Machine Learning Ⅱ

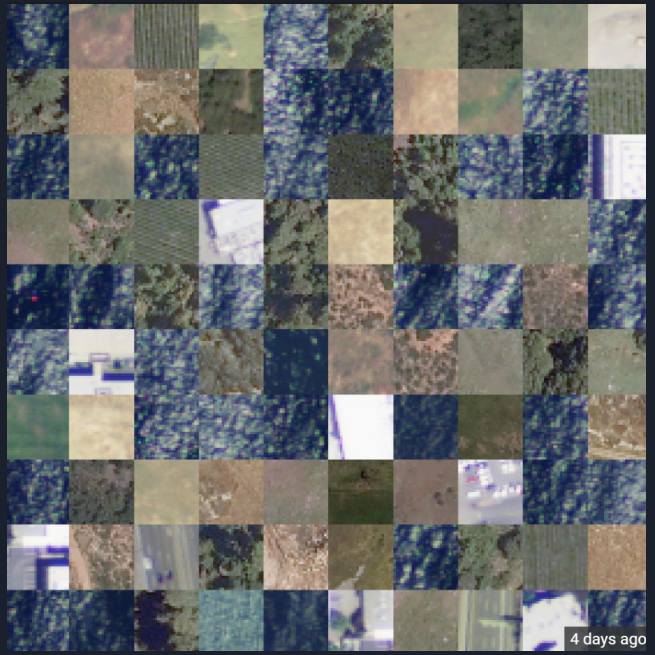
Deep Learning of Land Cover Classification with Pytorch

Qianwen Liu
Mark Bana
Thanh Nguyen

The Airborne Dataset

Our Dataset

Covering different landscapes like forests, water, mountains, etc;



Why Land Cover Classification?

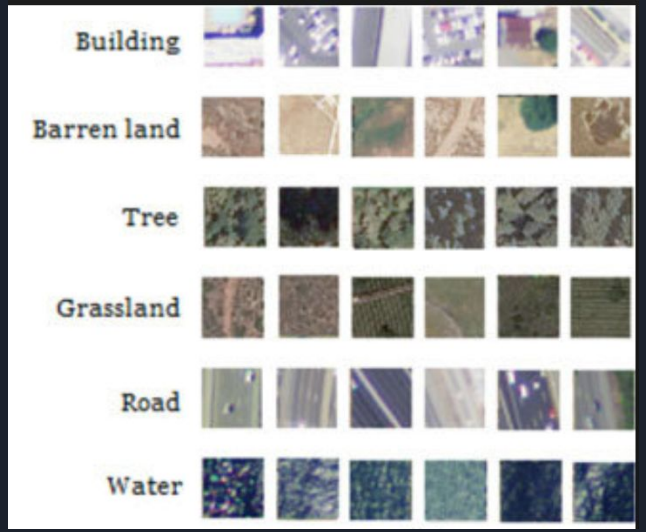
For years scientists across the world have been mapping changes in the landscape to prevent future disasters, monitor natural resources, and collect information on the environment. When land covers change, our health, economy, and environment can all be affected.

The Airborne Dataset

- **Size:** 2.42 G
  405,000 images
  Training Set: 324,000
  Testing Set: 81,000
  Image size: 28 X 28 X 4

- **6 Classes**

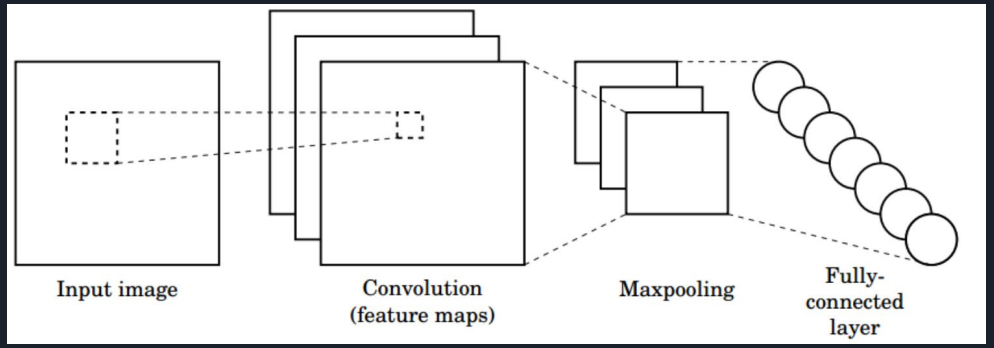
	Train	Test
Building	14923	3714
Barren land	73397	18367
Trees	56809	14185
Grassland	50347	12596
road	8192	2070
water	120332	30068
	324000	81000



