Predicting population density in coastal areas using satellite imagery

Authors:

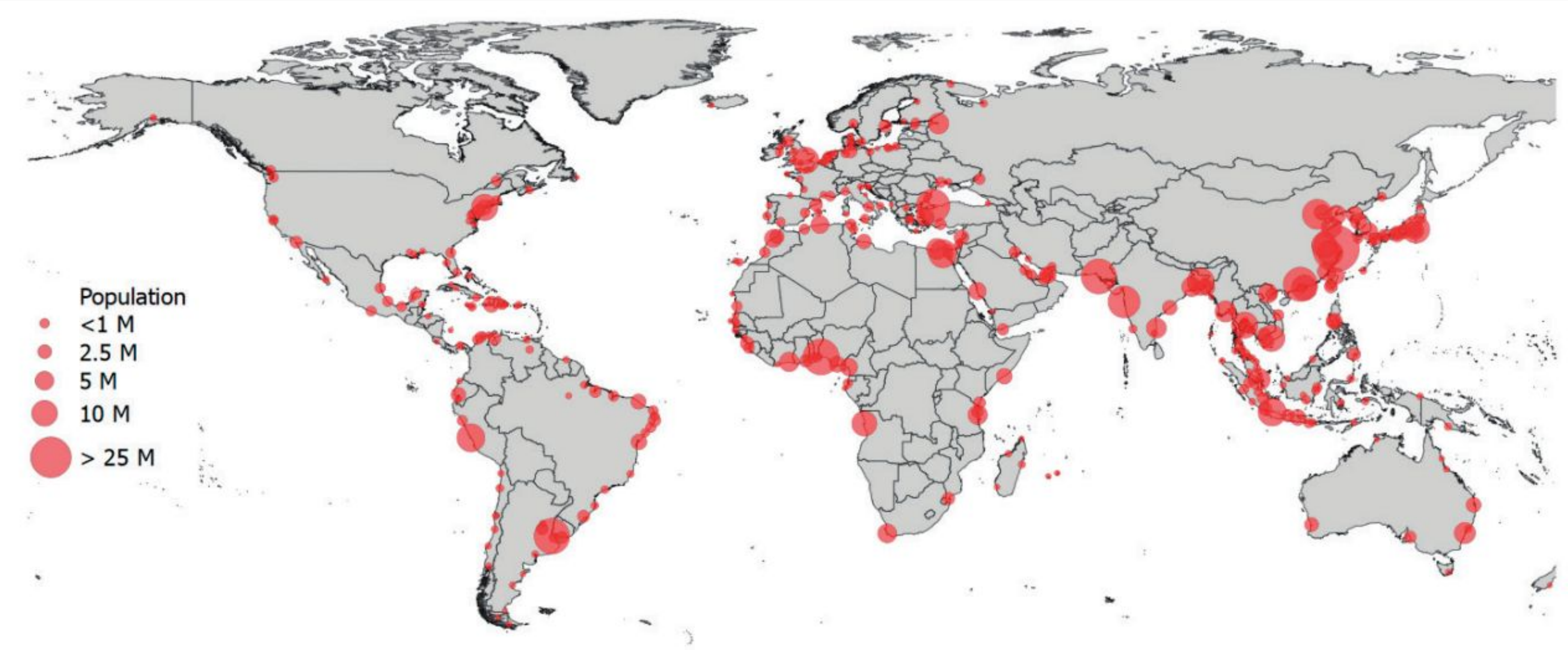
Ioanna Bertsia Kanatouri
Tinus Blæsbjerg
Laura Gómez
Mari Knudson
Marc Peradalta Negre

*All authors contributed equally to this project

Motivation & Objectives

Motivation: Population living in coastal regions

"Between 750 million and nearly 1.1 billion people globally live in the 10m LECZ, with the variation depending on the **elevation, population data sources** and **differing population classifications**." - MacManus et al, 2021.



King, 2022: Figure 1

Objective

We aim to reduce uncertainty in coastal population estimates using satellite imagery and CNN models.

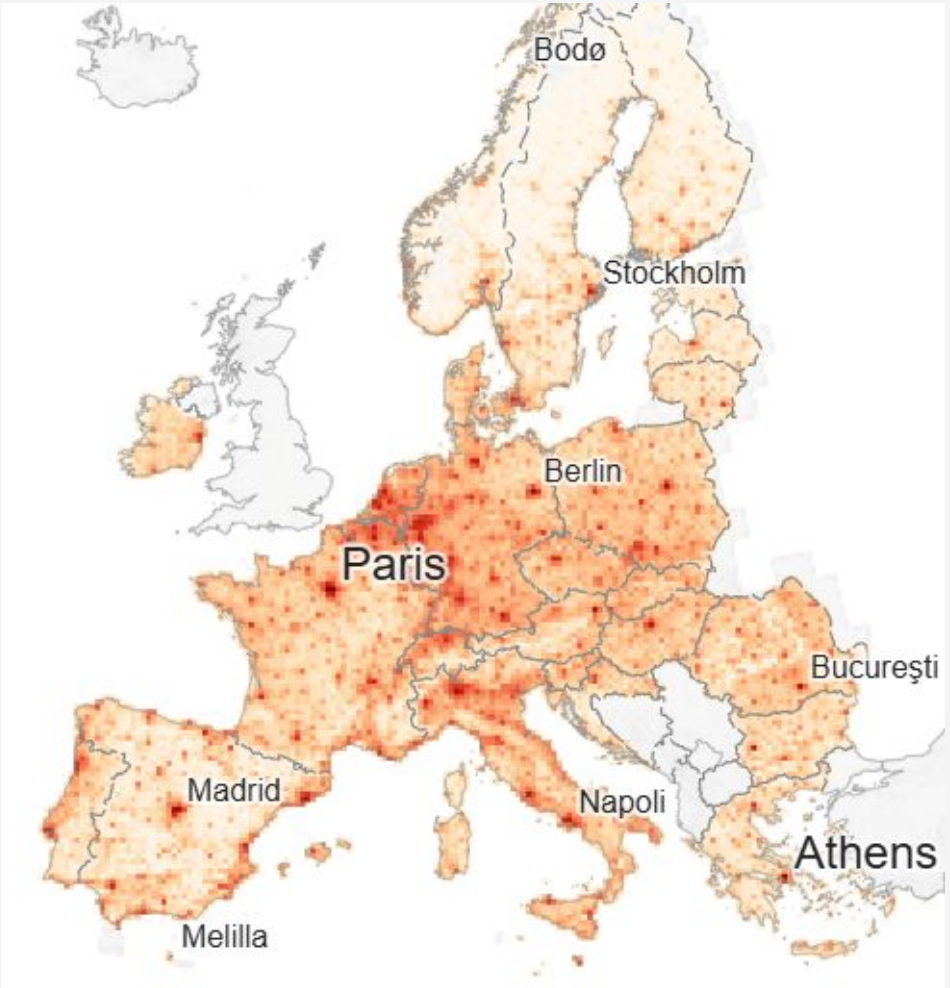
In particular, we train regional models using images from Spain and Malta separately, and evaluate their performance within and outside its training region. Additionally, we train and compare a European model using data from several regions. All models are trained using ResNet-50 and PyTorch CNN architectures.

We hypothesize that regional models will be more accurate locally, while the global model may be more adaptable and scalable across regions.

Datasets

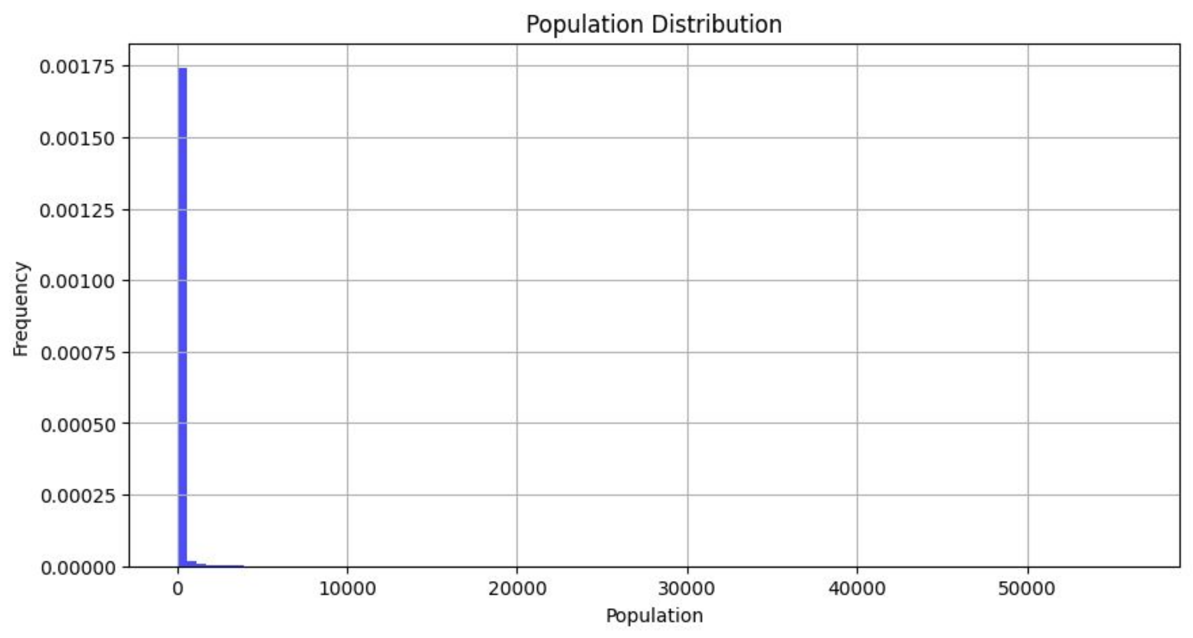
Population dataset

- **Data source:** Eurostat census grid 2021.
- **Tile Resolution:** 1x1km.
- **Assumptions:**
    - d(tile,coast)<=5km from the coast.
- **File size**: .parquet file, ~100MB



