A Study on Flood Detection Using Satellite Imaging and Deep Semantic Segmentation

Luke Van Der Veen

Introduction

- Using deep learning techniques to identify flooding in satellite imagery
  - Semantic segmentation
- Flooding is a costly disaster

Aims

- This study aimed to do the following
  - Provide current models' adaptability for other CNN backbones
  - Trialling a different CNN architecture on current models

Related works

Interpretable Deep Semantic Segmentation, Zhang (2021)	• AI decision making is difficult to interpret • Provides steps to understanding flooding classifications • Uses a prototype and decision-making architecture
WORLDFLOODS	• Satellite image database
ML4Floods	• Semantic segmentation backbone • Built for flood identification in satellite images

Methodology

- The project was split into the following stages
  - Understanding ML4Floods
  - Building the IDSS architecture into ML4Floods
  - Refining hyperparameters
  - Changing the CNN backbone

Architecture

- ML4Floods provided backbone
- IDSS implementation
  - Raw and latent feature extraction
  - Prototype layer
  - Decision making layer

Dataset

- WORLDFLOODS holds 300Gb satellite image, ground truth pair dataset
- Datasets for training, validation and testing
- Each image holds 13 dimensions
- Version 1 of the dataset utilised

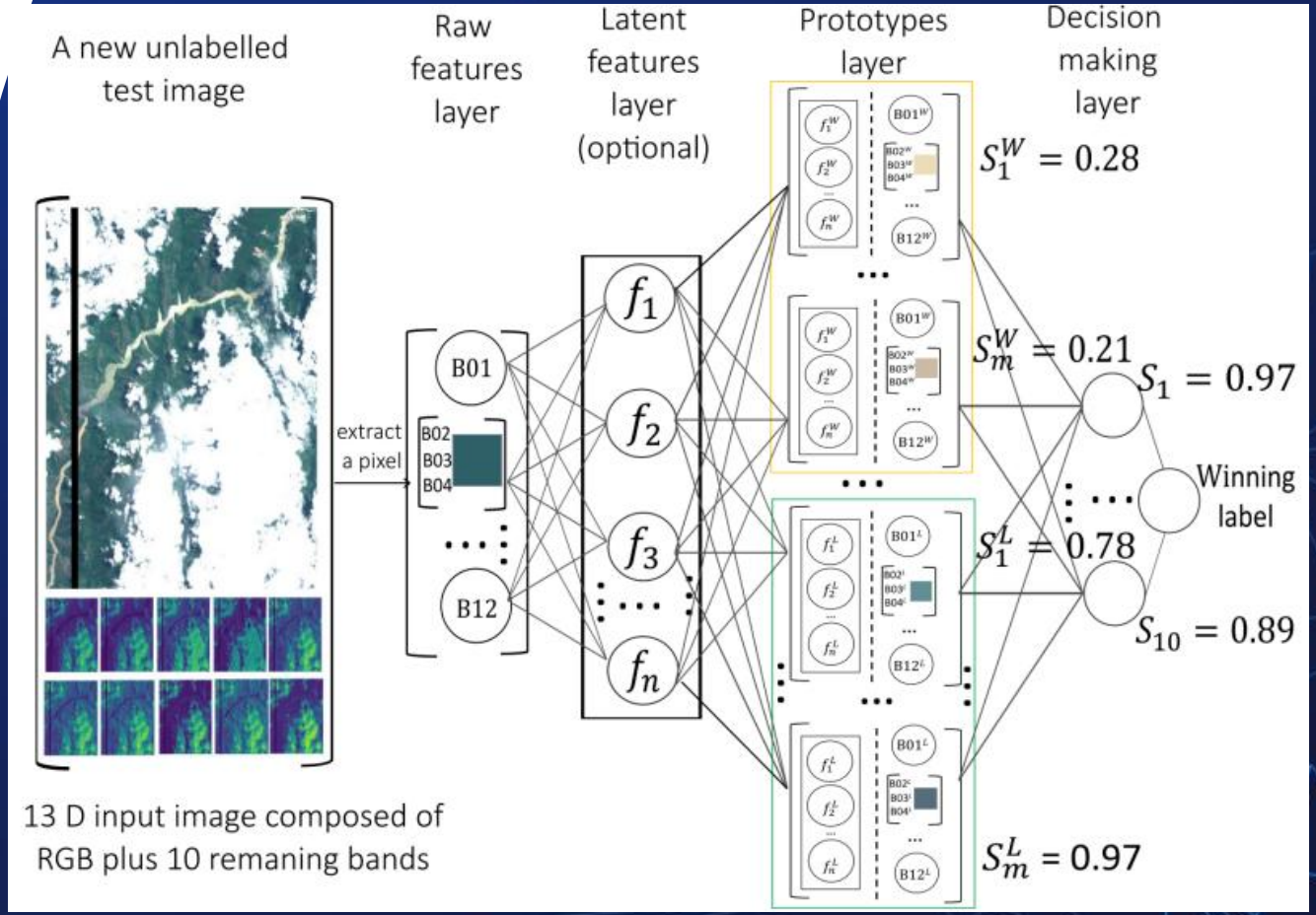
Name	Description
B01	Coastal aerosol
B02	Blue
B03	Green
B04	Red
B05	Vegetation red edge
B06	Vegetation red edge
B07	Vegetation red edge
B08	NIR
B8A	Narrow NIR
B09	Water vapour
B10	SWIR
B11	SWIR
B12	SWIR