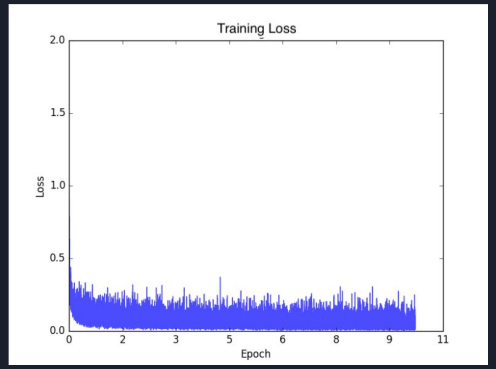
Project Overview

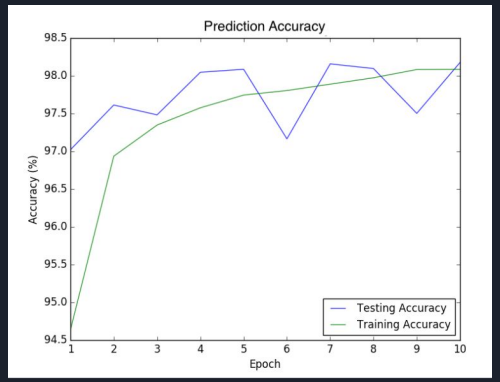
- Data Validation - Implemente Network - Vary Parameters - Performance Metrics

Project Overview - **Implementation of Convolution NN with Pytorch**



Project Overview - Network Configurations and Parameters

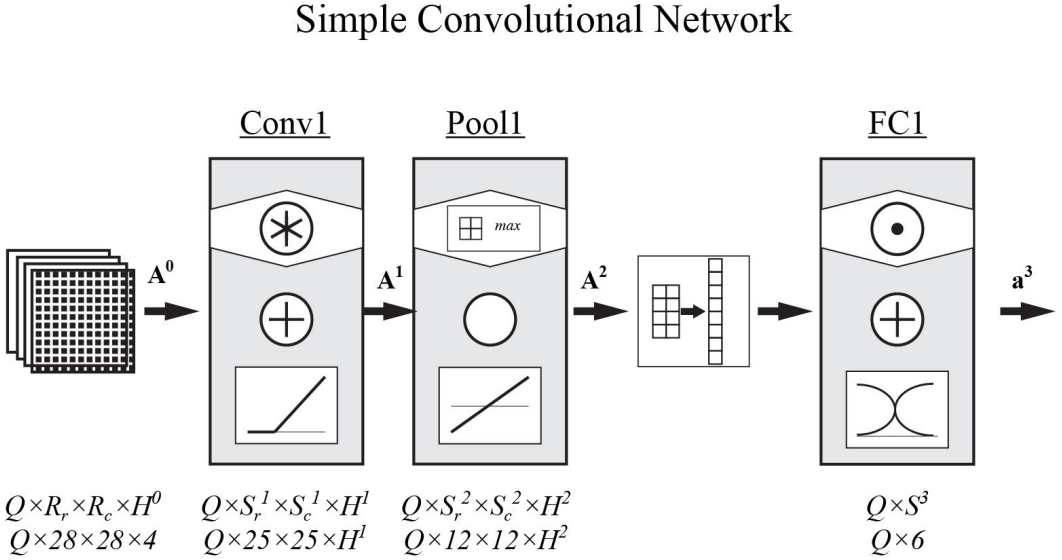




		QuasiAlex 1	QuasiAlex 2	Simple CNN 5	Simple CNN 8	Simple CNN Momentum 1	Simple CNN Momentum 2
Network Parameters	Learning rate	0.01	0.005	0.001	0.005	0.005	0.005
	Momentum	0.9	0.95			0.8	0.9
	Weight Decay	0.0005	0.0005				
	Epochs	10	10	10	10	10	10
Network Configuration	Convolutional Layers	5	5	1	1	1	1
	Pooling Layers	1	1	1	1	1	1
	Linear Layers	3	3	1	1	1	1
Result	Training time(s)	3243.2	3350.1	2744.9	2878.8	2764.9	2753
	Accuracy rate(%)	97.18	96.76	96.71	97.30	98.06	98.18