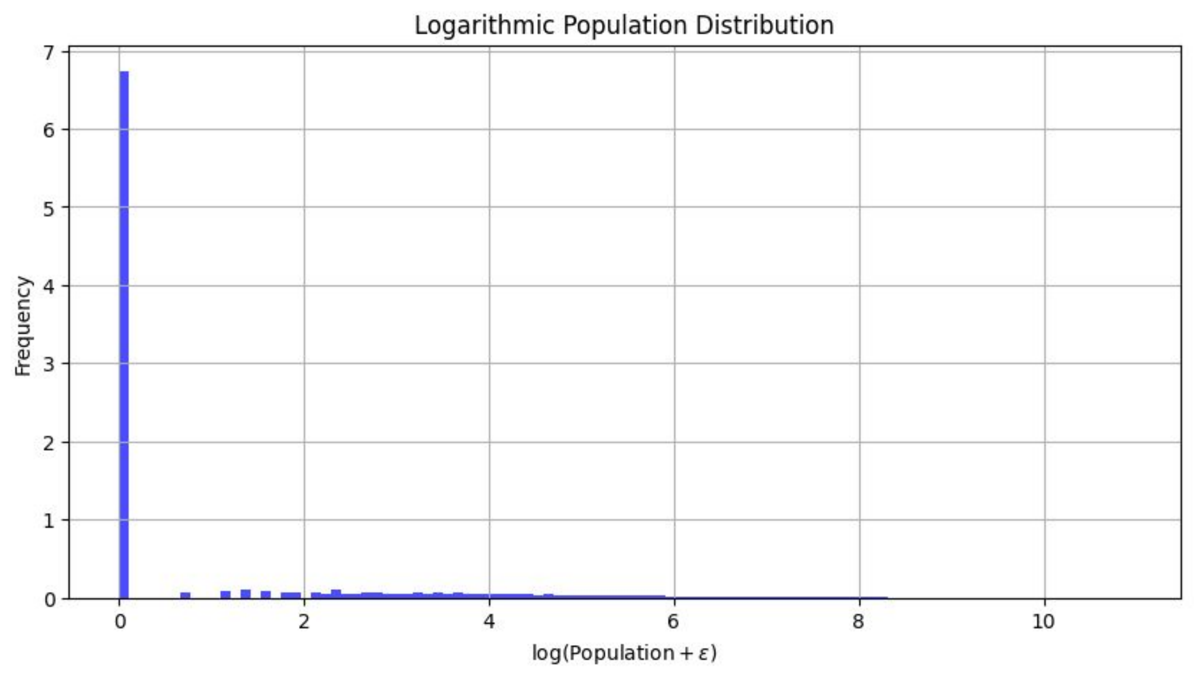
Population dataset

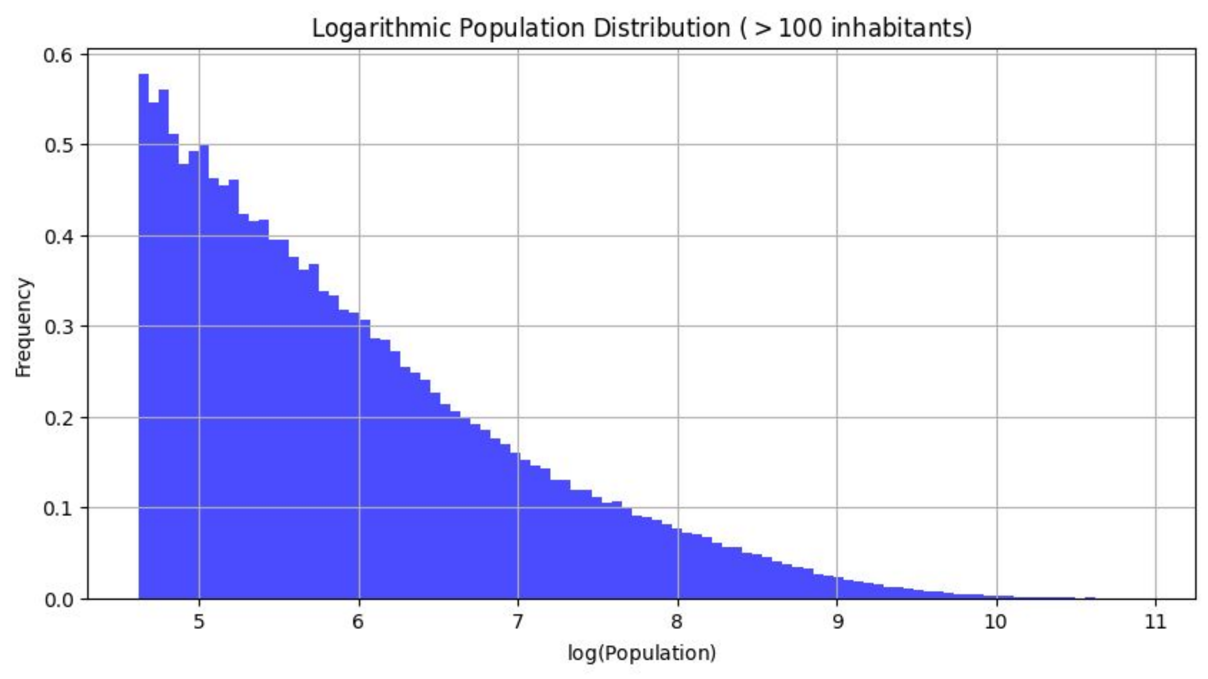
- Population **== 0: ~5.2M** grid cells
- Population range 1 - 100: ~1.3M grid cells
- Population range 100 - 1,000: ~410k grid cells
- Population range 1,000 - 5,000: ~80k grid cells
- Population range 5,000 - 10,000: ~10k grid cells
- Population range **>= 10,000: 3.6k** grid cells



Transformation of data

Highly skewed exponential population distribution!! Even log-transforming it doesn't give a smooth PDF. It is "fixed" only if we remove the segment [0,100]-population grid cells.





Satellite images dataset

- **Data source**: Sentinel-2 satellite, images extracted via the **Google Earth Engine**
- **Bands**: All but Band 1, 9 and 10.
- **Files size** (geotiff files):
  - **>~15GB** compressed into a .zip file
- Paid **Google Colab PRO!!**

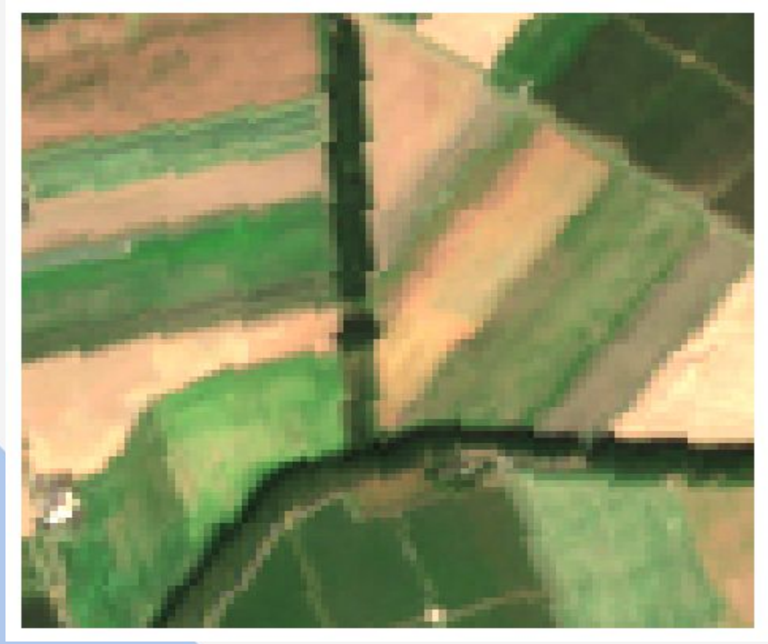
Sentinel-2 Bands	Central Wavelength (µm)	Resolution (m)
Band 1 - Coastal aerosol	0.443	60
Band 2 - Blue	0.490	10
Band 3 - Green	0.560	10
Band 4 - Red	0.665	10
Band 5 - Vegetation Red Edge	0.705	20
Band 6 - Vegetation Red Edge	0.740	20
Band 7 - Vegetation Red Edge	0.783	20
Band 8 - NIR	0.842	10
Band 8A - Vegetation Red Edge	0.865	20
Band 9 - Water vapour	0.945	60
Band 10 - SWIR - Cirrus	1.375	60
Band 11 - SWIR	1.610	20
Band 12 - SWIR	2.190	20

Google Earth Engine

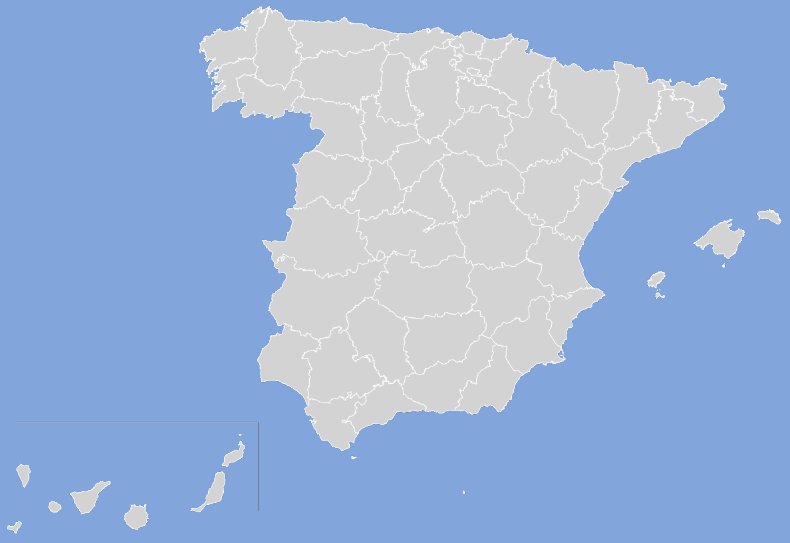
- Tool for analyzing geospatial data.
- Allows for automated download of Sentinel images.
- Took a four-year average and filtered for cloud cover.
- Reprojected satellite images to match population coordinate system (EPSG 3035).

ESA Sentinel-2 Satellite (from the European Space Agency)

