image into overlapping pieces

The resulting image labels are matrices whose elements represent probabilities of pixels in their location on the original image being a part of a wheat ear. In order to get the actual label mask from those outputs (containing only binary values 0 and 1), we performed an operation called *thresholding* - replacing every pixel with 0 or 1, depending on whether its value is lower or higher than some fixed threshold constant. To determine the threshold we used the following formula:

$$label(x,y) = \begin{cases} 1, & \text{if } \hat{y}(x,y) \geq t \cdot max\{\hat{y}(x,y) | x = 1,...,N; y = 1,...,N\} \\ 0, & \text{otherwise,} \end{cases}$$

where $\hat{y}(x,y)$ is the model output value at (x,y) pixel coordinate, and N is the length and height of the image, and $t \in (0,1)$. Essentially, if the pixel of the output label has a value that is above the certain percentage of the maximal value in the entire label, we consider it to be a good indicator there is a wheat ear there and give it a value 1, otherwise it is 0. For the parameter t we tried out values starting from 0.1 to 0.9, with

step 0.1, and recorded the performance metrics on labels obtained after thresholding, repeating this process for all test images and each model. Final threshold values for each model are a mean of best performing thresholds for individual images, and they are: $t_{unet} = 0.44$, $t_{unet++} = 0.45$, $t_{resunet} = 0.45$, $t_{resunet++} = 0.48$. Behavior of average F1 score for test images with different choice of thersholds is shown on Figure 24

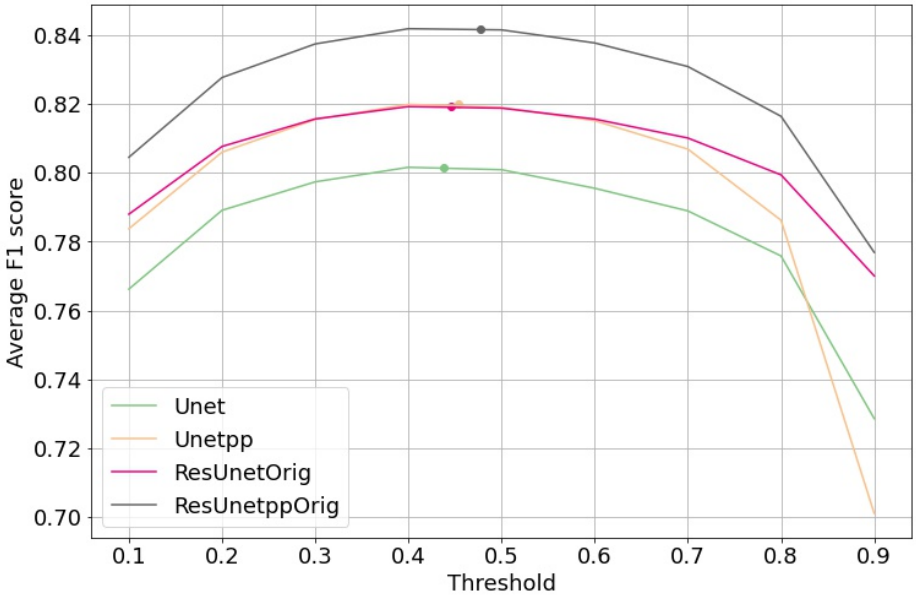


Figure 24: Average f1 score of test images for different threshold values. Thresholds were chosen to maximize average f1 score, and are shown as dots on each of the plot lines.

3.3 Numerical and Graphical Results

Table 1 contains the final performance metrics for all four models. These metrics were calculated by measuring each metric for each of the 13 test images and for each model, and taking an average value across all images. Model that has achieved the best results for a metric has its results in bold.

Table 1 shows that, performance-wise, ResUNet++ was the best model overall. This was expected, seeing as the ResUNet++ was built on top of other models. However, this performance comes at a cost - its training time and the time it takes to process a single image (runtime) are almost twice as long as the other models'. If one needed to label a large number of images, it could prove more efficient to use

Metrics	Models UNet	UNet++	ResUNet	ResUNet++
Accuracy	0.9595	0.9633	0.9633	**0.9682**
Precision	0.7840	0.8047	0.8056	**0.8298**
Recall	0.8258	0.8403	0.8377	**0.8587**
F1	0.8023	0.8202	0.8196	**0.8421**
Jaccard	0.6723	0.6976	0.6967	**0.7289**
Training time (s)	7299	8711	**6195**	16273
Runtime (s)	**1.0529**	1.8665	1.1645	2.8647

Table 1: Performance metrics and training time for all four models.

UNet++ or ResUNet, since they have only slightly worse results overall but much faster runtime. This comparison is illustrated in Figure 25.

We can see the difference in performance when displaying the resulting labels against each other. In Figure 26, we can see that all networks isolated the wheat ears very precisely when compared to the hand-made label. There is also an additional wheat ear that ResUNet++ found that was not labeled in the original image.