Understanding ML4Floods

- ML4Floods provides tutorials
  - Training models
  - Inferencing models
- Due to lack of depth in these
  - A lot of familiarising was required
- Through which many discoveries were made
  - Image tiling, allowed for removal of cropping
  - Easy CNN plugging
  - Config settings to easily transform batches

IDSS

- This method requires implementation of 2 additional layers
  - Prototype layer
  - Decision making layer
- Raw and latent features
  - Raw: feature output from the CNN
  - Latent: feature output from penultimate layer of CNN



IDSS Prototype Layer

- This layer clusters features into prototypes using MiniBatch K-means
- Set number of prototypes per class
  - Water, Land, Cloud
- Prototypes updated every 50 batches
  - Typically, out of 1000 batches

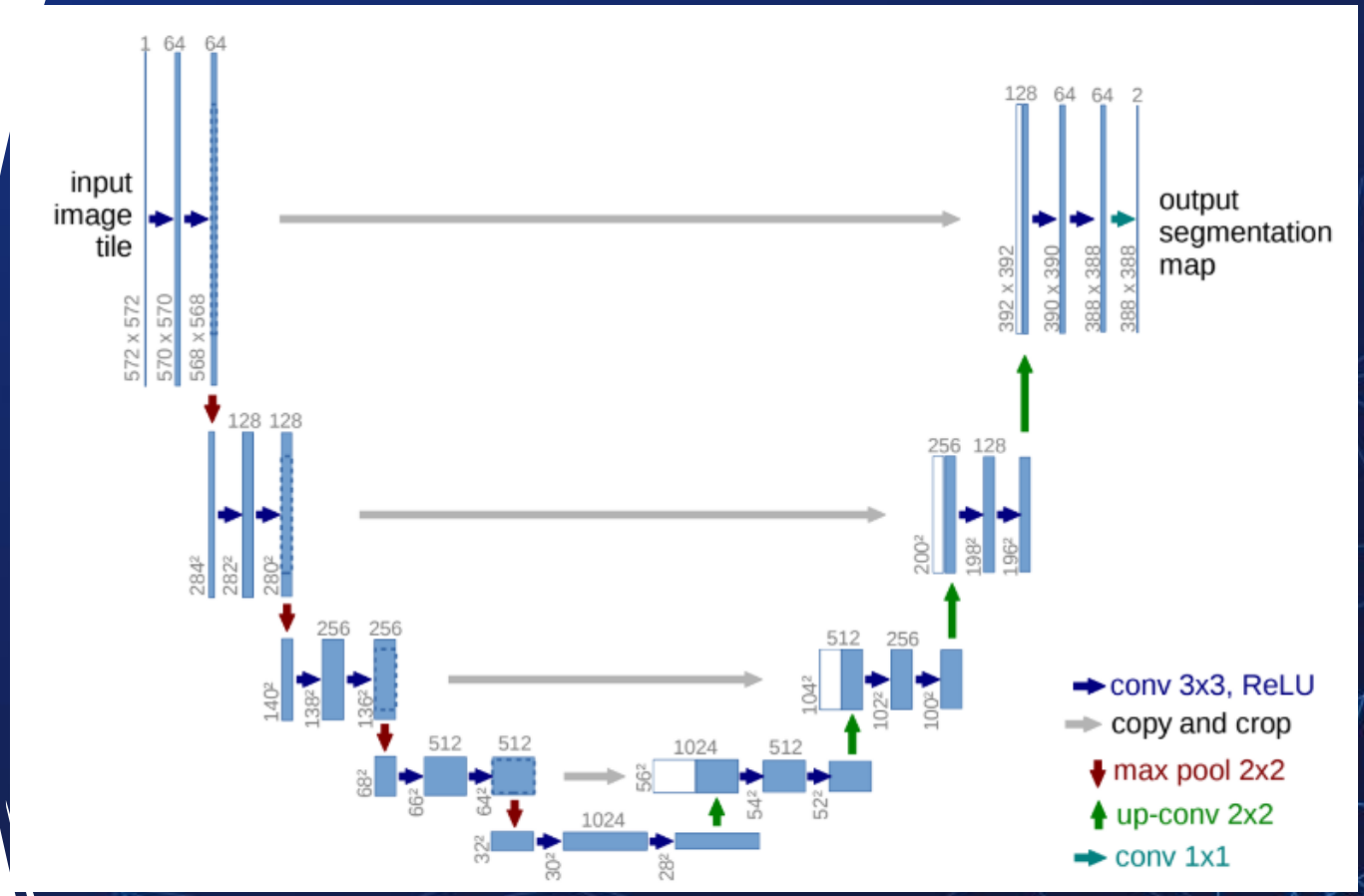
IDSS Decision Making Layer

- Assign class label to unclassified pixels
- Based on feature and rule-based similarity
- If a pixel reaches a certain threshold
  - Assign to class
- If pixel similar to another
  - Assign to class

If (SWIR1 > 0.7) OR (SWIR2 > 0.8)
AND (Blue > 0.3) AND (Red > 0.3), assign Cloud

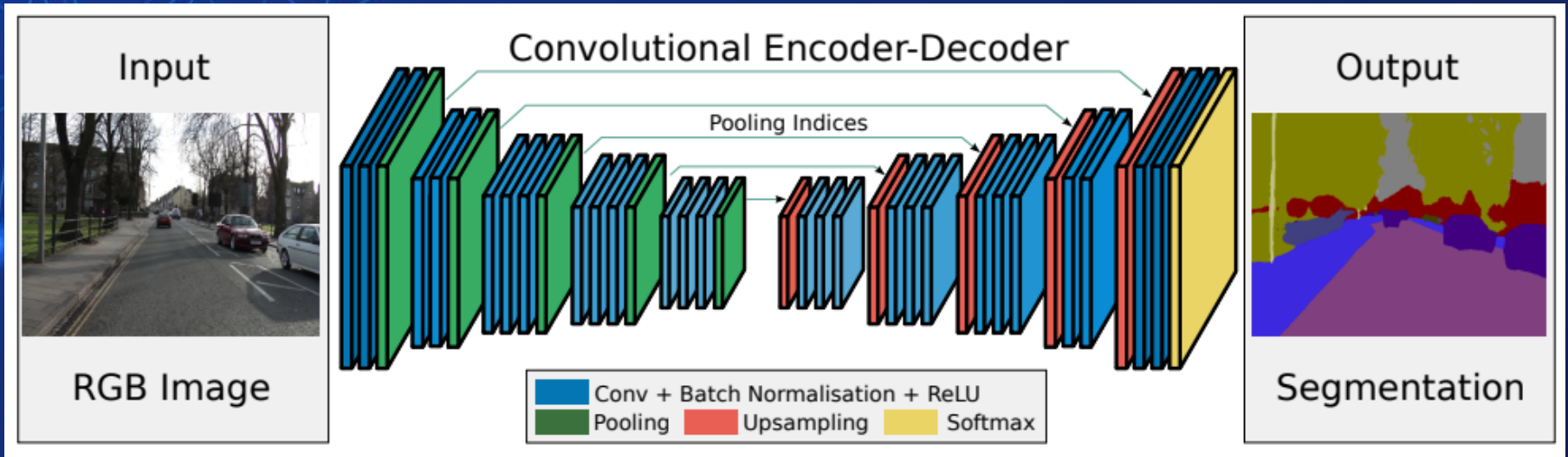
CNN Models: U-net

- CNNs were utilised for raw and latent feature extraction
- Standard CNN for ML4Floods
  - This is the initially utilised U-net model for the project
- Standard CNN for IDSS
- Provides the baseline
- Typically, highly accurate segmentation