Satellite images dataset: Examples



Regional models



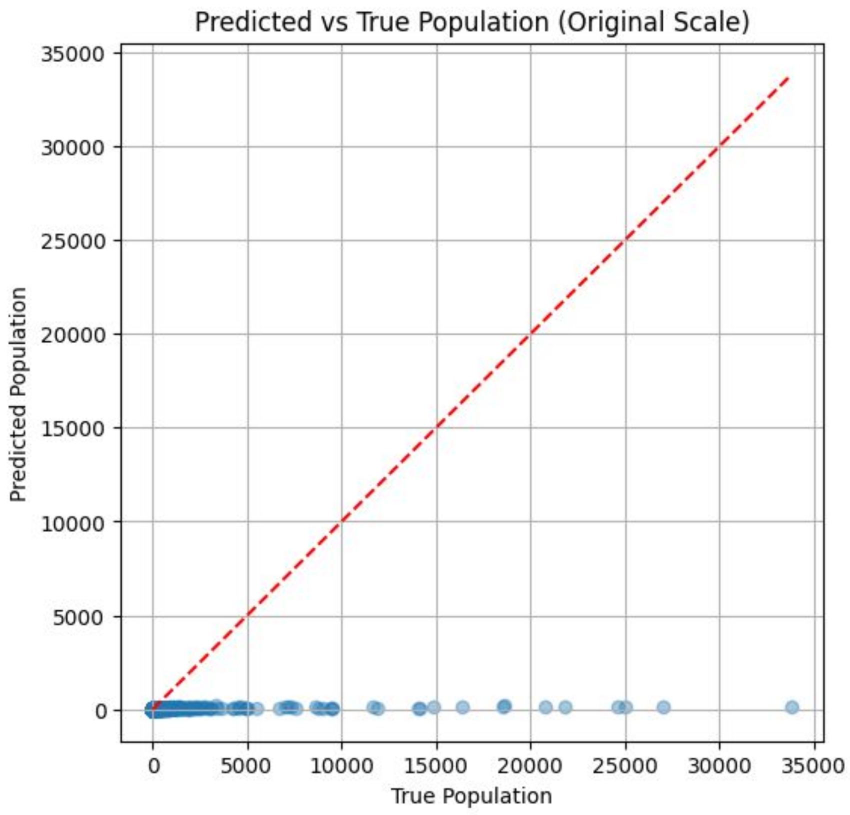
CNN from scratch: Spain

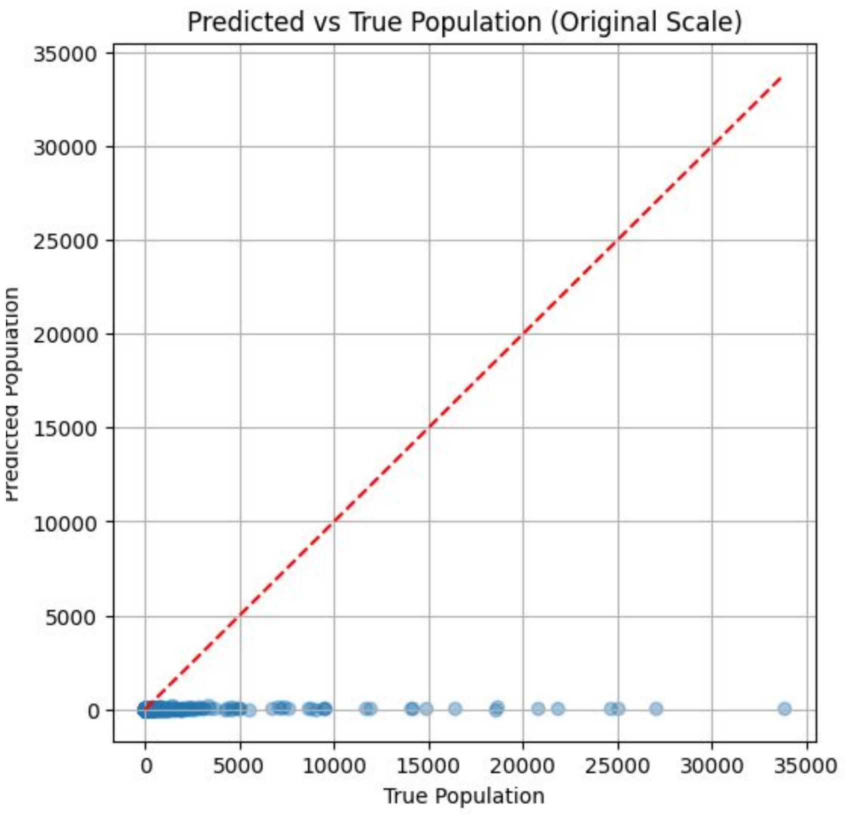
<u>General for all runs:</u>

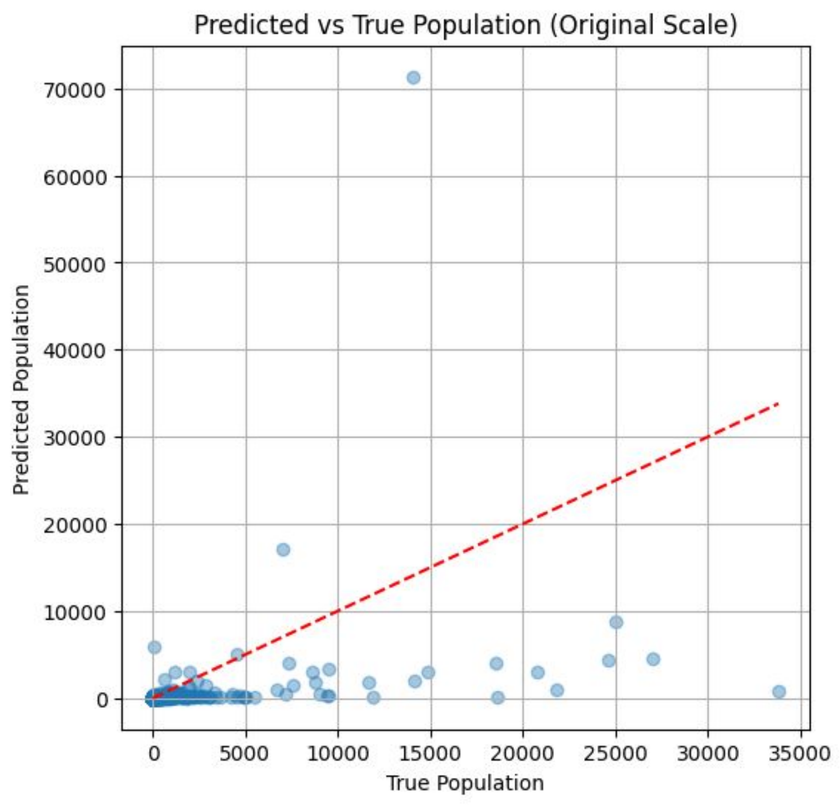
- Dataset showed the global distribution.
- Models were trained for a minimum of 20 Epochs, or if the validation loss had not improved for 5 epochs.
- Bands are normalized.
- No MAE or R^2 is reported because all models were bad.
- Usually the R^2 was a little above 0, which indicates the model is predicting super bad.

```
SimpleCNN Architecture
┌──────────────────────────────────────────────────────────────────────────┐
│  Input: 9-channel 100x100 image, kernel size=3                           │
├──────────────────────────────────────────────────────────────────────────┤
│ Conv Block 1                         │ Conv Block 2                      │
│                                      │                                   │
│ ┌─────────┐      ┌─────┐    ┌──────┐ │ ┌─────────┐     ┌─────┐  ┌──────┐ │
│ │Conv2d   │      │     │    │Max   │ │ │Conv2d   │     │     │  │Max   │ │
│ │9->32    │─────►│ReLU │───►│Pool  │─┼►│32->64   │────►│ReLU │─►│Pool  │─┤
│ │3x3 pad1 │      │     │    │2x2   │ │ │3x3 pad1 │     │     │  │2x2   │ │
│ └─────────┘      └─────┘    └──────┘ │ └─────────┘     └─────┘  └──────┘ │
│ 100x100x32       100x100x32 50x50x32 │ 50x50x64        50x50x64 25x25x64 │
│                                      │                                   │
├──────────────────────────────────────────────────────────────────────────┤
│ Conv Block 3                         │ Fully Connected                   │
│                                      │                                   │
│ ┌─────────┐      ┌─────┐    ┌──────┐ │ ┌─────────┐     ┌─────┐  ┌───────┐│
│ │Conv2d   │      │     │    │Max   │ │ │Flatten  │     │     │  │Linear ││
│ │64->128  │─────►│ReLU │───►│Pool  │─┼►│128x12x12│────►│ReLU │─►│Dropout││
│ │3x3 pad1 │      │     │    │2x2   │ │ │=18432   │     │     │  │(p=0.3)││
│ └─────────┘      └─────┘    └──────┘ │ └─────────┘     └─────┘  ├───────┤│
│ 25x25x128        25x25x128 12x12x128 │                          │Linear ││
│                                      │                          │256->1 ││
│                                      │                          └───────┘│
└──────────────────────────────────────────────────────────────────────────┘
```

Some model runs….







20000 images for training. Google Colab Pro really improved training time.

CNN from scratch: Spain

- Tried different CNN. More simple one, more complex one.
- Tested multiple loss functions and data augmentation(e.g. Flipping image) to increase amount of rare high population areas.
- Transformed data both with Yeo-Johnson and Log.
- None of these helped.

Why is the model always underpredicting?

- For loss functions such as MAE, MSE, the model is trying to **optimize average error**, and it gets pulled toward where most of the points are.

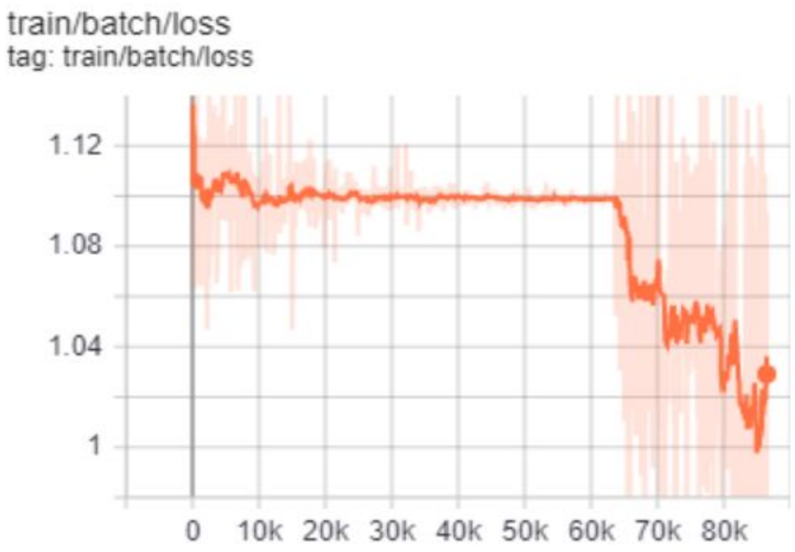
- This could in principle be compensated by:
  - Downsampling high frequency points and upsampling low frequency points(data augmentation
  - Transforming target
  - Adding weights to loss function

- This did not work for us

Potential solution

https://stackoverflow.com/questions/66182725/cnn-regression-model-gives-similar-output-for-all-inputs

It could also be that the problem is very hard to learn. I've had this and actually after 6 hours of identical outputs in each batch (which happens because the 'average' answer is the easiest to minimize loss), the network finally started learning:



Some things I plan on doing to make the learning happen earlier:

1. changes in learning rate
2. using features from earlier in the network

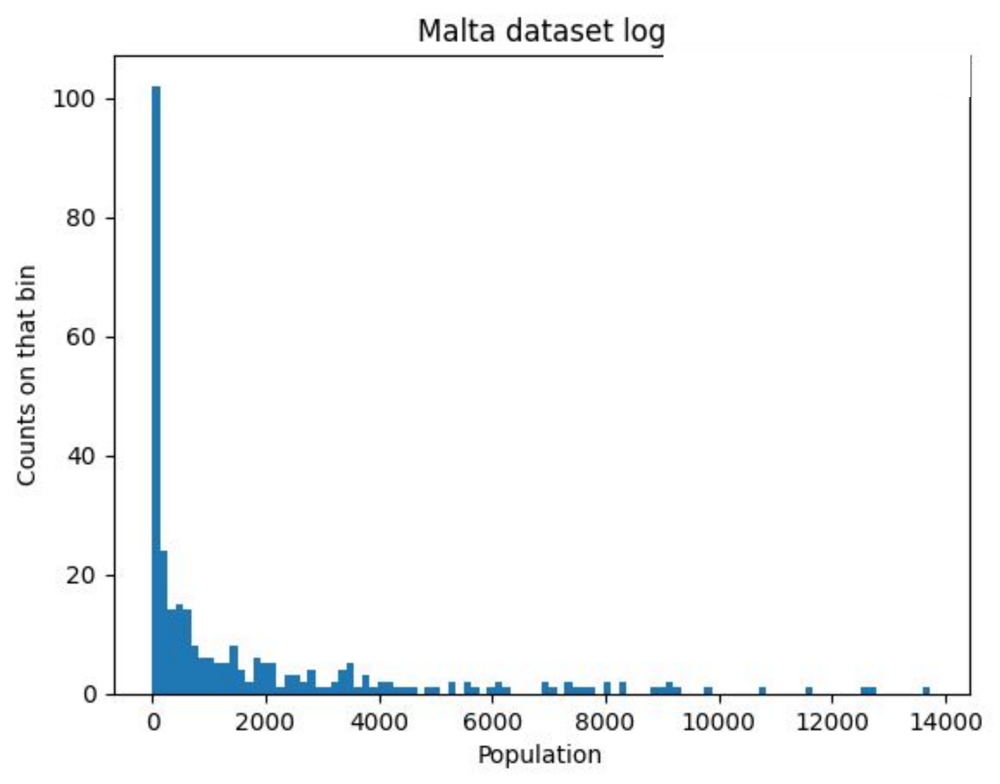
The model is trying to **optimize average error**, and it gets pulled toward where most of the points are.

Takeaways:

-Maybe with more parameter experimentation we could have gotten a model that worked.
-Probably use more data, if your computer have space and you have more time.
-Training time is tedious with images, so CNN from scratch is tough.

