Programming for GIS
and
Remote Sensing
How-To Guide

(EGM 722)

Josep Serra Gallego

serra_gallego-j@ulster.ac.uk

Student number: B00847706

Introduction and context

During the second term of this MSc programme, I joined a Work Placement module run by the University in partnership with OSNI. The project in which I was invited to participate involved identifying areas of mixed land cover on orthophotography 4-band raster data (RGB + near infrared), with a spatial resolution of 40 cm.

A significant amount of work had been covered by the OSNI team. There was, however, an area which had not been unexplored: using Machine Learning and Deep Learning for pixel classification.

Deep learning is a type of Machine Learning with several layers of nonlinear processing which allow users to identify patterns, objects, and pixels through models. It is a significant improvement on previous Machine Learning systems since it does not require vast amounts of training samples produced by expert users.

The Work Placement and the project were completed very successfully: I managed to conduct pixel classification of orthophotography data provided by OSNI to identify land cover using sparse sampling (the time I took to collect training samples was never more than 30'). The outcome of the classification was assessed and found to outperform previous approaches.

Although the project was a success, I realised there was room for improvement. I had conducted the analysis using ArcGis Pro user interfaces, tools, and menus. However, I believed that using ArcGis Pro's python environment, libraries and dependencies would allow me to deepen my analysis, customise it and provide additional features. I believed I could build a tool that could be used by people with little or no knowledge of python or Deep Learning algorithms, to speed up classification processes without requiring vast numbers of samples and staff hours.

In short, this project is the result of applying what I have learned in EGM722 (Programming for GIS and Remote Sensing) to a topic I was introduced to while doing EGM725 and consists of providing a tool to automatise land cover pixel classification by pre-processing target data, exporting training samples, training, and evaluating deep learning models, and displaying results – all without requiring any knowledge from the user.

Setup/Installation

Public GIT repository: https://github.com/JSG-GIS-722/Project.git

Test data which is affected by specific copyright arrangements can be found through this link.

A list of all dependencies used by the project can be found in the requirements file (ctrl + click to open).

The best way to use this ArcGis Notebook (EGM722Classifier.ipynb) is to open it from a new ArcGis Pro project. Some of the Deep Learning dependencies used in the Notebook require the ArcGIS Image Analyst extension, so a licensed copy of ArcGis Pro needs to be open while running the ArcGis Notebook. This repository includes the required .yml file to duplicate the python environment, but the before mentioned licensing constrain might still apply.

This project was developed using ArcGis Pro v2.9 with ESRI's Deep Learning libraries 2.9 and a functioning dedicated python environment. ESRI's Deep Learning libraries are available here.

Hardware-accelerated GPU scheduling is recommended to reduce latency and improve performance when executing processor-intensive Deep Learning algorithms.

The project files are divided in two folders, mapped within a "U:" drive created to facilitate file management. In this drive, I created a folder called **Project** for public GIT files, and a **ProjectData** folder for OSNI-copyright files and intermediate output files. Users of this ArcGis Pro notebook should replicate this structure or adapt the script accordingly.

The project uses training data obtained from OSNI's own 4-band orthoimagery (with spatial resolution of 40 cm, and red, green, blue and NIR bands). Training data used in this Notebook needs to be collected using the Label Objects for Deep Learning tool for Imagery Classification, exported with the Export Training Data feature, and saved as a feature class file called **trainingsamples.shp** in ProjectData.

Finally, this ArcGis Pro notebook creates a considerable number of files and folders, and therefore it is necessary to ensure that the device in which it is run has a minimum of 5 GB of memory available).

Methods

Pre-Processing

This ArcGis Notebook starts by pre-processing the target raster (i.e., the image we want to classify) to optimize it for Deep Learning algorithms. The first stage of pre-processing consists of extracting the first three bands of our raster (some Deep Learning algorithms support only 3-band rasters) and smoothing the target image using a 5 x 5 filter. Secondly, the image is stretched. Then, the resulting raster is resampled and segmented. Segmentation is a key process as it changes the characteristics of the image and allows classification. The parameters used for this process follow those used by the supervising team at OSNI and featured high spectral detail (18 out of 20), low spatial detail (3) and a minimum segment size of twenty-five pixels.