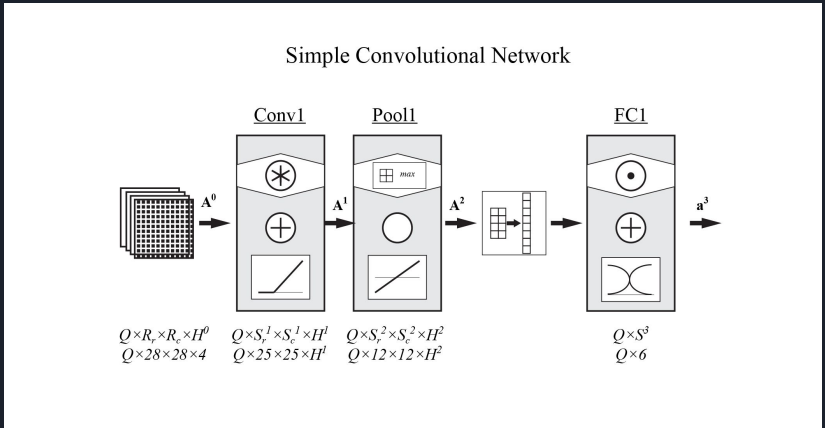
Network Architecture

Simple Configuration



Network Architecture

Simple Configuration



- Convolution kernel size: 4 x 4
- $H^1 = 4, 8, 16, 32$
- Pooling layer kernel size: 3 x 3, stride 2

Training Algorithm

- Output layer, *softmax* transfer function:

$$a = \frac{e^n}{\sum_{i=1}^{6} e_i^n}$$

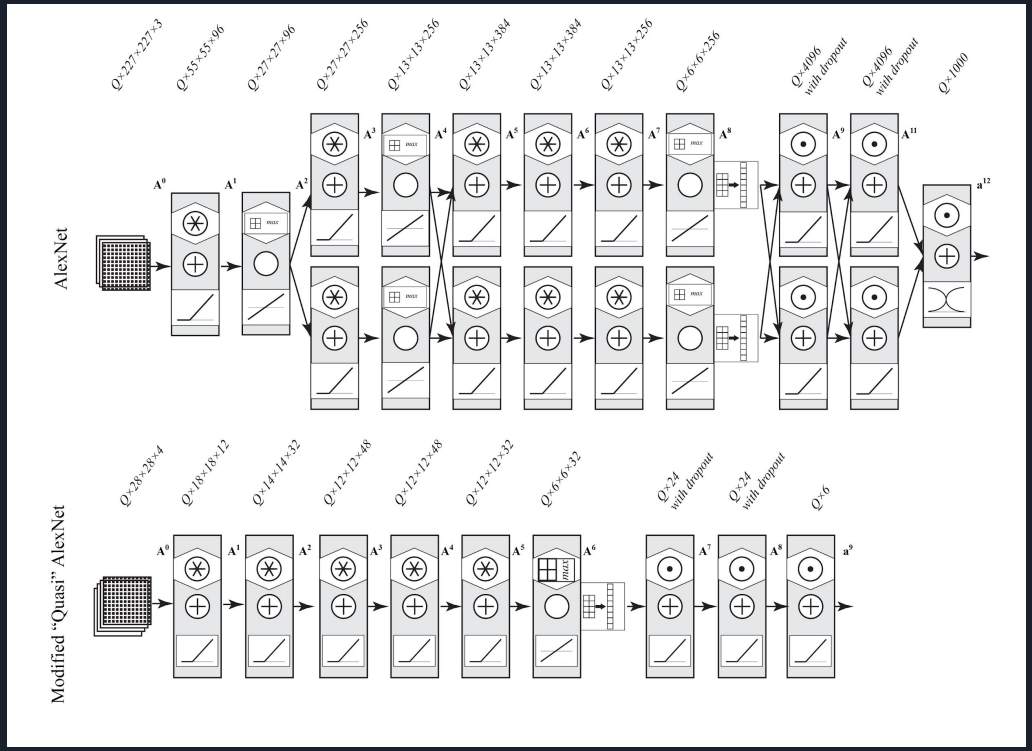
- Loss function, *cross-entropy*:

$$\hat{F}(x) = -\sum_{i=1}^{6} t_i \log(a_i)$$

- Optimization method: Gradient Descent with mini-batches
  - Batch sizes 10, 100, 1000
  - Included momentum on some trials

Network Architecture

Complex Configuration



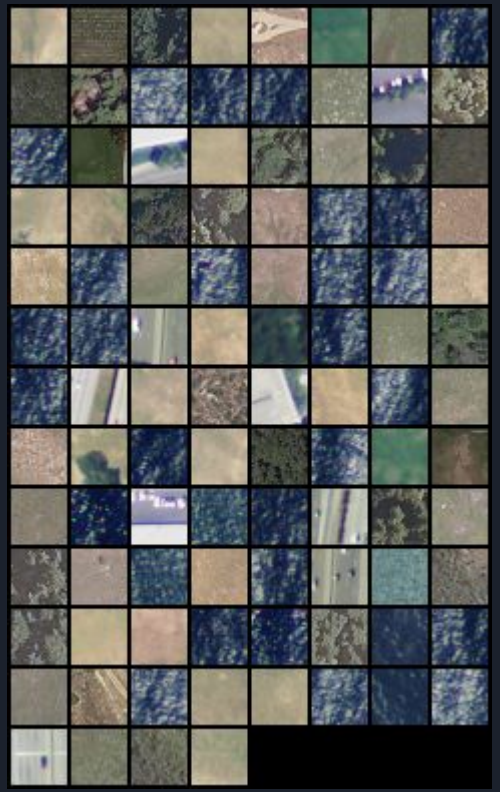
Experiment Setup

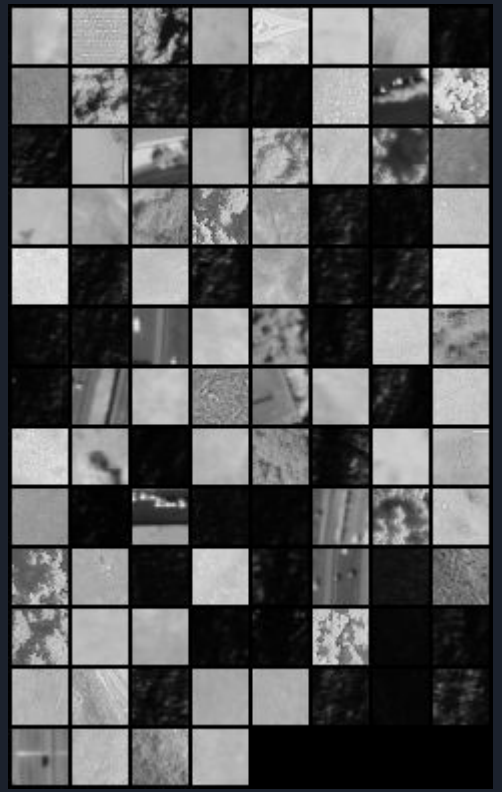
- Data Collection
  - Kaggle API
- Data Integrity
  - Number of entries
  - No null values
  - 0-255 for X and 0/1 for Y
  - Shuffle data to avoid single label for a mini batch
- Network Implementation
  - Extensive documentation and flexibility
  - Pytorch Module and cuda
  - Mini-batches optimization
- Parameter Update
  - Subset testing
- Performance Evaluation
  - Over fitting
  - Accuracy rates vs Training time