CNN Models: Seg-Net

- Chosen adapted CNN model



- Typically, more memory efficient and computationally faster
- ML4Floods allows for easy adaptation
  - Seg-Net model adapted from a PyTorch segmentation project

U-net differences

- No skip connections
- Pooling indices stored at each encoder layer
- Indices reused in decoder for upsampling
- Speeds up segmentation

Development issues

Time

- Each run would take x amount of time depending on batch size etc
- Coupled with hyperparameter tunning this took a long time
- Full dataset too large

Resources

- A 3070 ti GPU is only able to do so much processing
- Limited memory
  - Limited batch size
  - Dimension processing
  - Number of prototypes

Development issues

ML4Floods complexity

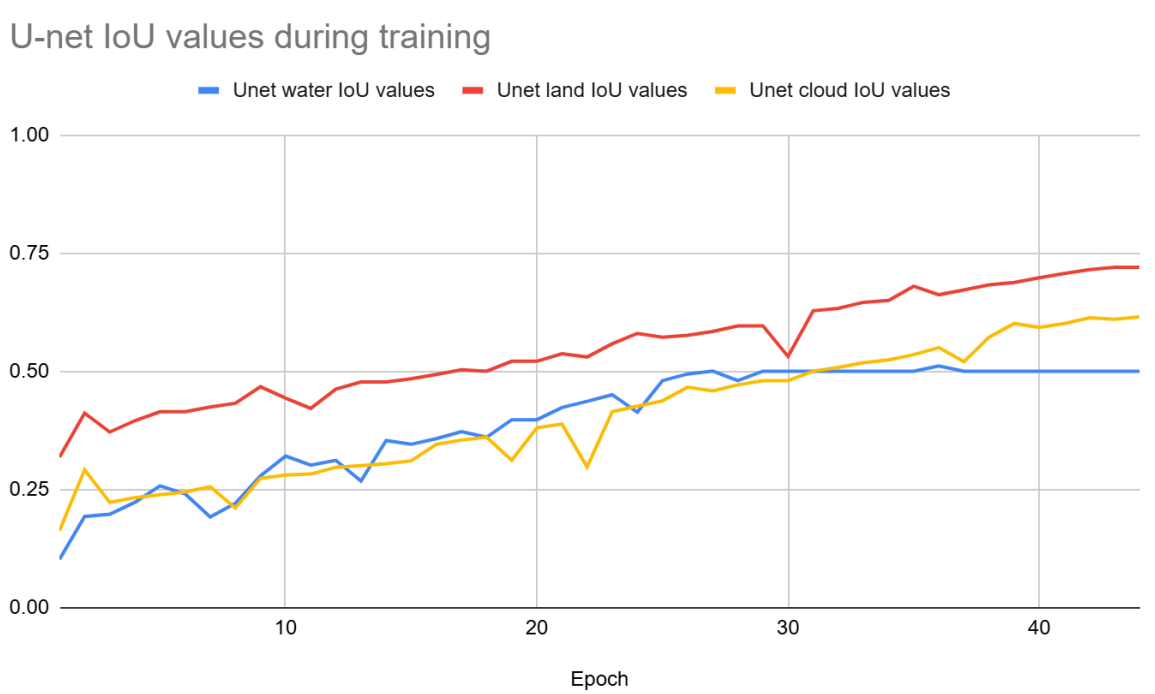
- The architecture is incredibly complex and in depth
- Little documentation on architecture innerworkings
- Unimplemented sections