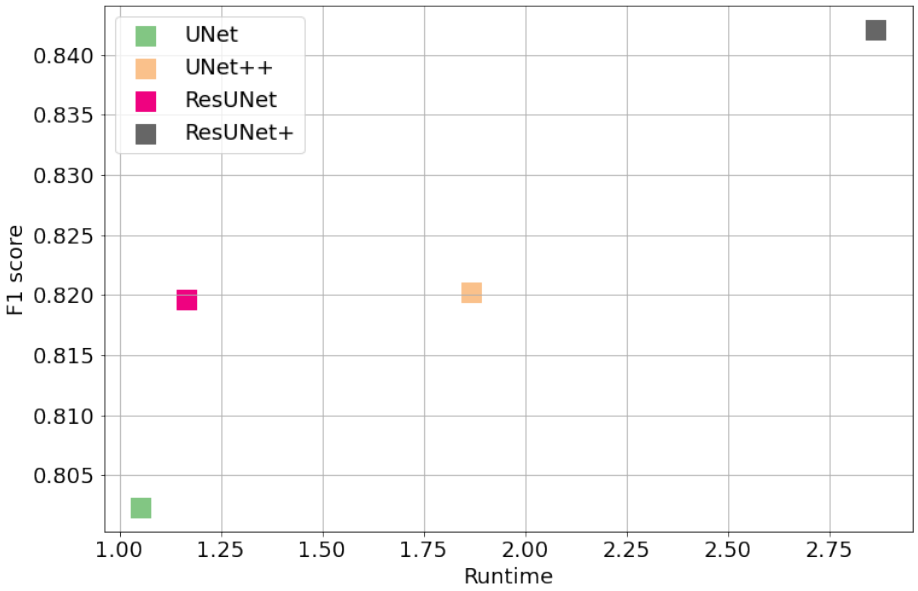


Figure 25: Runtime vs average f1 score for all four models.

All four algorithms had trouble with the more mature wheat images, often labeling shadows and leaves as wheat ears. We can see the example of this on Figure 27. All four models mislabeled a good portion of the image. Here we can see that again,

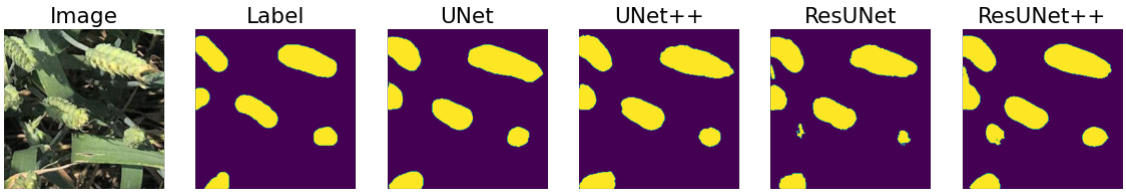


Figure 26: An example of good performance of all models. Note that ResUNet++ detected a wheat ear that was in the original image but not labeled.

ResUNet++ had the least amount of errors. In Appendix B we will provide best and worst performing complete images for all four models.

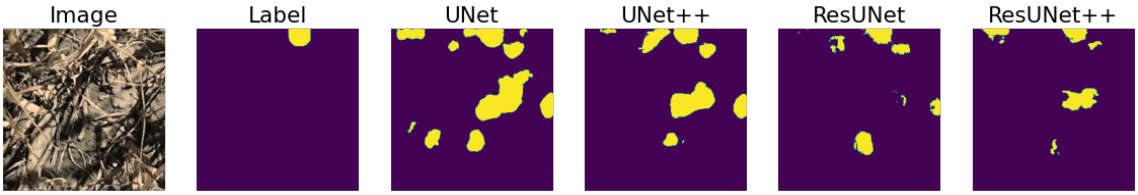


Figure 27: An example of poor segmentation by all models. Note that the ResUNet++ had the least amount of mislabeled pixels.

4 Discussion

ResUNet++ is the model that had the best results in our experiments. In this section we will explore how it was able to achieve these results, and provide ideas on how they could be further improved.

4.1 Network Exploration

It is a common critique of deep learning algorithms that they function on a "black box" principle - data comes in, results come out, and everything in between is unknown or unreadable. However, in computer vision tasks we can, to a certain extent, visualize how deep learning algorithms make their decisions. Here we will show how our best performing algorithm, ResUNet++, extracts information from images and uses it to create wheat ear labels.

The model has three "descending" blocks, one "bridge" block, and three "ascending" blocks , as shown on Figure 18. Each new descending block introduces double the amount of filters that previous block had, while reducing individual filter sizes