Malta Dataset

Initial dataset:
n: 316
Min: 0
Max: 13725
Mean: 1687.53
Median: 540
Non-zero count: 301 CHECK



ResNet-50: Malta

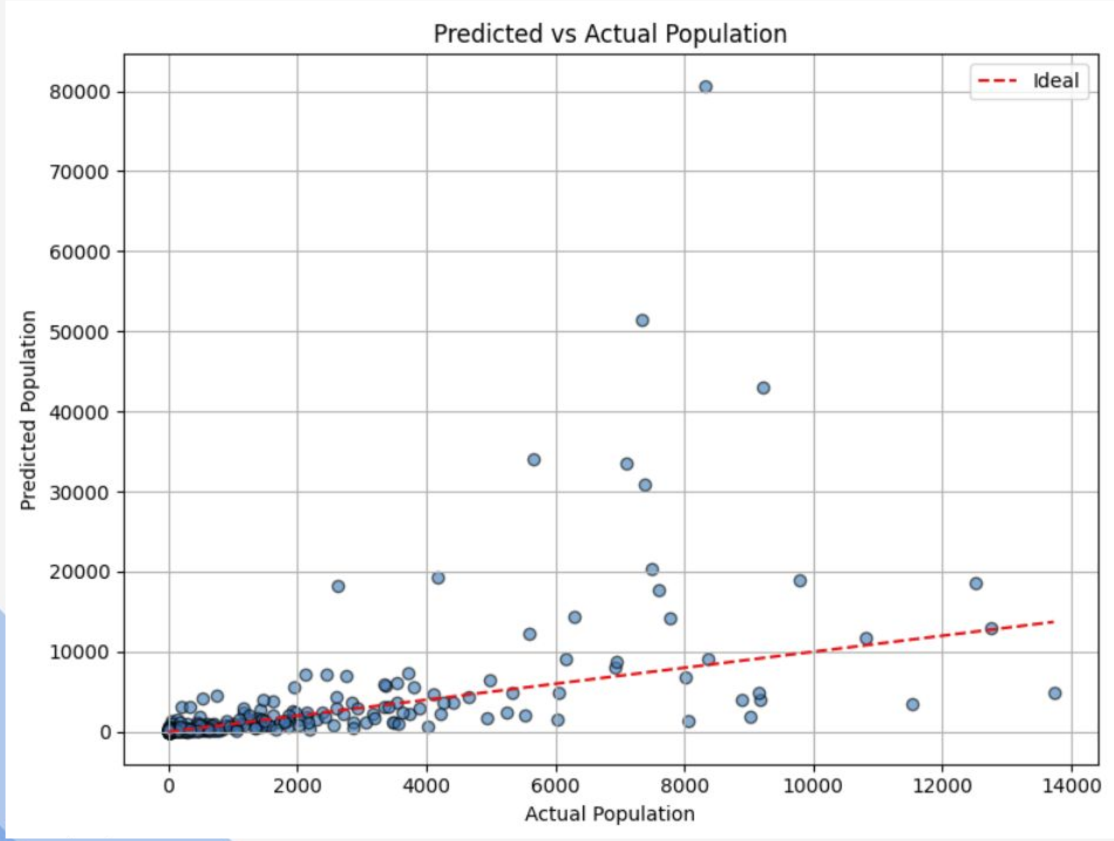
Hyperparameters

- Learning Rate : `9.905e-5`
- Weight Decay: `1.13e-05`
- Batch Size : `8`
- Optimiser : `Adam`

Model Architecture

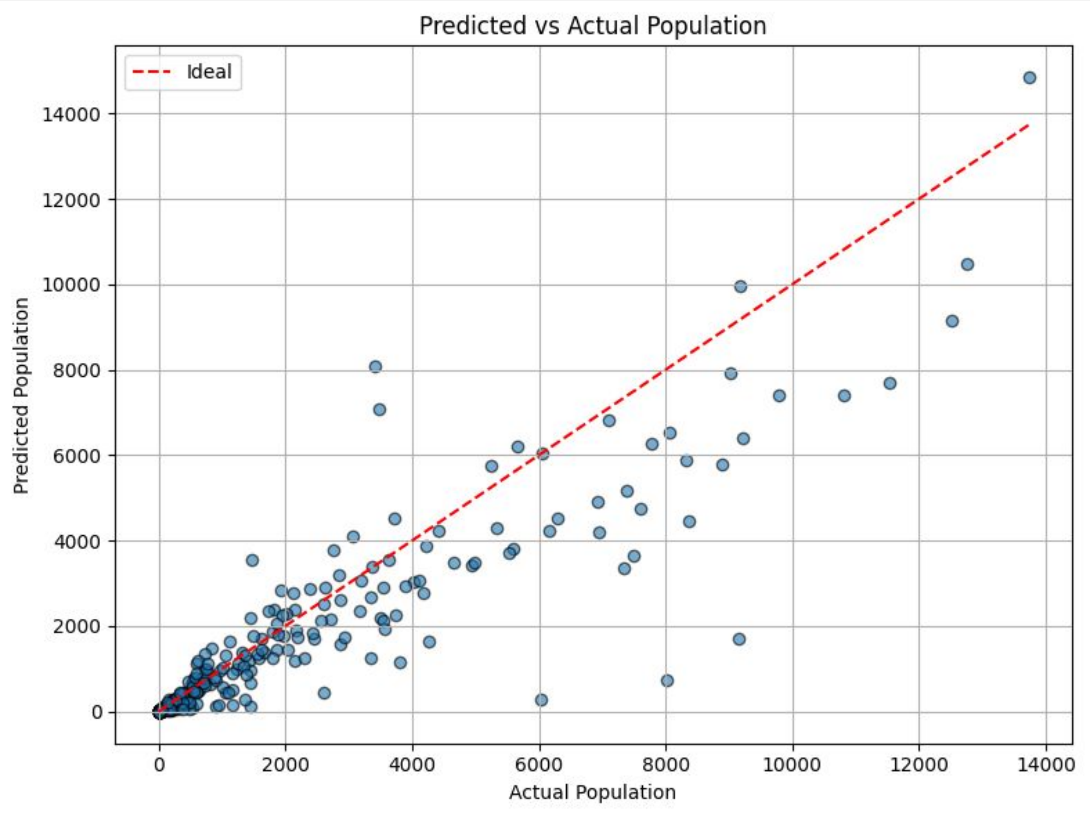
- Kept default ResNet-50 architecture (see appendix)
- Initial train for 20 epochs, then tuned with Optuna

ResNet-50: Freezing & fine tuning



Performance on test set:

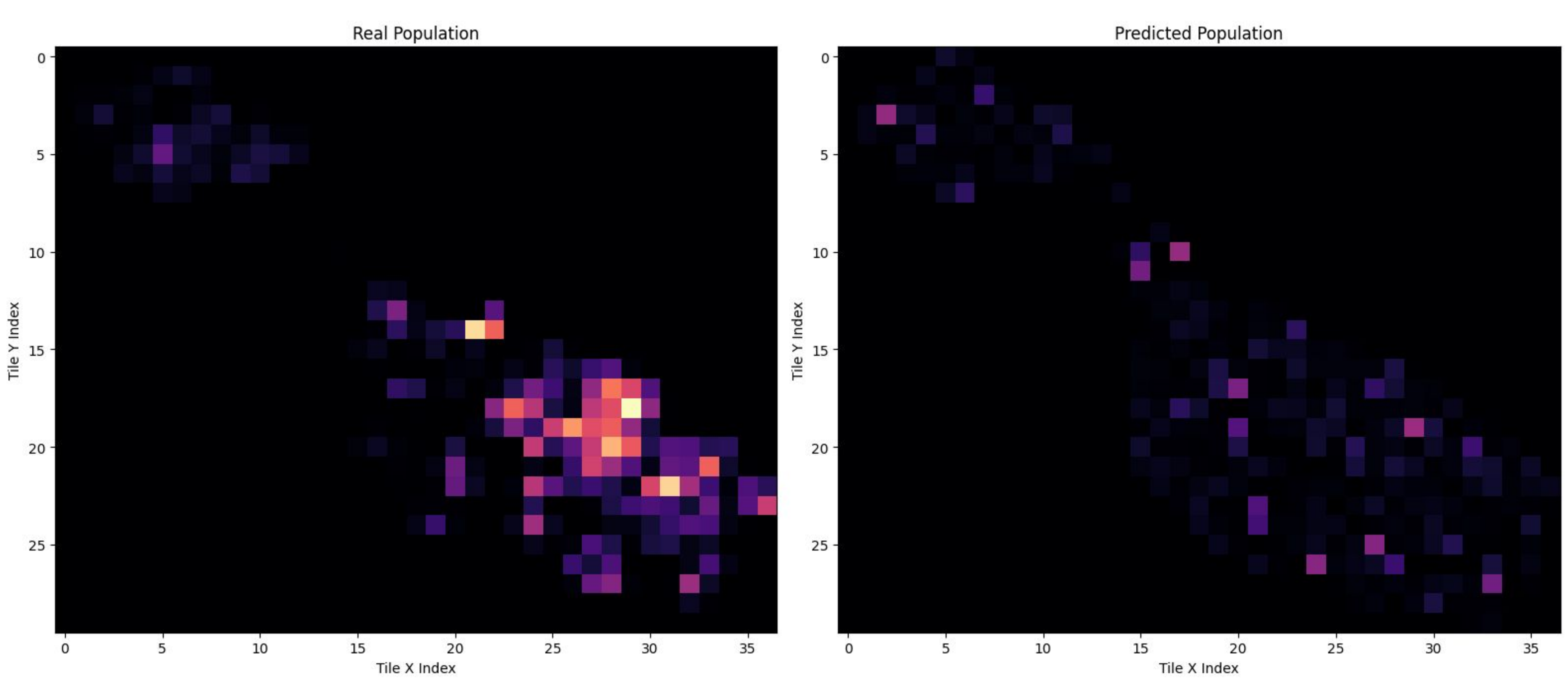
- MAE: 1772.23
- R^2: 0.11
- Training time: 28s



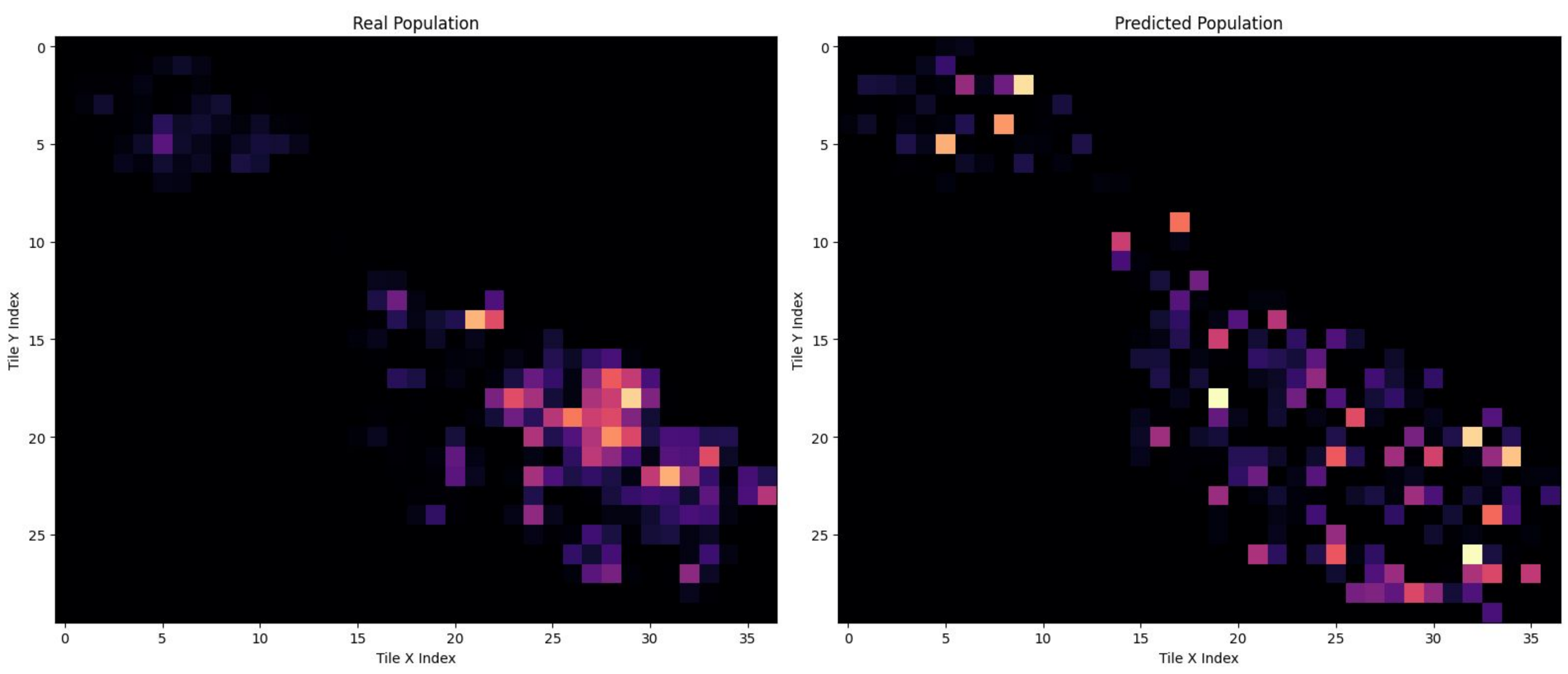
Performance on test set:

- MAE: 546.51
- R^2: 0.69
- Training time: 174s

ResNet-50: Freezing



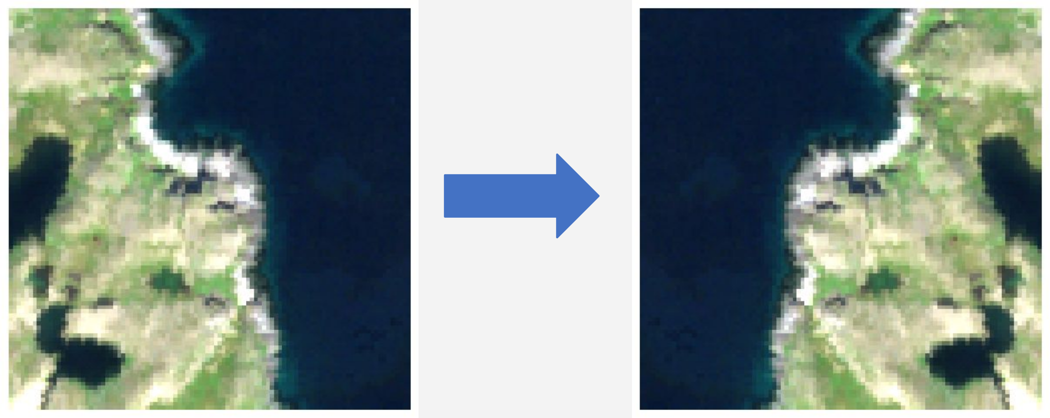
ResNet-50: Fine tuning



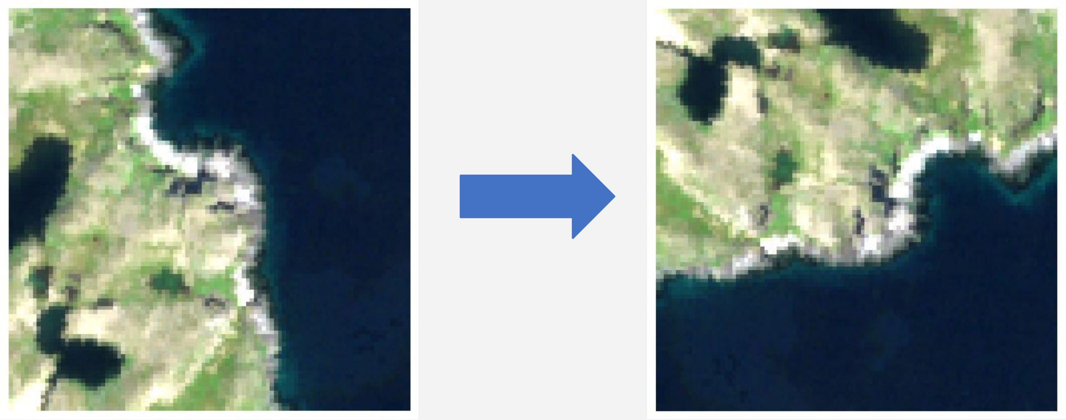
Data Augmentation

Initial dataset:
n: 316

Data augmentation:
Flipping + rotation
N: 903
Library: torchvision.transforms

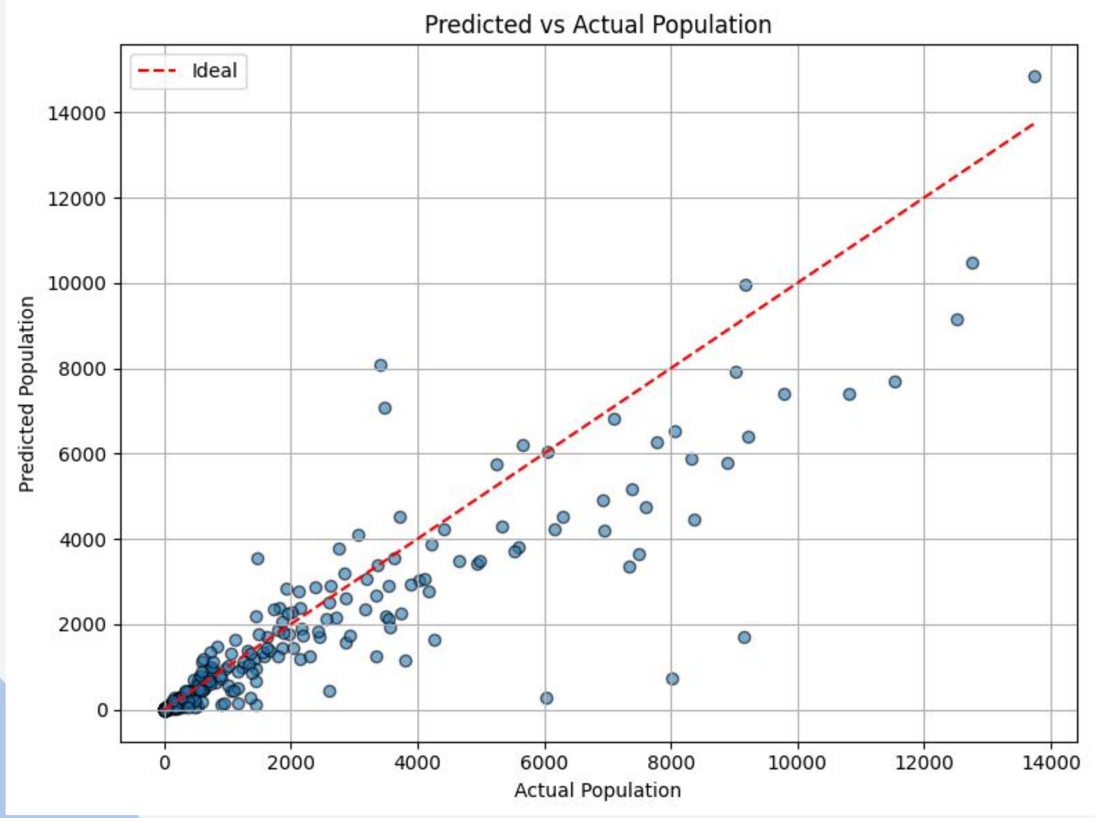


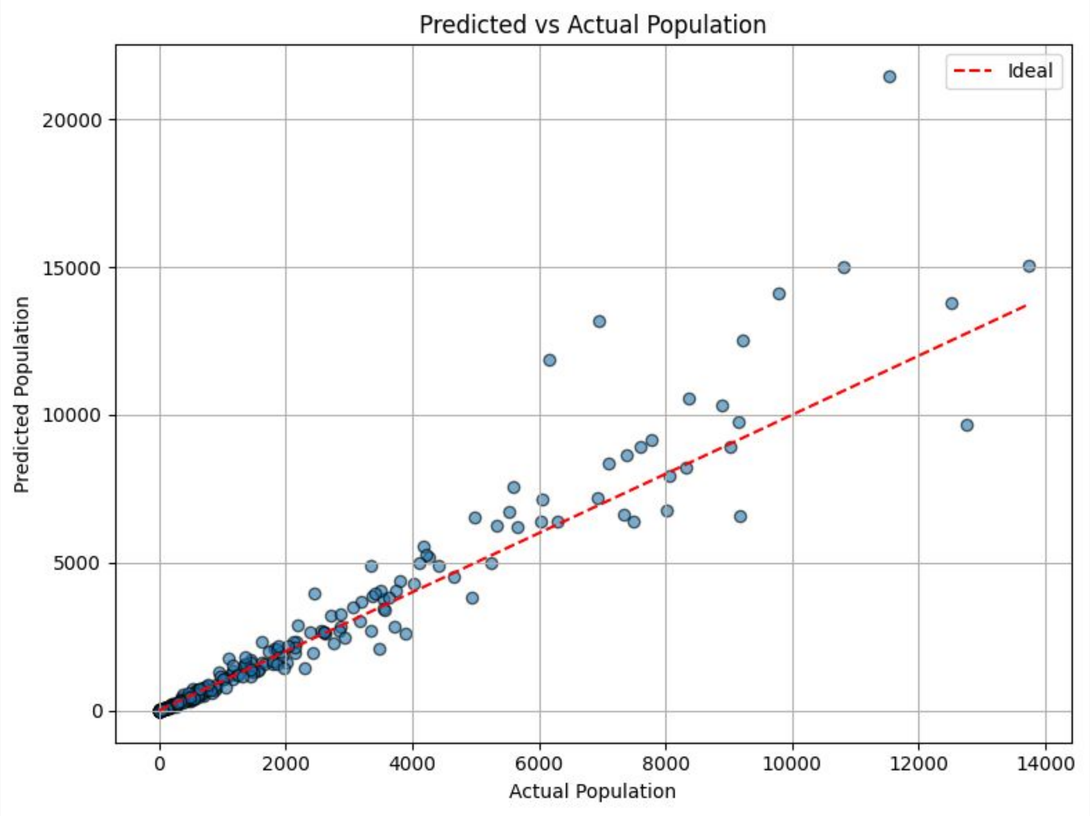
Horizontal flip



90 ° rotation

ResNet-50: Data augmentation





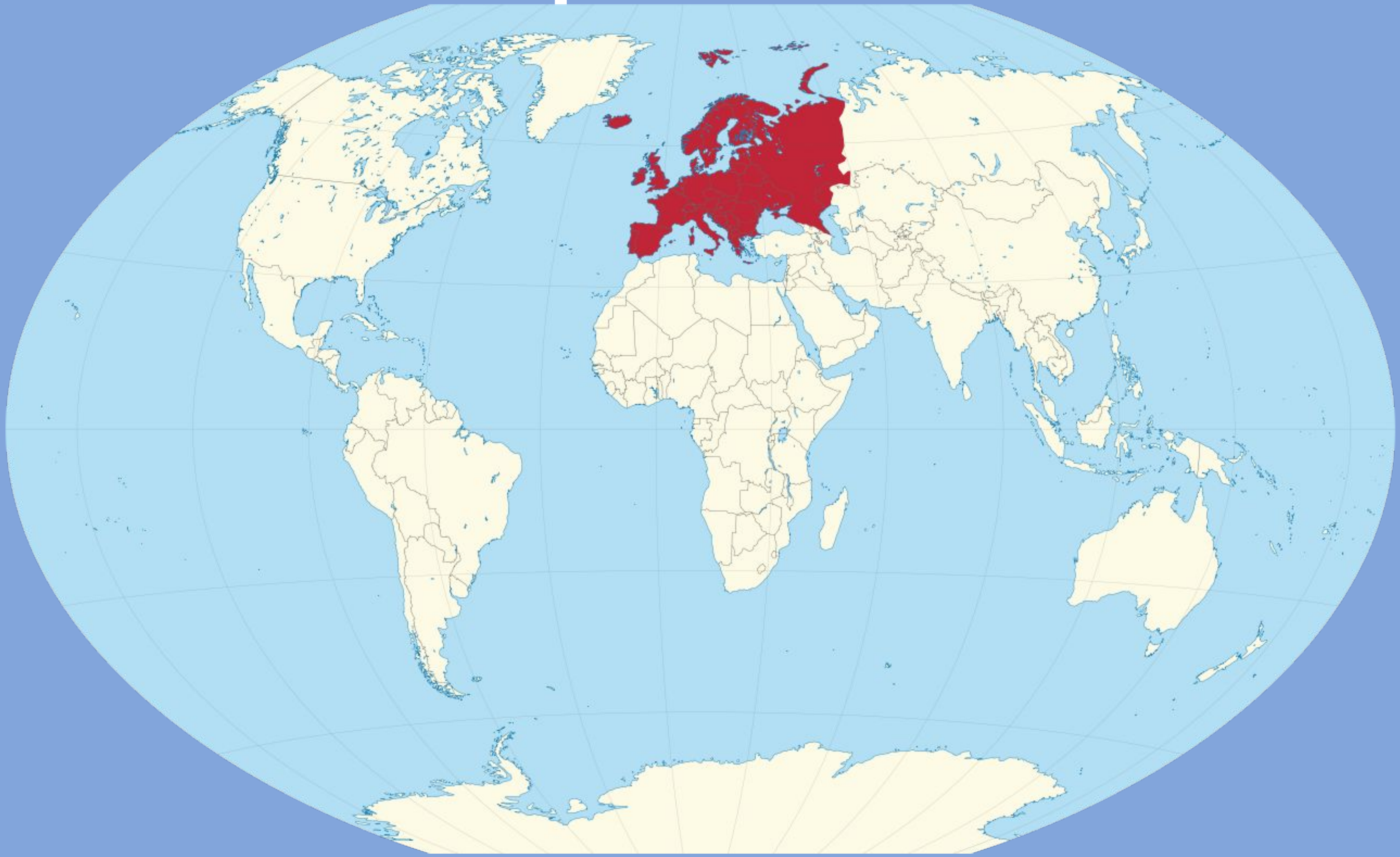
Performance on test set (original dataset):