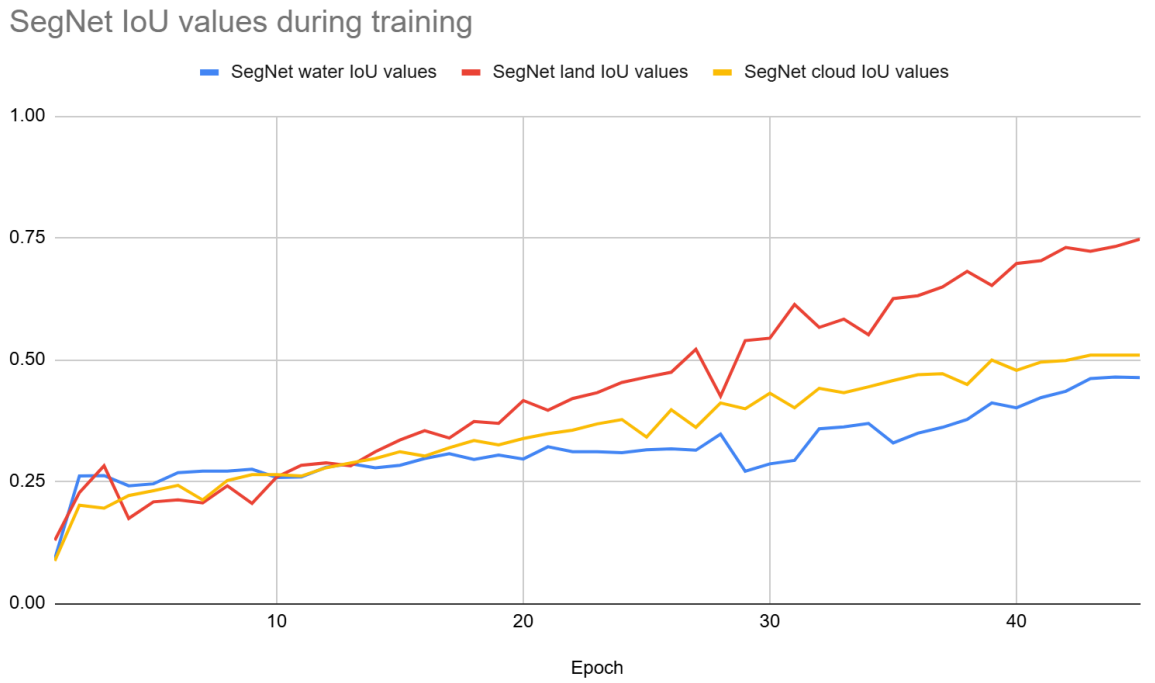
Understanding IDSS

- Complex structure
- Prototype layer was reduced
  - 600 utilised
  - Original IDSS used1500