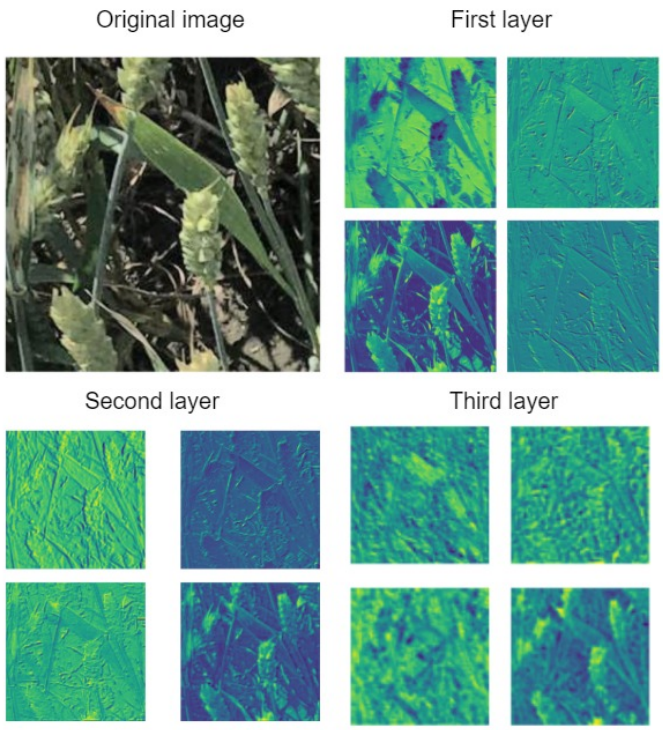


Figure 28: A piece of wheat image and samples of filters resulting from that image passing through first three layers of ResUNet++

to a quarter of the previous size. In Figure 28 we can see how this process looks on an example of a piece of image of wheat. The first three layers, a.k.a. "the encoder" of the network, isolate certain features, sometimes emphasizing the difference between lighter and darker parts of the image, sometimes showing edges of objects, all while progressively reducing the size of the filters, and keeping the most relevant information.

In the "bottom" or "bridge" part we can already see the labels forming, as shown in Figure 29. Bottom convolutional layer on the left of the Figure further reduced the filter size and kept the most relevant information on the location and shape of wheat ears, while ASPP layer on the right helped bring focus on the ears by introducing wider context with dilated convolutions.

Going upwards, each level first introduces attention blocks, which reduce the noise and bring attention to the important parts (wheat ears), upsamples the filters to four times their size and then combine them with the output of the blocks from the same level in the "encoder" part. After that, filters are put through convolution blocks

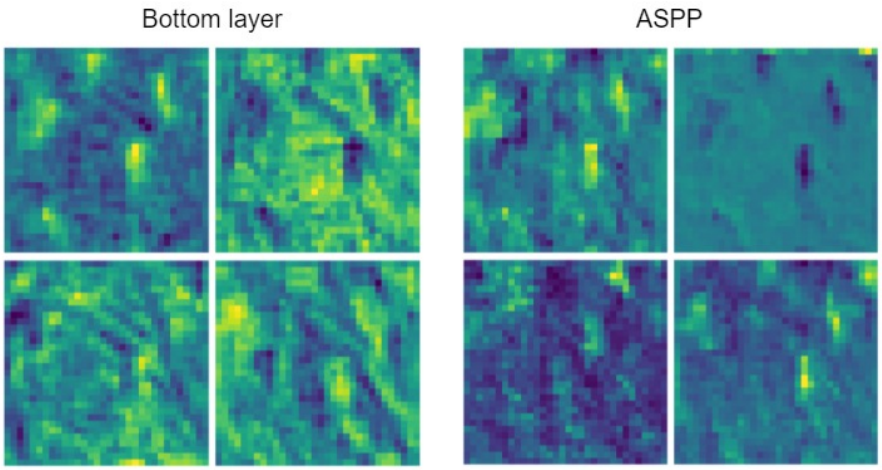


Figure 29: A piece of wheat image and samples of filters resulting from that image passing through the bottom layers of ResUNet++

which reduce the number of filters by half. Figure 30 illustrates how attention and convolutional layers transform the filters. We can see how wheat ears become more clear with each subsequent level.

Finally, at the two final layers, first the second ASPP layer is applied, after which the final convolutional layer and sigmoid activation functions give us the final output. We can see in Figure 31 how ASPP introduces "fuzzines" around the edges of wheat ears, as a result of diluted convolution. This is the additional spacial information that helps the network separate wheat from the rest. The output precisely labels the positions of wheat ears.

4.2 Future Research

In this thesis we have shown that UNet-based models are very good at wheat ear segmentation task. However, there are many ways our results could be improved. In this section we will explore ideas that could provide better outcome.

Images on which our models were trained are very similar in nature. Each image was taken in strictly predefined conditions, in order to ensure best visibility. While this makes it easier for our models to learn patterns, it could prove difficult for them to generalize to other wheat types in different conditions (cloudy day, night, artificial lighting), taken at different angles. Some of these points could be emulated by artificially enriching the dataset with more preprocessing steps (adjusting image gamma, random size cutouts), but there is no doubt that introducing more natural