The final pre-processing step involves verifying that the target raster is an 8-bit unsigned file by saving it in that format.

Export Training Samples

After pre-processing our target raster, training data (which had been gathered through ArcGis Pro and exported into a feature class) is used to generate the Chips and Labels required for training the Deep Learning model. Chips are small sub-images (which include the feature of interest) while labels are their corresponding classification category. Our training samples identified eight categories of land cover (Figure 1).

Figure 1 - Land Cover Classes

Data preparation

Training data (i.e., Chips and Labels) is then read and inspected by the prepare_data() arcgis.learn method. This method produces the structures required for training and validation, and the specified changes relating to data augmentation, chip and batch size, and train-validation split percentage it sets the right hyper parameters to create a good model.

Classifiers: U-NET and PSPNet

Once the training data is prepared, it is used to train 2 Deep Learning classification models implemented in ArcGis Pro: U-NET and PSPNet.

U-NET is a convolutional neural network (CNN) that is designed to learn from very few training samples, and which was developed at the University of Freiburg in 2015. CNNs use little pre-processing compared to other image classification algorithms.

As we can see in Figure 2 below, the U-Net architecture consists of a series of convolutions (Figure 2) where a filter is applied (also known a kernel or feature extractor) on input data (Yao *et al* 2018). The results are then pooled, retaining only the key features and, crucially, reducing the resulting image (Ronneberger *et al* 2015). This part of the architecture is known as the coder section, and it is reversed in the decoding section, through up sampling the data and applying the features obtained by the kernels (the grey arrows in Figure 2). This process means that the network learns to optimize the filters through automated learning, requiring little human intervention, which is a major advantage (especially in our scenario at OSNI, where time and labour were valuable and scarce resources).

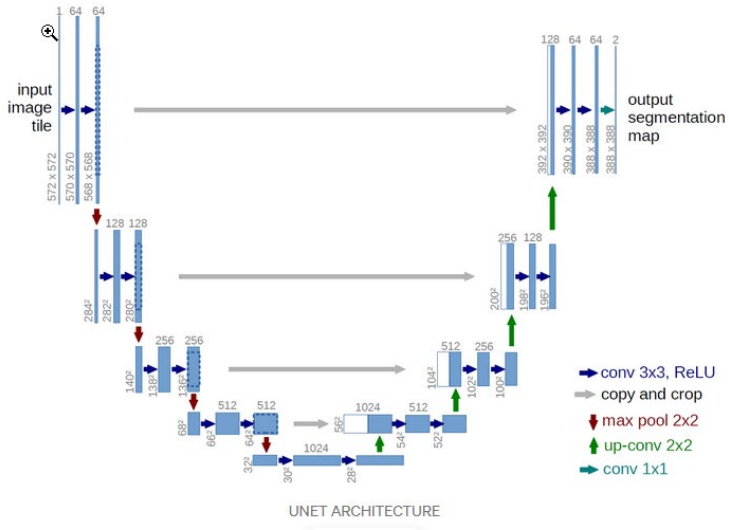

Figure 2 - The U-Net architecture (Ronneberger *et al,* 2015)

PSPNet (which stands for Pyramid Scene Parsing Network) incorporates global features through dilated convolutions for scene parsing and classifications as shown in Figure 3. It includes a Pyramid Pooling Module where it fuses the features in four scales. It won the ImageNet Scene Parsing Challenge 2016 and it is a suitable candidate for pixel classification tasks (Bhatnagar 2020).