- MAE: 546.51
- R^2: 0.69

Performance on test set (data augmentation):

- MAE: 338.24
- R^2: 0.85

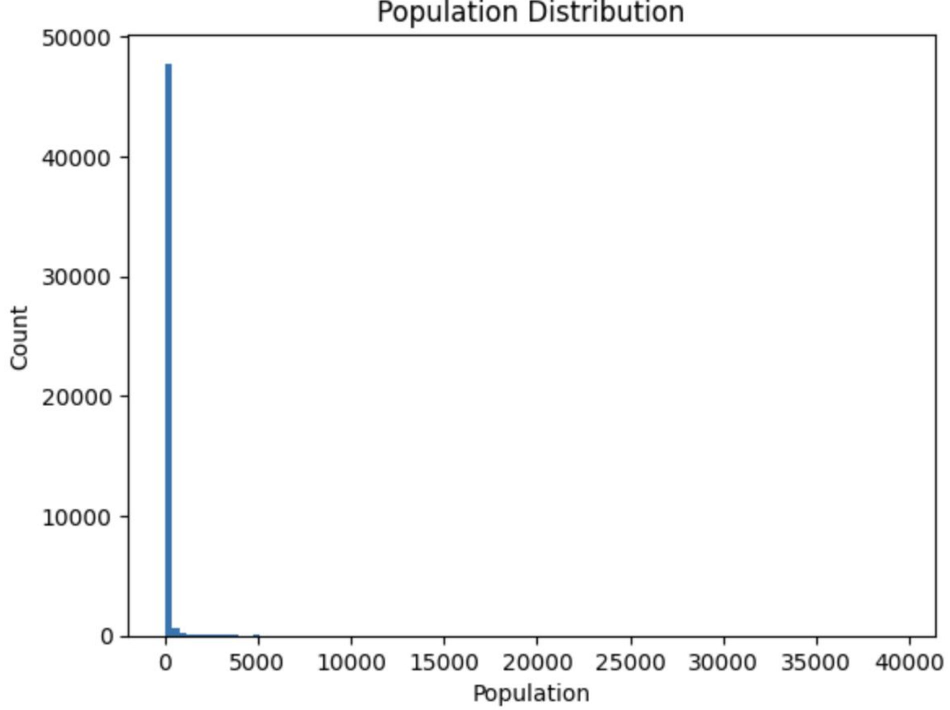
Europe models

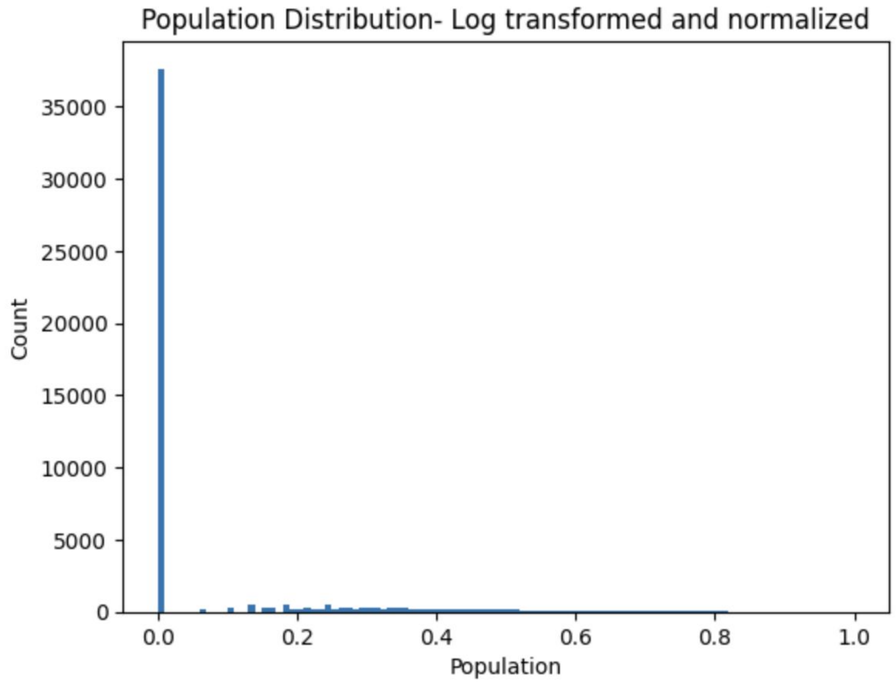


Dataset

Quick Stats:

n: 49,539
Min: 0
Max: 39,434
Mean: 92.14
Median: 0
Non-zero count: 11,899





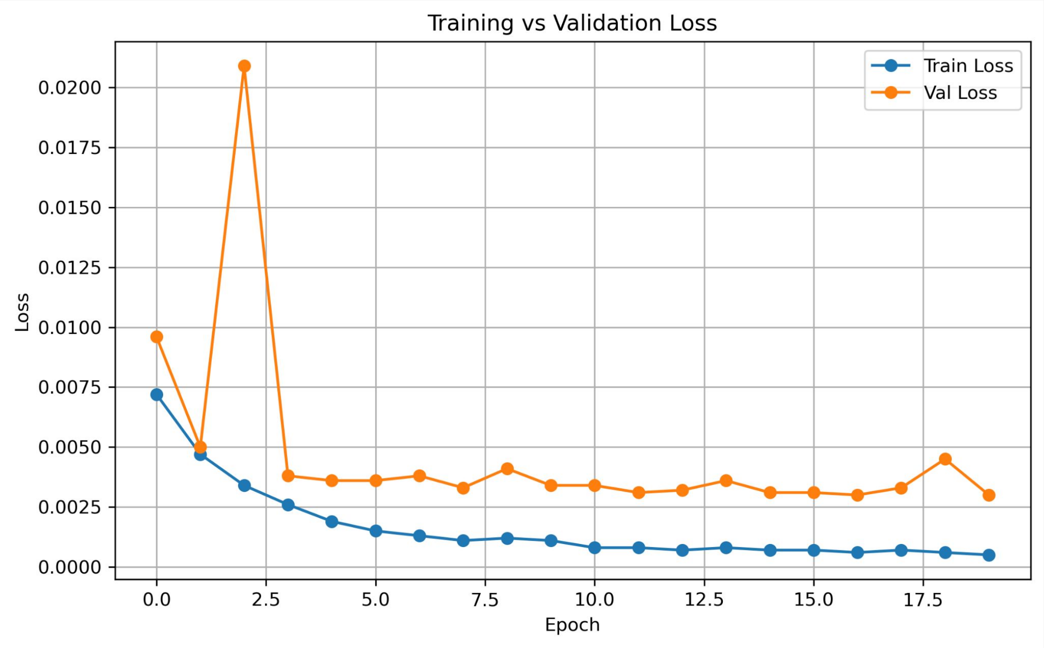
ResNet-50: Europe

Hyperparameters

- Learning Rate : 1e–4
- Weight Decay: 4.922e-5
- Num_workers: 6
- Batch Size : 128
- Optimiser : Adam

Model Architecture

- Kept default ResNet-50 architecture (see appendix)
- Initial train for 20 epochs to fine tune weights
- Tuned hyperparameters with Optuna on a subset of 1000



ResNet-50 Results

Input Channels: 9 bands from Sentinel-2

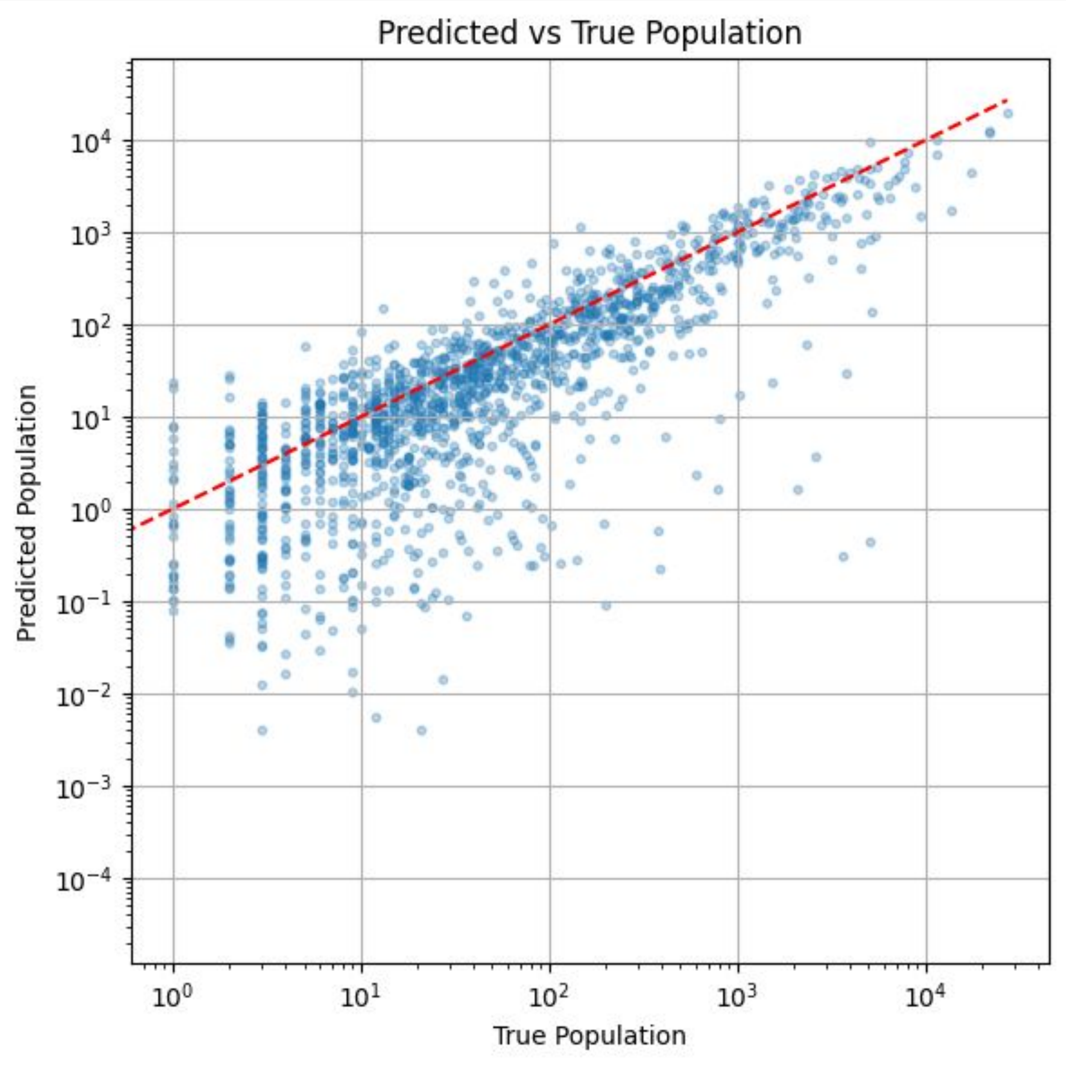
Training Time: ~2 hours initial train, 2 hours tuning on 1k subset (L4 GPU)