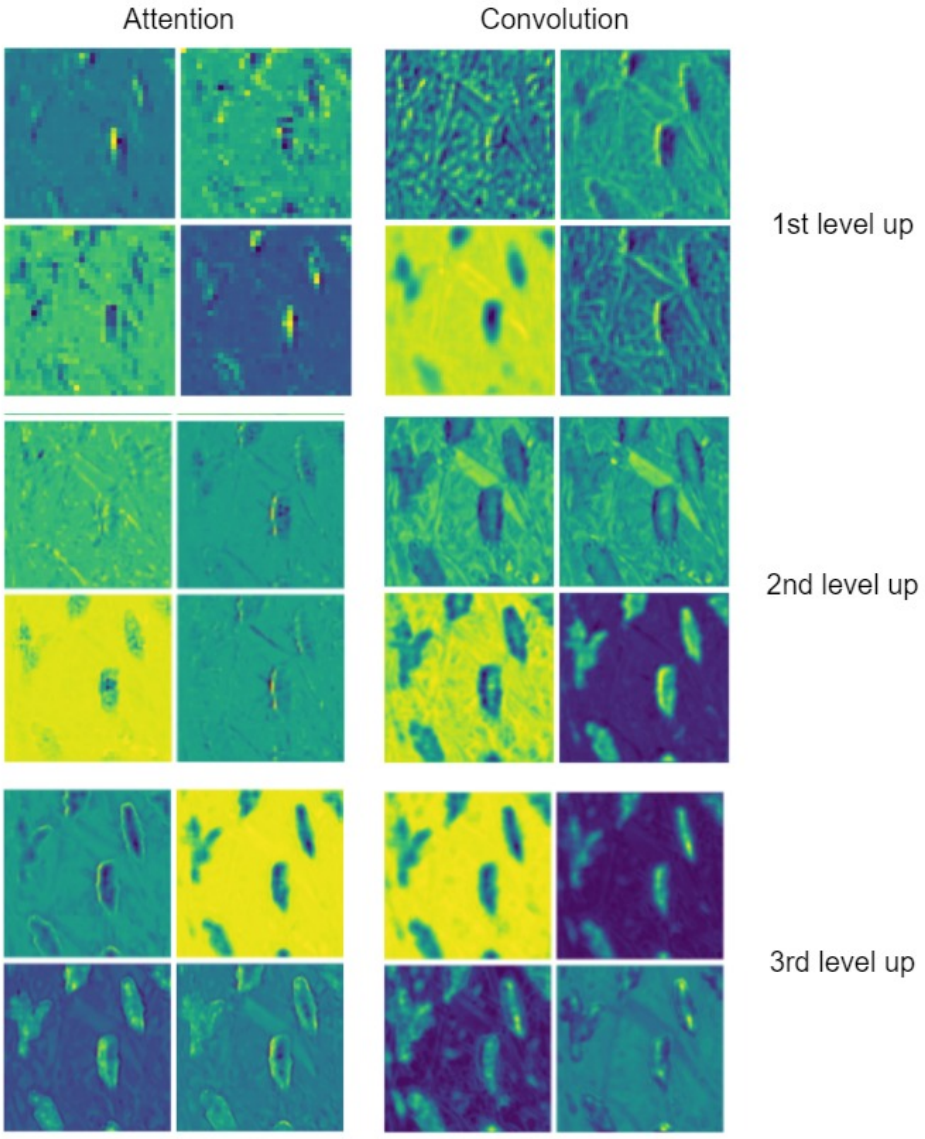


Figure 30: A piece of wheat image and samples of filters resulting from that image passing through the layers of ResUNet++ in the decoder part, going "upwards".

images would be beneficial.

Enlarging the dataset will only increase training time, which is already long. Cloud GPU services could solve this problem. There are a lot of "rental" GPU services available that have much better capabilities that the device used for model training

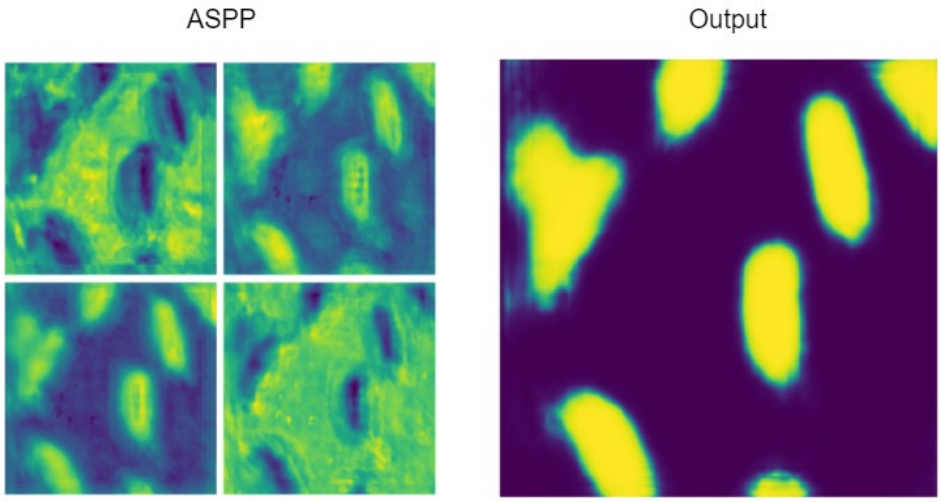


Figure 31: Filters from final layers of ResUNet++

in this project, such as Azure ML Studio and Amazon SageMaker. With improved training time, we could use larger training sets and perform more extensive hyper-parameter tuning. Another technique that could help networks learn faster is transfer learning - we could use an already trained deep learning model for feature extraction ("encoder" part of our models) and train only the "decoder". This approach proved to be very effective in computer vision tasks, reducing the training time and need for large datasets. [33]