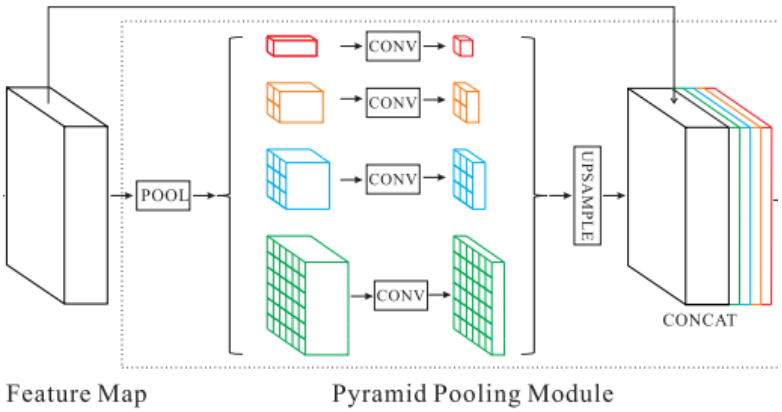

Figure 3 - The PSPNet architecture (Bhatnagar 2020)

Training and classification

For each Deep Learning model, the ArcGis Pro Notebook calculates a learning rate, which is the amount of change applied to the model during each step of the optimization process. It is the most important hyperparameter to tune for a neural network to achieve satisfactory performance, since it controls the rate or speed at which the model learns.

Once the learning rate is calculated, it is fed into the classifier U-NET and PSPNet objects. These objects then use the training data and the learning rate to train a model for several iterations (also known as epochs, typically 20). This model is then exported and used to classify the target image.

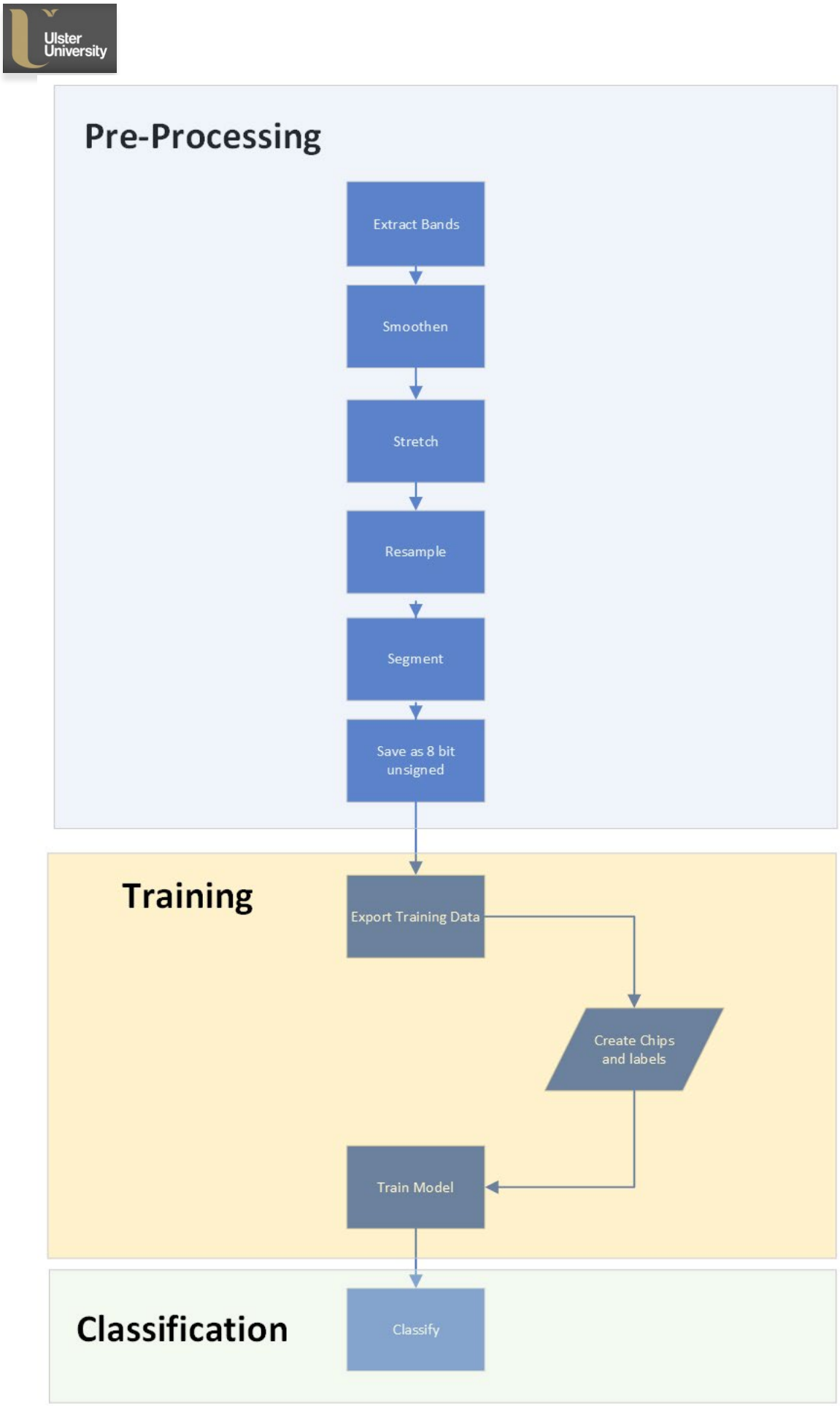
Pre-Processing
Extract Bands
Smoothen
Stretch
Resample
Segment
Save as 8 bit unsigned
Training
Export Training Data
Create Chips and labels
Train Model
Classification
Classify