Train, test, validation: 0.75, 0.12, 0.12 (stratified)

Performance on test set:
- MAE: 48.52
- R^2: 0.7198

Memory-intensive data was major issue

- Tried to preprocess data and save as tensors (~30GB) to avoid bottleneck for GPU
- Initializing model became prohibitively slow



CNN from scratch: Europe

Hyperparameters used

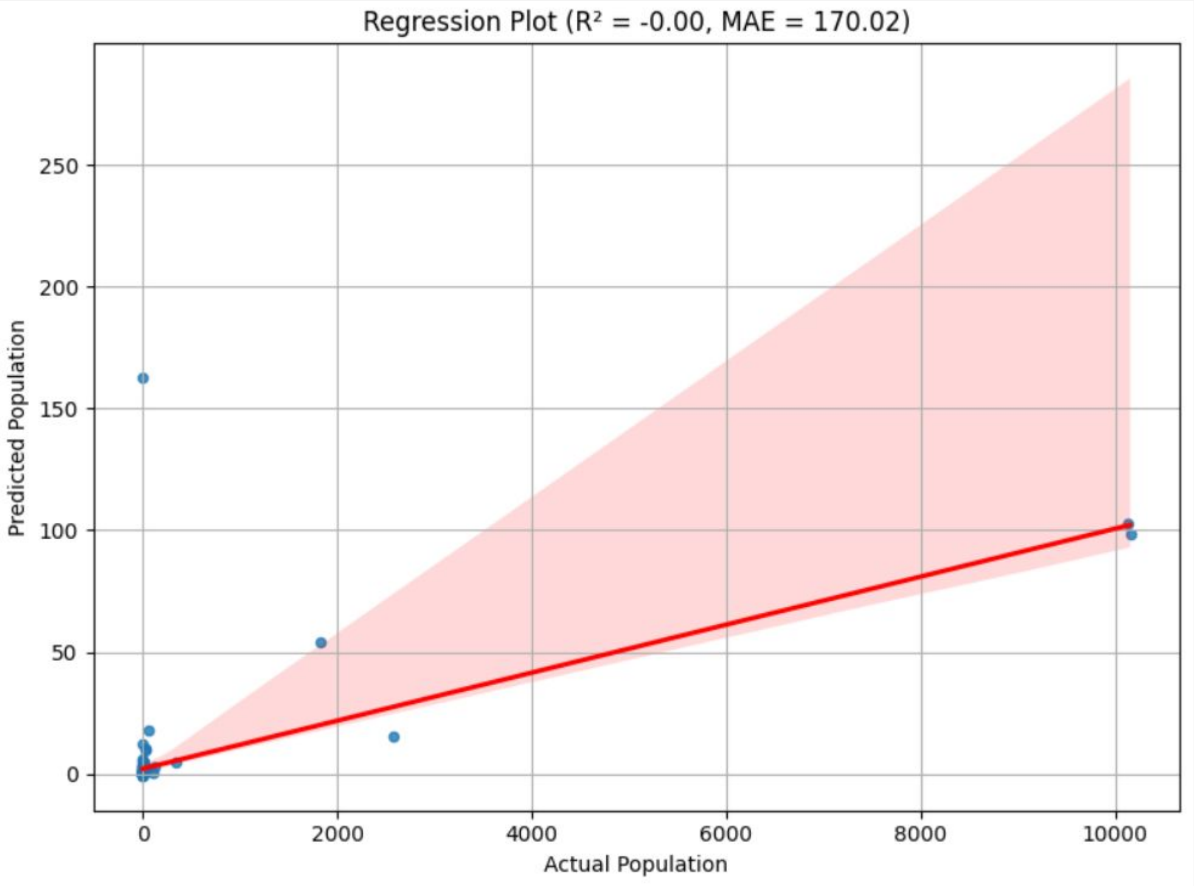
- Learning Rate : `1e-4`
- Batch Size : `16`
- Number of Epochs : `20`
- Optimiser : Adam optimizer (`torch.optim.Adam(model.parameters(), lr=1e-4)`)
- Weighted Huber Function Loss: `Smooth L1`

Input Channels: 9 bands from Sentinel2

Training Time: ~3 hours

Dataset: 5000 sub-dataset

Train, test, validation: 0.7, 0.15, 0.15



Metrics

R^2: 0.0

MAE: 170.02

Let’s try to improve it

Changes in Hyperparameters

- Number of Epochs : `20 → 15`

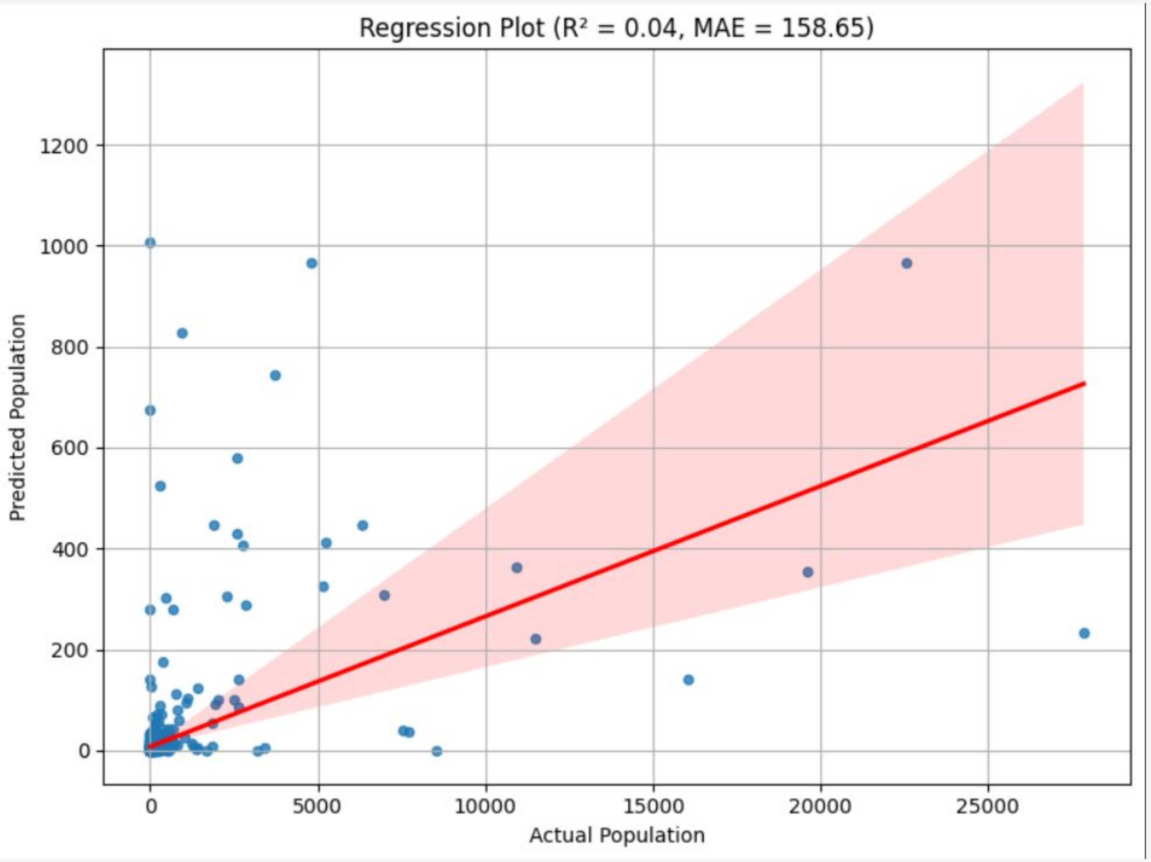
Changes in Model Architecture

- Dropout (Regularization): `Dropout(0.4→0.6)`

and `Dropout(0.3→0.5)`

Changes in Training

- **10k** dataset
- 1 hour training time (GPU Google Colab)



Metrics

R^2: 0.04

MAE: 158.65

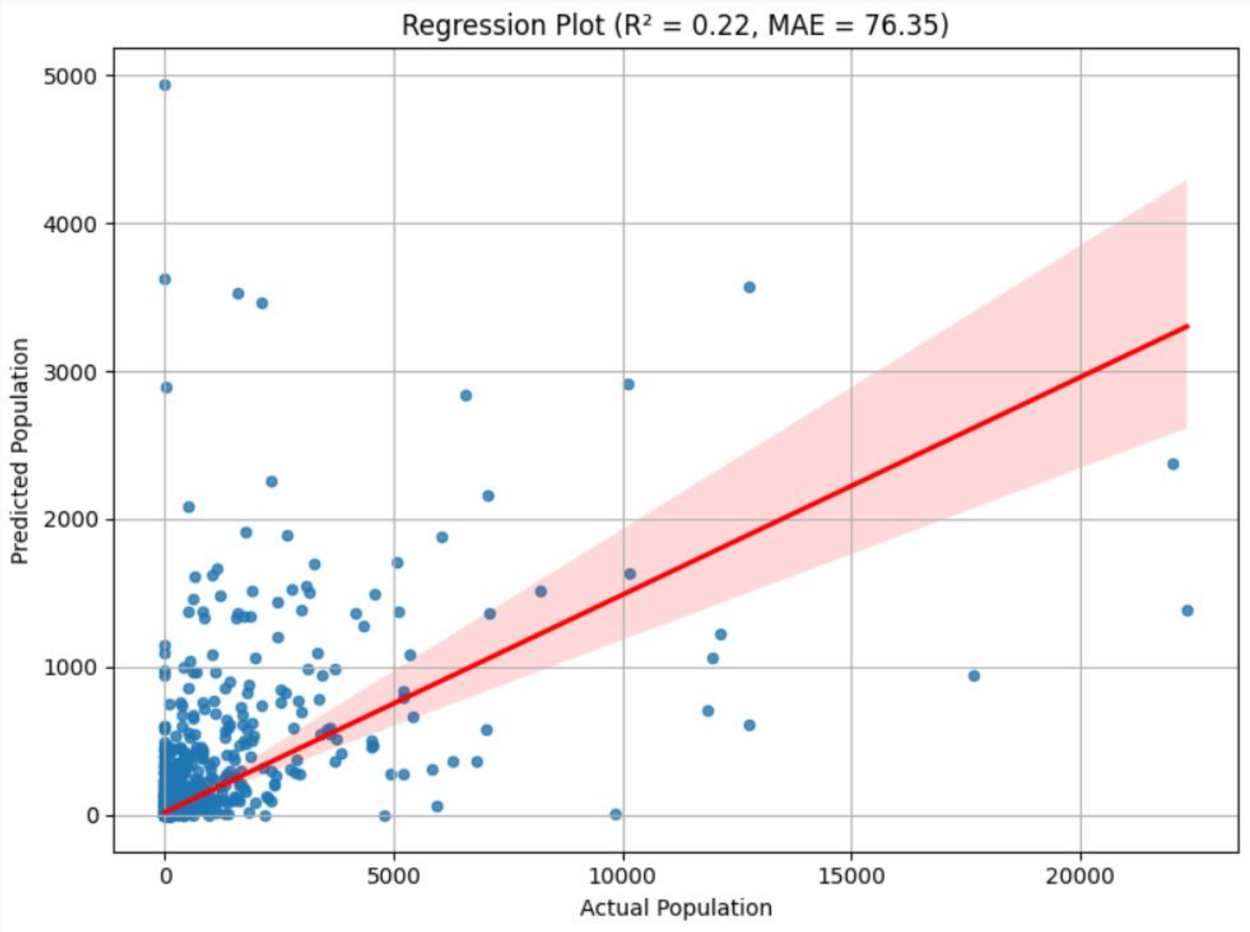
Let's try once more

Changes in Hyperparameters

- Learning Rate : `1e-4→3e-5`
- Batch Size : `16→128`
- Optimiser : Adam optimizer (`torch.optim.Adam(model.parameters(), lr=3e-5, weight_decay=1e-4)`)
- Number of workers: **`8`**

Changes in Training

- **50k** dataset
- 1 hour training time (L4 GPU)



Metrics

R^2: 0.22

MAE: 76.35

CNN from Scratch: Europe

Some thoughts

- How effective is the CNN's architecture for the data we have?
  Is it too shallow or too complicated? Is the loss function too sensitive or not sensitive enough?

https://ineducationonline.org/2021/04/02/the-importance-of-critical-thinking/

Conclusions & Further Research

Conclusions & Future Work

- ResNet50 >>> CNN built from scratch → Use foundation models!
- More data/data augmentation → Not consistent improvement.
- Efficient data loading and GPU utilization is key for large CNNs with high-resolution images.
- Ideas for future work:
  - Add extra bands from other satellites: elevation, slope, night light…
  - Try different transformations on population data.
  - Better optimize the hyperparameters/CNN architecture.