Let us not forget that the goal of wheat ear segmentation is to count the wheat ears in order to get an estimate of yield. There are many techniques that could be combined with our approach to get a wheat ear count estimate. One idea is to apply the border weight technique used by authors of the original UNet paper [28], which adds additional error to edges of objects to be segmented, forcing the algorithm to isolate separate shapes, and then perform a connected component count. The drawback of this approach is that new labels will have to be created that mark a very precise location of wheat ears, and this will take a lot of time. Another idea is to add another network that will learn alongside the main network, whose goal will be to isolate just instances of wheat ears, and not perform precise segmentation. This approach was done in [19], with great results.

5 Conclusion

In this thesis, we have explored the effectiveness of UNet and various UNet-based deep learning models in the task of segmenting wheat ears from images of wheat in outdoor conditions. We performed experiments on images provided by BioSense institute, and used well known computer vision techniques techniques during preprocessing for image enhancement and dataset enrichment in order to get better results. Our conclusion is that this family of models give good results on average, with tendency to sometimes mislabel leaves and shadows as wheat ears on images of more mature wheat. Best performing model was ResUNet++, which was also the slowest model, due to its complexity. We provided analysis on how this model extracts information from images, and given ideas for future research. We believe that this model, with some modifications listed in Section 4.2, could be a very effective tool for solving crop yield estimation problem.

Bibliography

[1] Yield and production estimation of wheat for punjab, pakistan. https://nasaharvest.org/news/yield-and-production-estimation-wheat-punjab-pakistan.

[2] Nvidia cuda, Jan 2022.

[3] N. Alharbi, J. Zhou, and W. Wang. Automatic counting of wheat spikes from wheat growth images. In *ICPRAM*, pages 346–355, 2018.

[4] G. Bradski and A. Kaehler. Opencv. *Dr. Dobb's journal of software tools*, 3, 2000.

[5] F. Cointault, D. Guerin, J.-P. Guillemin, and B. Chopinet. In-field triticum aestivum ear counting using colour-texture image analysis. *New Zealand Journal of Crop and Horticultural Science*, 36(2):117–130, 2008.

[6] B. C. Curtis, S. Rajaram, H. Gómez Macpherson, et al. *Bread wheat: improvement and production.* Food and Agriculture Organization of the United Nations (FAO), 2002.

[7] FAO. World food situation. fao cereal supply and demand brief, 2015.

[8] J. A. Fernandez-Gallego, S. C. Kefauver, N. A. Gutiérrez, M. T. Nieto-Taladriz, and J. L. Araus. Wheat ear counting in-field conditions: high throughput and low-cost approach using rgb images. *Plant methods*, 14(1):1–12, 2018.

[9] K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[10] R. C. Gonzalez, R. E. Woods, et al. Digital image processing, 2002.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning.* MIT press, 2016.

[12] Z. Grbovic, M. Panic, O. Marko, S. Brdar, and V. Crnojevic. Wheat ear detection in rgb and thermal images using deep neural networks. *environments*, 11(12):13, 2019.

[13] M. M. Hasan, J. P. Chopin, H. Laga, and S. J. Miklavcic. Detection and analysis of wheat spikes using convolutional neural networks. *Plant Methods*, 14(1):1–13, 2018.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[16] M. Hossin and M. N. Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[17] S. Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–7. IEEE, 2020.

[18] D. Jha, P. H. Smedsrud, M. A. Riegler, D. Johansen, T. De Lange, P. Halvorsen, and H. D. Johansen. Resunet++: An advanced architecture for medical image segmentation. In *2019 IEEE International Symposium on Multimedia (ISM)*, pages 225–2255. IEEE, 2019.

[19] S. Khaki, N. Safaei, H. Pham, and L. Wang. Wheatnet: A lightweight convolutional neural network for high-throughput image-based wheat head detection and counting. *arXiv preprint arXiv:2103.09408*, 2021.

[20] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár. Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9404–9413, 2019.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[23] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

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

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] M. P. Pound, J. A. Atkinson, D. M. Wells, T. P. Pridmore, and A. P. French. Deep learning for multi-task plant phenotyping. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2055–2063, 2017.

[28] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[29] P. Sadeghi-Tehran, N. Virlet, E. M. Ampe, P. Reyns, and M. J. Hawkesford. Deepcount: in-field automatic quantification of wheat spikes using simple linear iterative clustering and deep convolutional neural networks. *Frontiers in plant science*, 10:1176, 2019.

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[31] F. Sultana, A. Sufian, and P. Dutta. Evolution of image segmentation using deep convolutional neural network: a survey. *Knowledge-Based Systems*, 201:106062, 2020.

[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[33] Vinithavn. The power of transfer learning in deep learning. https://medium.com/analytics-vidhya/the-power-of-transfer-learning-in-deep-learning-681f86a62f79, Jan. 2022.