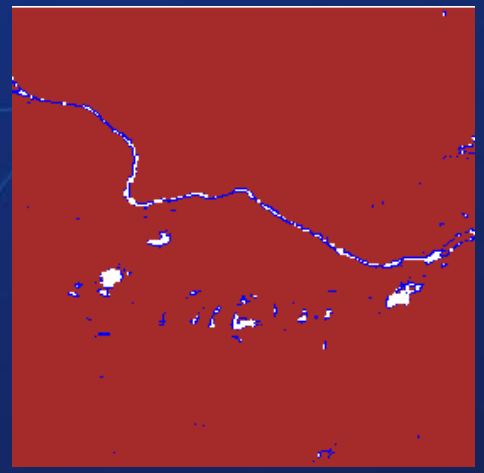
Findings

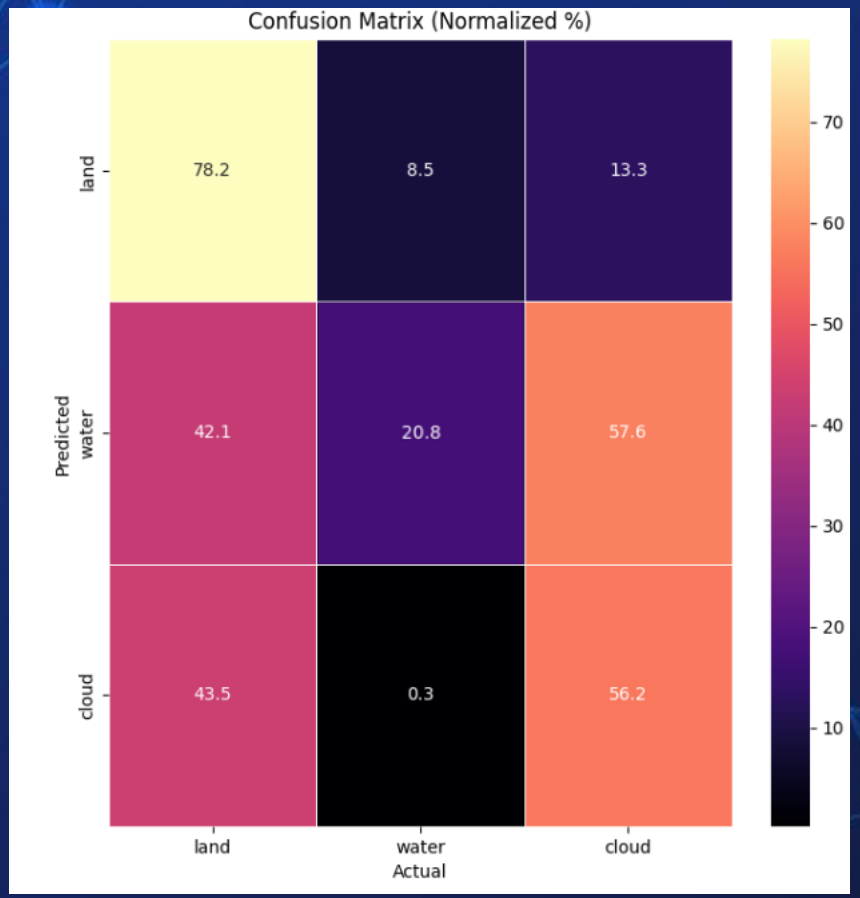
- Water IOU during training
  - U: 0.501
  - S: 0.463
- During inference
  - U: 0.414
  - S: 0.271
- Recall also recorded
- These were some of the best scores
  - Many runs produced near negative values

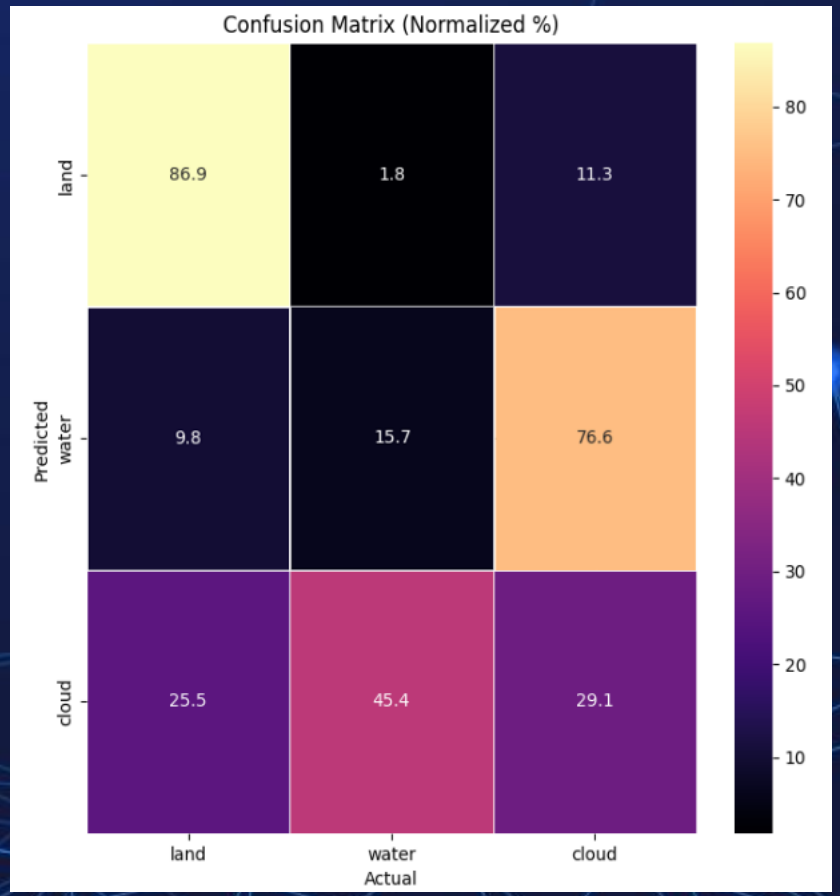




Visual Findings







Discussion

What could be improved

- Better resources
  - Would allow for 1500 prototypes
  - Use a larger dataset
- More time
  - Allow for more epochs
  - More training time
  - More finetuning

What do the findings infer

- Model was learning well
- Inferencing was weaker than training
- Overfitting

Conclusion

- IDSS can easily adapt a new CNN
- Further testing required
- Other CNNs could offer useful benefits

Questions?