Figure 4 - Script Workflow

Results

Main outcomes

This ArcGis Pro Notebook creates two classified rasters: one for each of our two models. They are displayed within the Notebook and can be also accessed through the Map view in ArcGis Pro's UI. For our test data, PSPNet did not provide good enough results. U-NET, on the contrary, produced particularly good classification outcomes (with a K value of 80%, according to analysis conducted in EGM722), specially compared to benchmark equivalent data available (such as UK CEH). Particularly interesting was the fact that results were remarkably good even when using the models to classify different land cover areas, or using data captured in separate times or dates (under different light conditions).

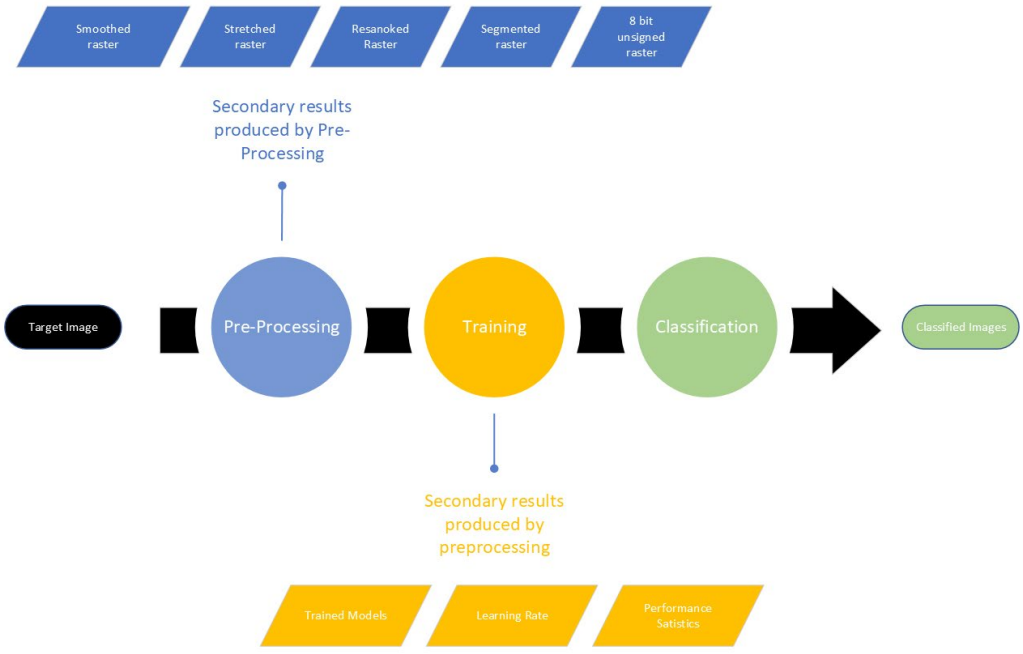

Figure 5 – Results

Figure 6 - Sample image

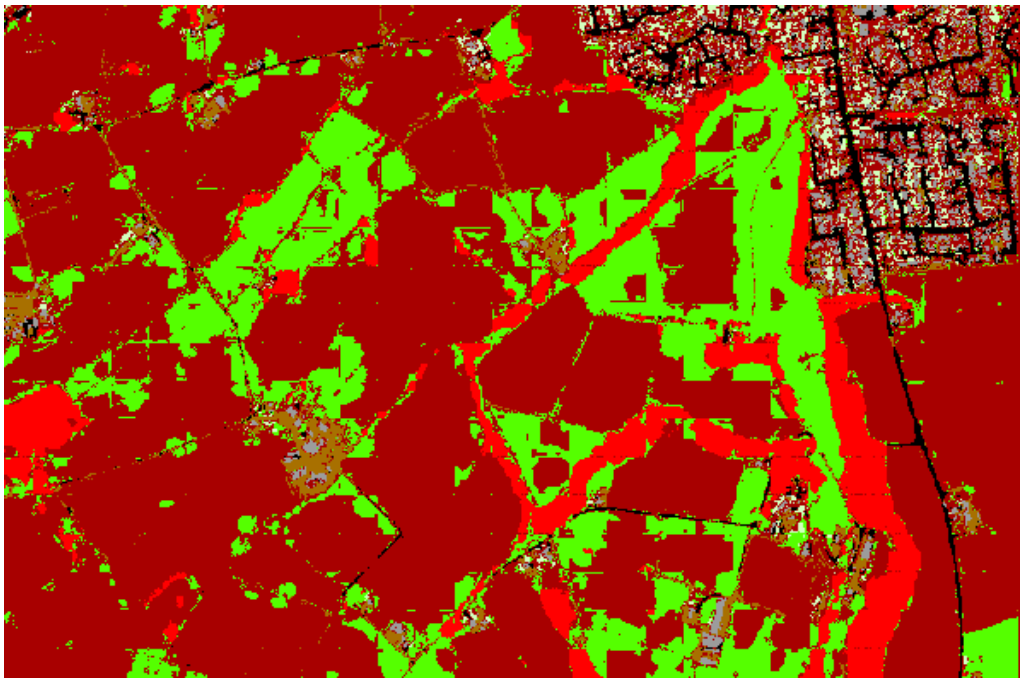

Figure 7 - U-Net Classification of a sample image

Secondary outcomes

Several useful files are generated as a by-product of running the script: the collection of Chips and Labels which are produced by exporting classification samples could be used to train other models. The U-NET and PSPNet

models (.emd files) created to classify our image can be reused in other scenarios. The results of pre-processing our target data, and the segmented image, can all be also employed in different applications. The script also displays accuracy metrics for the model training process.

Troubleshooting

As mentioned in the Installation and Setup section, running the ArcGis Notebook within a ArcGis Pro licensed python environment should make running the script a seamless experience. However, there are several potential issues which the user needs to consider:

- Memory overflow errors can appear if the computer where the script is run lacks GPU capabilities. If memory errors appear, please change the "batch_size" parameter to 4 or 2 – this will slow down processing but will spend less CPU resources.
- The script has been evaluated on an Intel(R) Core(TM) i9 device with 16 cores and 32 GB Ram, with GPU and parallel processing enabled. Running a Deep Learning script through an ArcGis Notebook can take a lot of time and resources (this could be the result of ArcGis Pro by default not splitting tasks among different hardware cores to speed up processing within the ArcGis Notebook python environment). To solve this problem, I enabled parallelism through the multiprocessing python library, successfully running the script a lot quicker (less than 15 minutes). Systems with lower specifications might slow down significantly while running the script – in these environments, it might be advisable to run the script overnight, when the computer does not need to be used, or in a virtual environment.
- The directory structure must be followed i.e. the script uses the paths U:\Project and U:\ProjectData to create different files. If different arrangements are required, the script will need to be changed accordingly i.e., paths will need to be modified according to the user's specifications.

Conclusion

Results using a U-Net model for land cover pixel classification and sparse sampling seem to be consistently better than traditional approaches. Minimal pre-processing and model training was required to obtain results which showed important levels of accuracy.

Using the python environment allowed us to understand each step of the classification process and automatize and customize it according to our requirements. The script is now a tool that can be run by users with no previous knowledge of python or Deep Learning, or by experienced users who want to speed up their daily tasks or procedures. It can be particularly useful to benchmark or compare additional Deep Learning models, or for adding complexity to existing ones: for example, the work of Yan *et al* (2022) on adding the channel attention

module (CAM-UNet) to the original U-Net framework identifies increases of up to 5% on original models (albeit with a higher number of samples) and would be an interesting complement to this ArcGis Pro Notebook.

The possibilities for improvement and progression are immense. For example, shadows were one of the factors which impaired the model overall accuracy, as sometimes they produced surface misclassification. In this respect, the work of Fan *et al* (2019) on Image Shadow Removal Using End-to-End Deep Convolutional Neural Networks shows that using Deep Learning algorithms in python implementations to conduct further pre-processing to remove shadows could greatly improve our image classification models.

References

Bhatnagar, S., Gill, L., Ghosh, B.(2020), Drone Image Segmentation Using Machine and Deep Learning for Mapping Raised Bog Vegetation Communities. *Remote Sensing,* Vol. 12, p. 2602. Available at https://www.researchgate.net/publication/343627992_Drone_Image_Segmentation_Using_Machine_and_Deep_Learning_for_Mapping_Raised_Bog_Vegetation_Communities[Accessed 08/05/2022]

Fan, H., Han, M., Li, J. (2019) Image Shadow Removal Using End-To-End Deep Convolutional Neural Networks. *Applied Sciences*, Vol. 9, No. 5. Available at https://www.mdpi.com/2076-3417/9/5/1009 [Accessed 08/05/2022]

Ronneberger, O., P.Fischer, Brox, T., (2015), U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Vol. 9351, p. 234-241. Available at http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a [Accessed 08/05/2022]

Taghanaki, S., Abhishek, K., Cohen, J., Cohen-Adad, J., Hamarneh, G. (2021) Deep semantic segmentation of natural and medical images: a review. *Artificial Intelligence Review*, Vol. 54. Available at https://arxiv.org/abs/1910.07655 [Accessed 08/05/2022]

Yan, Chuan / Fan, Xiangsuo / Fan, Jinlong / Wang, Nayi (2022), Improved U-Net Remote Sensing Classification Algorithm Based on Multi-Feature Fusion Perception. *Remote Sensing*, Vol. 14, No. 5. Available aat https://www.mdpi.com/2072-4292/14/5/1118 [Accessed 08/05/2022]

Yao, W., Zeng, Z., Lian, C., Tang, H. (2018), Pixel-wise regression using U-Net and its application on pansharpening. *Neurocomputing*, Vol. 312, p. 364-371. Available at https://www.sciencedirect.com/science/article/pii/S0925231218307008. [Accessed 0805/2022]

Zandbergen, P. A. (2020) Advanced Python Scripting for ArcGIS Pro. California: ESRI Press.

Zandbergen, P. A. (2020) Python Scripting for ArcGIS Pro. California: ESRI Press.