[34] Y. Wang, Y. Qin, and J. Cui. Occlusion robust wheat ear counting algorithm based on deep learning. *Frontiers in Plant Science*, 12:1139, 2021.

[35] Z. Zhang, Q. Liu, and Y. Wang. Road extraction by deep residual u-net. *IEEE Geoscience and Remote Sensing Letters*, 15(5):749–753, 2018.

[36] C. Zhou, D. Liang, X. Yang, H. Yang, J. Yue, and G. Yang. Wheat ears counting in field conditions based on multi-feature optimization and twsvm. *Frontiers in plant science*, 9:1024, 2018.

[37] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 3–11. Springer, 2018.

APPENDIX A

Figure 32 shows training and validation losses compared individually for every model.

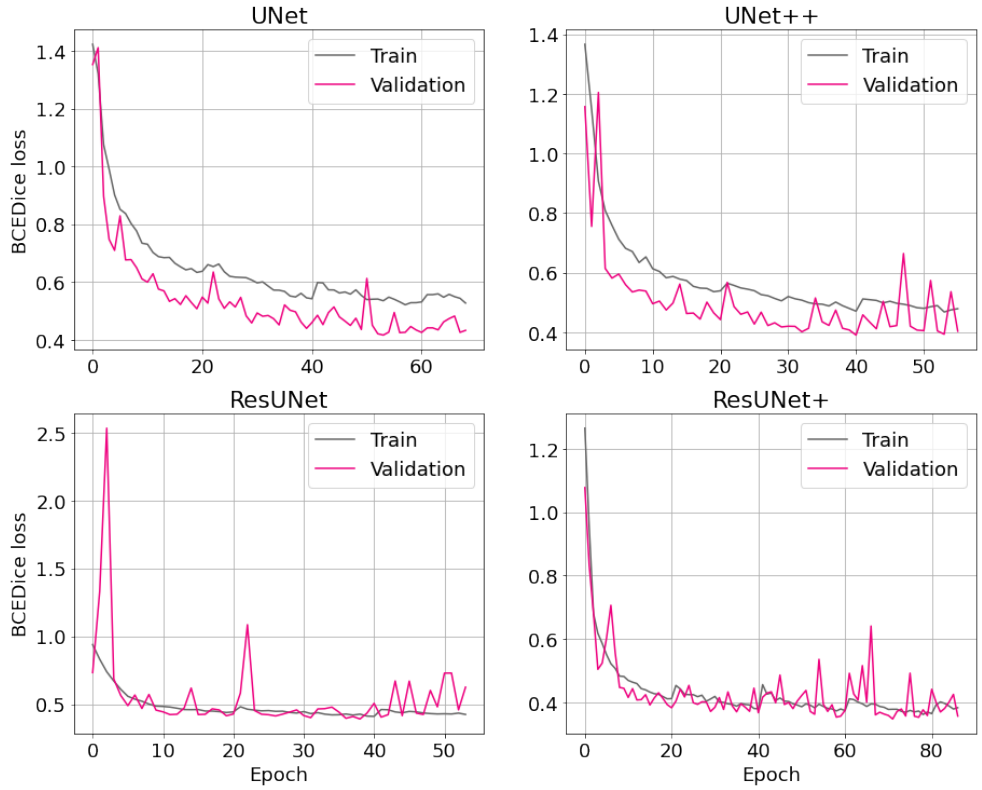


Figure 32: Training vs. validation losses for all four models.

APPENDIX B

Enclosed are whole images of wheat, with both hand made labels(red) and the resulting labels (blue) from ResUNet++. Notice that most of the labels overlap resulting in a violet color. First image is from the batch taken on June 10, while the second is from the second batch taken on June 26. All networks had more errors on the images of more mature wheat, but still produced great results.