Result

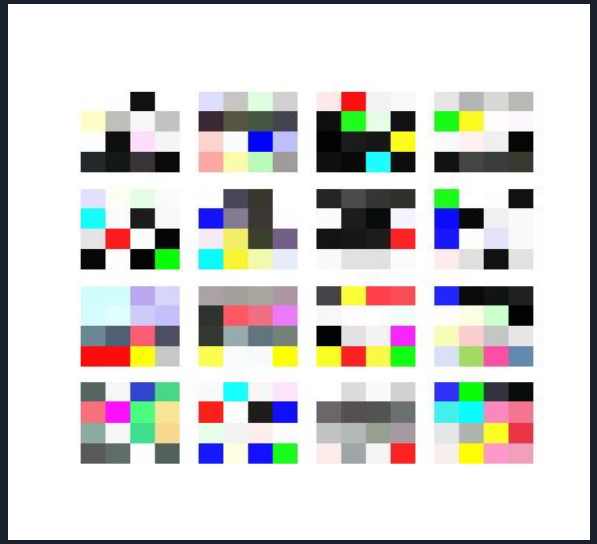
		QuasiAlex 1	QuasiAlex 2	Simple CNN 5	Simple CNN 8	Simple CNN Momentum 1	Simple CNN Momentum 2
Network Parameters	Learning rate	0.01	0.005	0.001	0.005	0.005	0.005
	Momentum	0.9	0.95			0.8	0.9
	Weight Decay	0.0005	0.0005				
	Epochs	10	10	10	10	10	10
Network Configuration	Convolutional Layers	5	5	1	1	1	1
	Pooling Layers	1	1	1	1	1	1
	Linear Layers	3	3	1	1	1	1
Result	Training time(s)	3243.2	3350.1	2744.9	2878.8	2764.9	2753
	Accuracy rate(%)	97.18	96.76	96.71	97.30	98.06	98.18

