# 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 been left unexplored: using Machine Learning and Deep Learning to solve this problem. In particular, I became interested in applying Deep Learning in sparse sampling scenarios, as some of the literature has shown ().

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 carry out 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 carried out 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.

Summarising, 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 while doing EGM725 and consists of providing a tool to automatise land cover image 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 in:

The best way to use this Classifier tool is to open the notebook EGM722Classifier.ipynb from a new ArcGis Pro project. Some of the deep learning dependencies used here require the ArcGIS Image Analyst extension, ArcGIS Spatial Analyst extension or the ArcGIS 3D Analyst extension, so a licensed copy of ArcGis Pro needs to be open while running the notebook. This repository includes the required .yml file (created using conda env export > environment.yml) to duplicate the python environment, but the before mentioned licensing constrains still apply.

This project was carried out using ArcGis Pro v2.9 with ESRI's Deep Learning libraries 2.9 and a functioning dedicated python environment.

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

The project uses training data obtained from OSNI's own 4-band orthoimagery (spatial resolution of 40cm, and red, green, blue and NIR bands). This training data has been exported using ArcGIS Pro for the models being analysed. The exported feature class file is called **trainingsamples.shp**.

To help organising the project, I mapped a new U: drive, where I created a ***Project*** folder for public GIT files, and a ***ProjectData*** folder for OSNI copyrighted files and script output files to improve readability and understanding of the project. User of this script should replicate this structure or adapt the script accordingly.

Finally, the script creates a significant number of files and folders, and therefore it is necessary to ensure that the machine which is running it has enough memory resources (a minimum of 5 GB free memory is recommended).

Finally, I realised that using the python environment through a notebook had many advantages, but it run a lot slower than through ArcGIS Pro user interface menus. I believe that's the result of ArcGis Pro using parallelism (splitting tasks among different hardware cores to speed up processing). I therefore also employed parallelism through the multiprocessing python library, successfully running the process 16 times quicker (the number of my cores) than before.

# Methods

## Pre-Processing

The Classifier starts by pre-processing the raster file being analysed to optimize it before applying Deep Learning algorithms. The first stage of the pre-processing process consists of smoothing the target image using a sharpening 5 x 5 filter. Secondly, the image is stretched using a Sigmoid function (this highlights moderate pixel values while maintaining sufficient contrast in the perimeter). Then, the resulting raster is resampled and segmented. Segmentation is a key process as it changes the characteristics of the image to facilitate classification. The parameters used for this process followed those used by my supervising team at OSNI, and featured relatively high spectral detail (18 out of 20), low spatial detail (3) and a minimum segment size of 25 pixels.

The final pre-processing step is to verify that the raster being analysed is a 3 band, 8-bit unsigned file. Deep Learning models run in the ArcGis Pro environment must have a maximum of 3 bands, so the 4th band (near infrared) was removed, and the raster was saved as a 8-bit unsigned image.

## Export Training Samples

Training data was gathered through ArcGis Pro and exported using Classified Tiles as meta data format into a feature class. This feature class was in turn used by our classifier script 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 category. Our training samples identified 8 categories of land cover, which matched a subgroup of those used by UKCEH and XXXX.

## Data preparation

Subsequently, our training data (i.e. our Chips and Labels) was read and inspected by the ***prepare\_data()***method from arcgis.learn. This method produced the structures required for training and validation, and the specified changes relating to data augmentations, chip and batch size, and train-validation split percentage – essentially, it set the right hyper parameters to create a good model.

## Classifiers: U-NET and PSPNet

The script evaluates 2 Deep Learning models used for pixel classification implemented in ArcGis Pro using PyTorch: UNET and PSPNet.

UNET 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 relatively little pre-processing compared to other image classification algorithms.

As we can see in Figure 1 below, the U-Net architecture consists of a series of convolutions where a filter is applied (also known a kernel or feature extractor) on input data and the results are (Figure 2 and Figure 3) in turn, pooled, retaining only the important features (and, **crucially**, reducing the resulting image). 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 1). 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 are very valuable and scarce resources).

Chart, box and whisker chart

Description automatically generated

|  |  |
| --- | --- |
| A picture containing text, electronics, cellphone  Description automatically generated | Diagram  Description automatically generated with medium confidence |
|  |  |

PSPNet (which stands for Pyramid Scene Parsing Network) incorporates global features through dilated convolutions for scene parsing and classifications as shown in Figure . It includes a Pyramid Pooling Module where it fuses the features in four scales. It won the ImageNet Scene Parsing Challenge 2016 and thus is a good candidate for this task.

Diagram, schematic

Description automatically generated

|  |  |
| --- | --- |
| A picture containing businesscard, envelope  Description automatically generated | Shape  Description automatically generated |

## Training Models

Both PSPNet and U-Net classifiers were training using our training data. For each of them, the script calculates a learning rate. The learning rate is the amount of change to the model during each step of the optimization process. It is the most important hyperparameter to tune for a neural network in order to achieve good 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 during several iterations (also known as epochs, typically 20). This model is then exported and used to classify the target image, and can also be used in other scenarios and rasters.

## References

<https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1>

<https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>

https://developers.arcgis.com/python/guide/how-pspnet-works/

https://www.researchgate.net/profile/Saheba-Bhatnagar/publication/343627992\_Drone\_Image\_Segmentation\_Using\_Machine\_and\_Deep\_Learning\_for\_Mapping\_Raised\_Bog\_Vegetation\_Communities/links/5f34f72092851cd302f159b4/Drone-Image-Segmentation-Using-Machine-and-Deep-Learning-for-Mapping-Raised-Bog-Vegetation-Communities.pdf?origin=publication\_detail