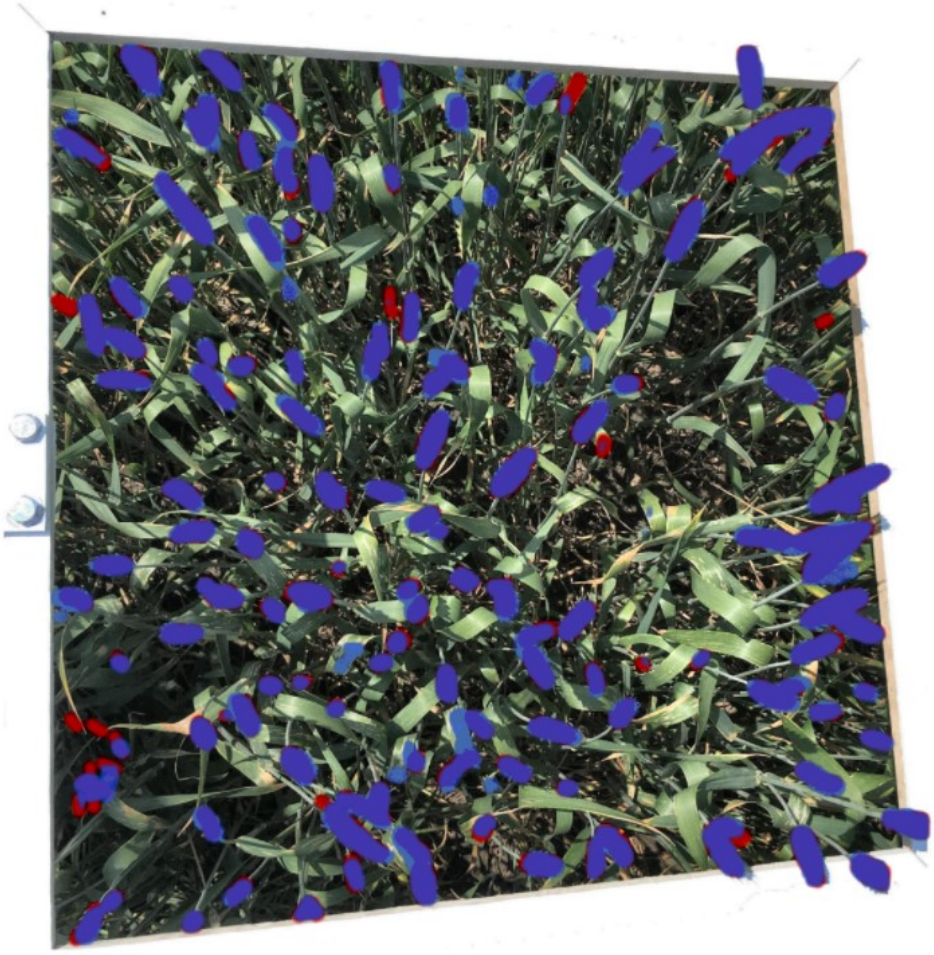


Figure 33: Whole image from June 10 with labels

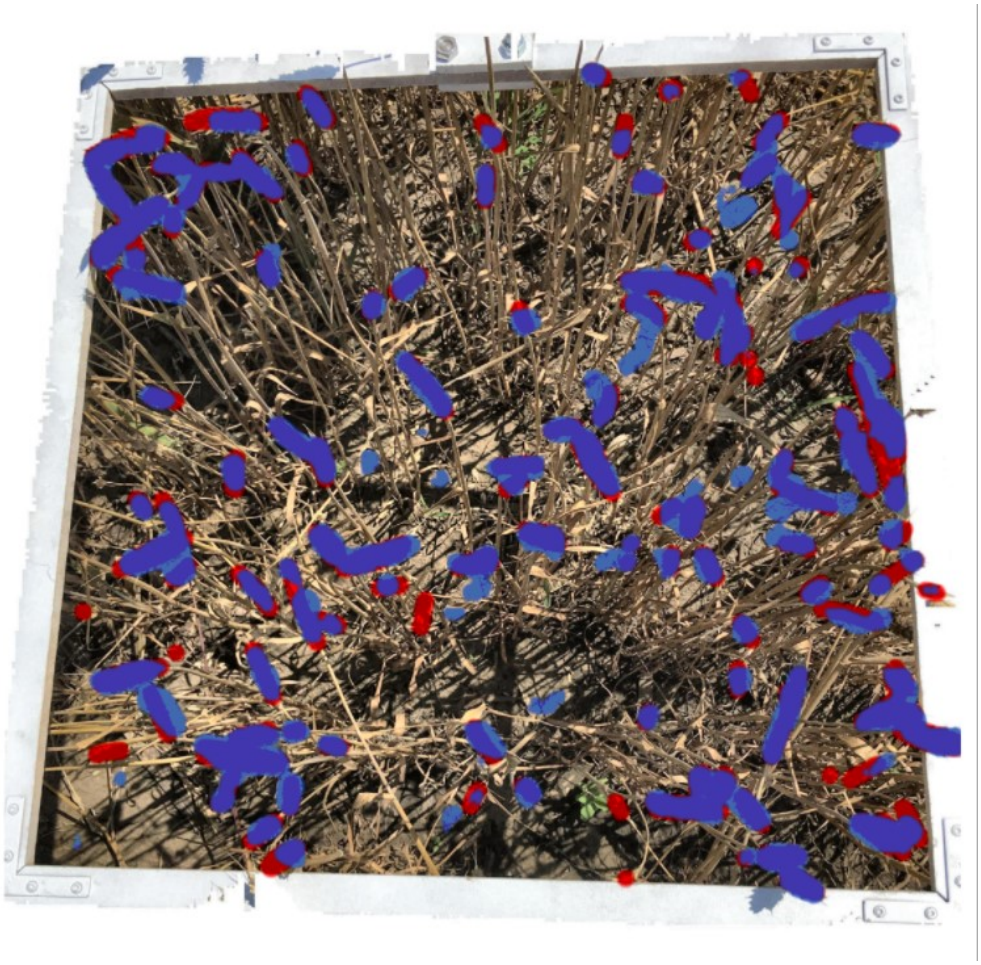


Figure 34: Whole image from June 26 with labels

BIOGRAPHY

Tamara Krivokuca was born on 28^{th} of January 1995 in Novi Sad, Serbia. She grew up in the town of Odzaci, where she attended elementary school Branko Radicevic and grammar school Jovan Jovanovic Zmaj. In 2013 she started her bachelor degree studies in theoretical mathematics at Faculty of Natural Sciences, University of Novi Sad, which she finished in 2017 with GPA 9.16. Next year, she continued her education at the same university, enrolling in masters degree program in Applied Mathematics - Data Science. She finished the master program with GPA of 9.88. From July 2019 to October 2020, he worked as a Researcher at the Faculty of Sciences, University of Novi Sad, at the Horizon2020 project. She was also a junior assistant at Petnica Science Center from January 2018 until January 2020, where she helped educating highschool students interested in learning advanced mathematical topics not covered in highschool, and organizing and running seasonal seminars. Starting from September 2021, she works at VegaIT as a Data Engineer.