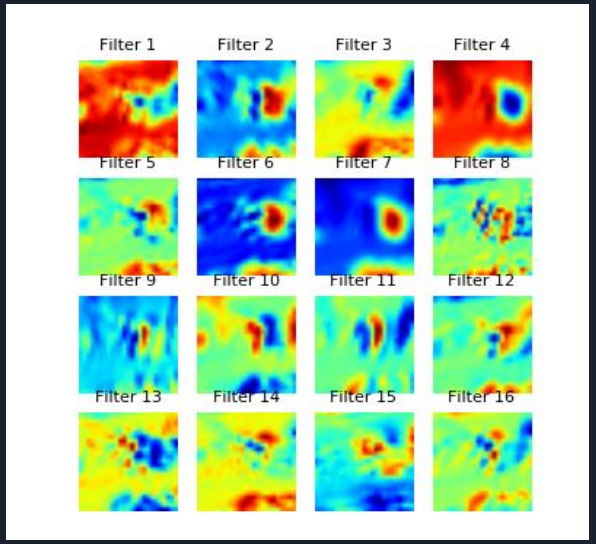
Result



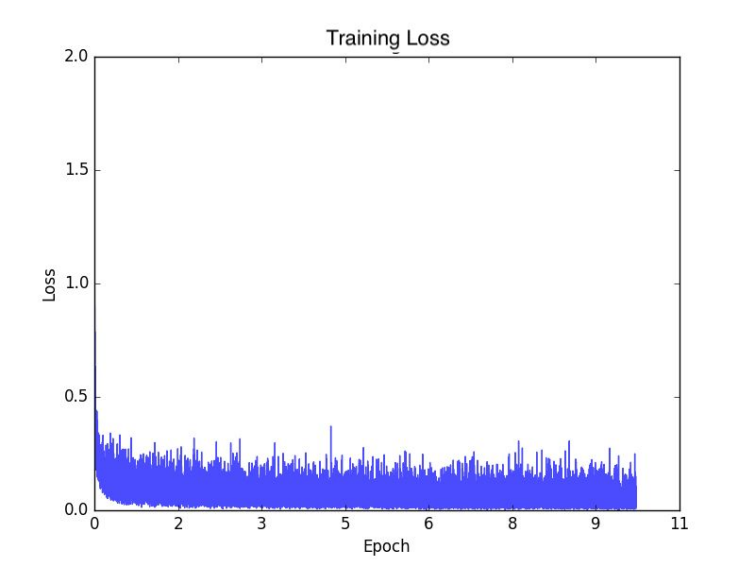


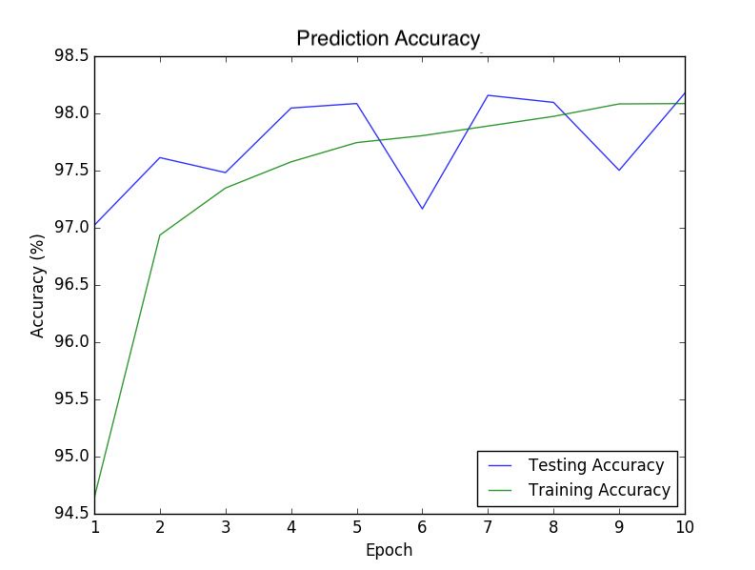
Result




Filter 1
Filter 2
Filter 3
Filter 4
Filter 5
Filter 6
Filter 7
Filter 8
Filter 9
Filter 10
Filter 11
Filter 12
Filter 13
Filter 14
Filter 15
Filter 16

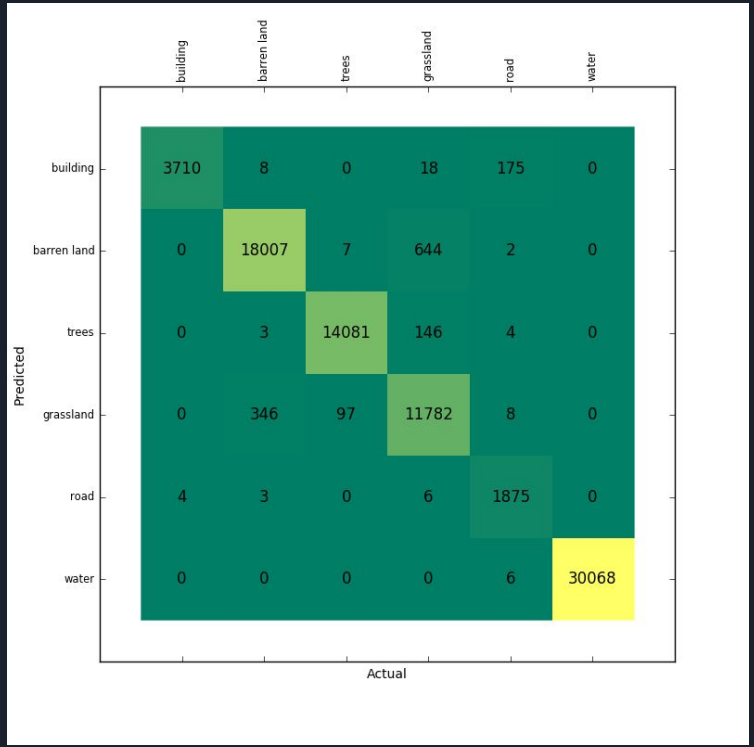
Result





- Average loss: 0.06
- Accuracy rate: 98.18%

Result



- Water: 100%
- Building: 99.89%
- Trees: 99.27%
- Barren land: 98.03%
- Grassland: 93.53%
- Road: 90.58%

Recommendations

1. Dirty data? Overlapping, sub-class?

2. Large images?

3. To reduce the bias in the confusion matrix, we can try tuning the decision boundaries in a proper way in certain classes.

4. Train RGB and INR separately?

Q & A