UNIVERZITET U NOVOM SADU ● Prirodno-matematički fakultet
21000 Novi Sad, Trg Dositeja Obradovića 3

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, RBR:	
Identifikacioni broj, IBR:	
Tip dokumentacije, TD:	Monografska dokumentacija
Tip zapisa, TZ:	Tekstualni štampani materijal
Vrsta rada, VR:	Master rad
Autor, AU:	Tamara Krivokuća
Mentor, MN:	dr Oskar Marko
Naslov rada, NR:	Segmentacija klasova pšenice pomoću modela dubokog mašinskog učenja baziranih na UNet arhitekturi
Jezik publikacije, JP:	Engleski
Jezik izvoda, JI:	Engleski
Zemlja publikovanja, ZP:	Republika Srbija
Uže geografsko područje, UGP:	Vojvodina
Godina, GO:	2022.
Izdavač, IZ:	Autorski reprint
Mesto i adresa, MA:	Novi Sad, Departman za matematiku i informatiku, PMF, Trg Dositeja Obradovića 2
Fizički opis rada, FO:	Poglavlja (5), strana (44), literaturnih citata (38), tabela (1), slika (34), grafikona (0), priloga (2)
Naučna oblast, NO:	Matematika
Naučna disciplina, ND:	Primenjena matematika
Ključne reči, KR:	Mašinsko učenje, Duboko učenje, Implementacija, Konvolutivne neuronske mreže, Unet, Obrada slike, Segmentacija slike, Python, PyTorch
Univerzalna decimalna klasifikacija, UDK:	

Čuva se, ČU:	Biblioteka Departmana za matematiku i informatiku Prirodno-matematičkog fakulteta, u Novom Sadu
Važna napomena, VN:	
Izvod, IZ:	U ovom radu ispitujemo efikasnost algoritama baziranih na UNet arhitekturi na problemu segmentacije slika klasova pšenice na njivi. Ukupno su implementirana četiri modela i upoređeni rezultati.
Datum prihvatanja teme od strane NN veća, DP:	30.09.2021
Datum odbrane, DO:	16.03.2022
Članovi komisije, KO: Predsednik: Mentor: Član:	 Prof. Dr Dušan Jakovetić, vanredni profesor, PMF, Novi Sad dr Oskar Marko, naučni saradnik Instituta BioSense, Novi Sad dr Sanja Brdar, naučni saradnik Instituta BioSense, Novi Sad

UNIVERSITY OF NOVI SAD ● Faculty of Science
21000 Novi Sad, Trg Dositeja Obradovića 3

KEY WORDS DOCUMENTATION

Accession number, ANO:	
Identification number, INO:	
Document type, DT:	Monograph type
Type of record, TR:	Textual material, printed
Contents code, CC:	Master thesis
Author, AU:	Tamara Krivokuća
Mentor, MN:	dr Oskar Marko
Title, TL:	Segmentation of Wheat Ears Using Deep Learning Models Based on UNet Architecture
Language of text, LT:	English
Language of abstract, LA:	English
Country of publication, CP:	Republic of Serbia
Locality of publication, LP:	Vojvodina
Publication year, PY:	2022.
Publisher, PU:	Author's reprint
Publ. place, PP:	Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Trg Dositeja Obradovića 2
Physical description, PD:	Chapters (5), pages (44), references (38), tables (1), figures (34), graphs (0), additional lists (2)
Scientific field, SF:	Mathematics
Scientific discipline, SD:	Applied Mathematics
Key words, KW:	Machine learning, Deep learning, Implementation, Convolutional neural networks, UNet, Image processing, Image segmentation, Python, PyTorch
Universal decimal classification, UDC:	
Holding data, HD:	Department of Mathematics and Informatics' Library, Faculty of Science, Novi Sad
Note, N:	

Abstract, AB:	In this thesis we research the effectivenes of Unet based deep learning models in solving the problem of segmentation of images of wheat ears in outdoor conditions. In total four models were implemented and results were compared.
Accepted by the Scientific Board on, ASB:	30.09.2021
Defended on, DE:	16.03.2022
Thesis defend board, DB:	
Chairperson:	Prof. Dr Dušan Jakovetić, professor at PMF, Novi Sad
Mentor:	dr Oskar Marko, scientific associate of BioSense Institute, Novi Sad
Member:	dr Sanja Brdar, scientific associate of BioSense Institute, Novi Sad