Thank you all!

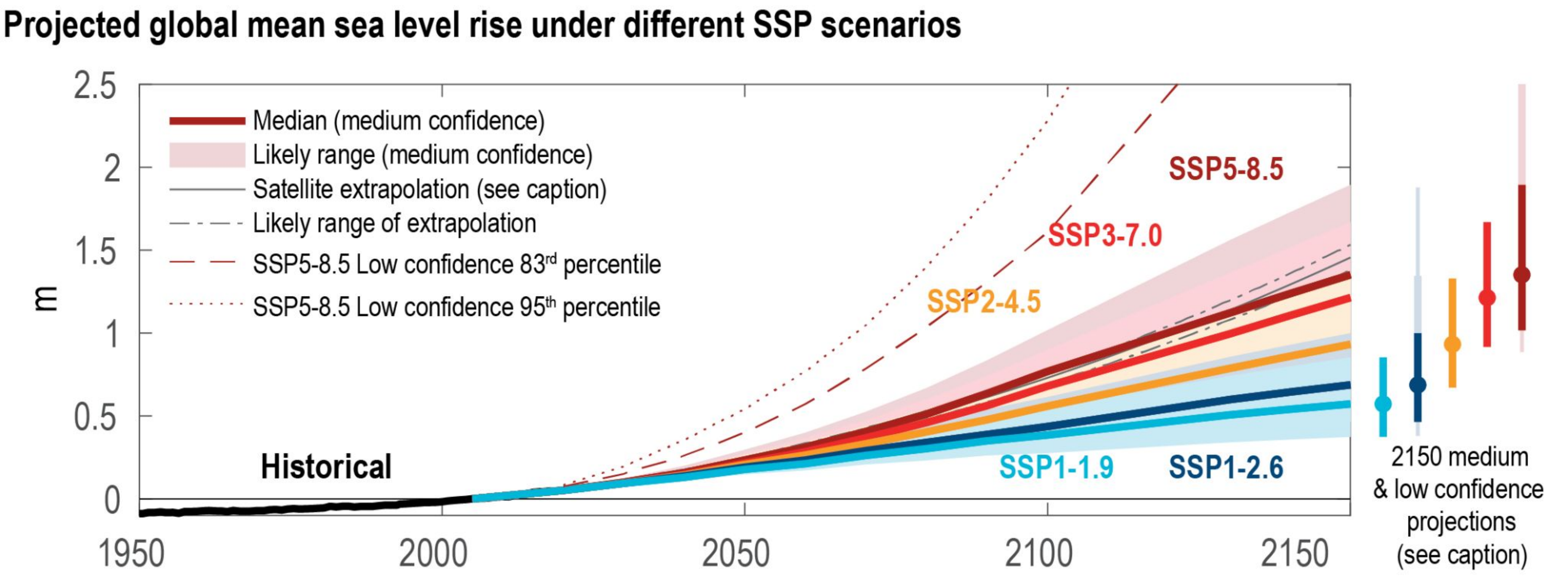
Special thanks to Aslak for the guidance during the project

Questions?

Appendix

Motivation in more detail

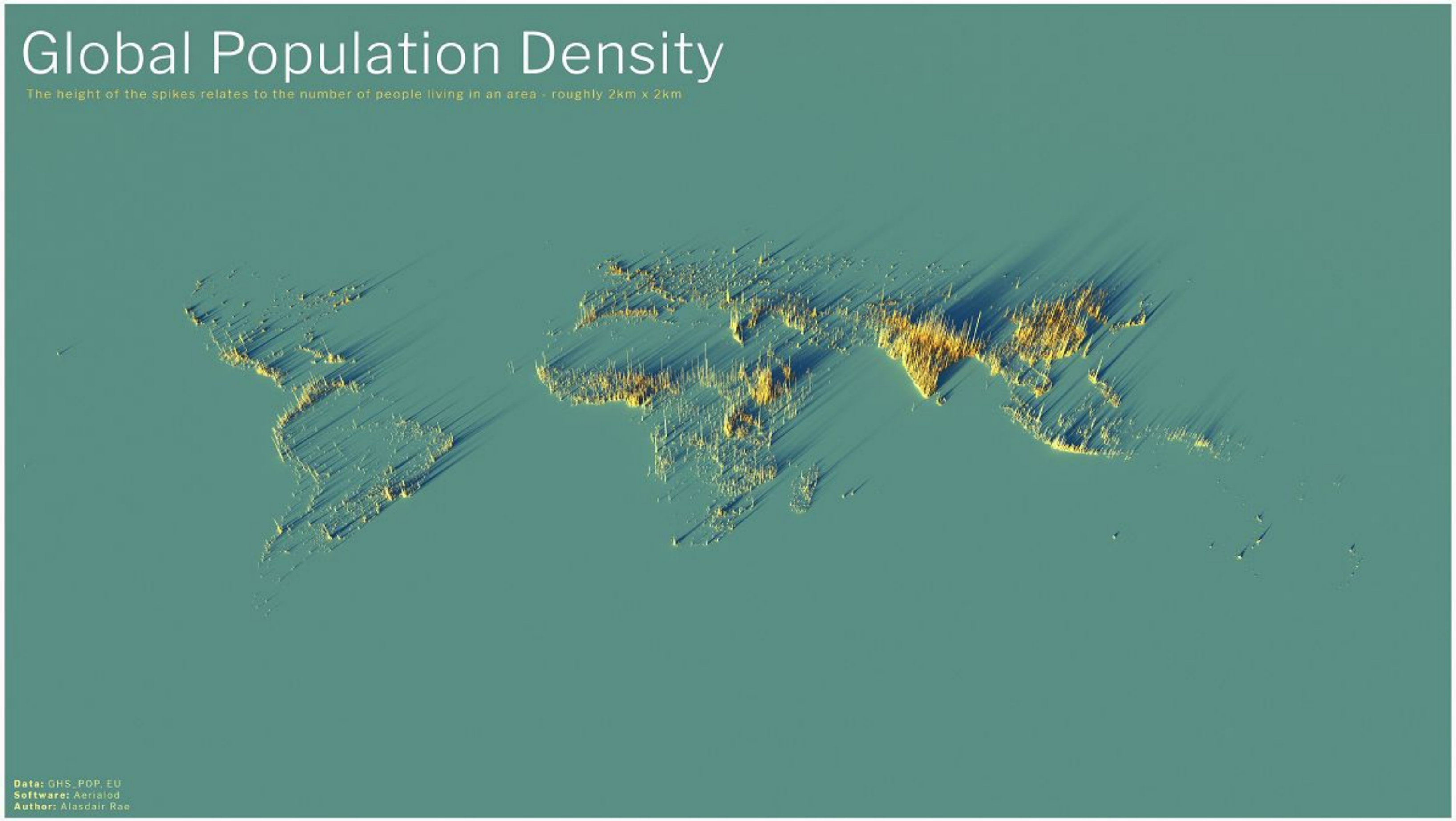
Sea Level Rise under Climate Change



IPCC AR6, WG1: Chapter 9. Figure 9.27

Population living in coastal regions

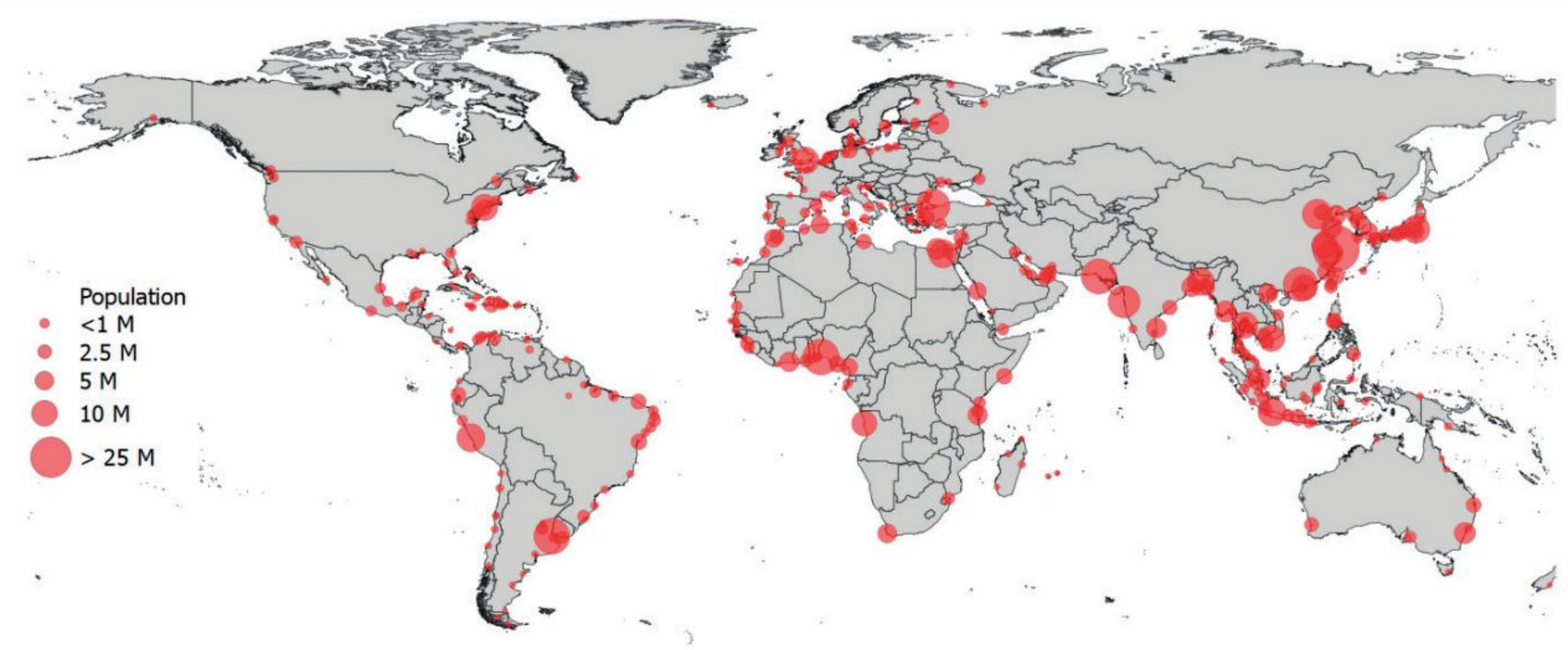
"Much of the world's population, economic activities and critical infrastructure are concentrated near the sea, with nearly **11% of the global population** (**896 million people**) already living on low-lying coasts." - IPCC AR6 WG2, Cross-Chapter Paper 2



Data: GHS_POP, EU.
Author: Alasdair Rae

Population living in coastal regions

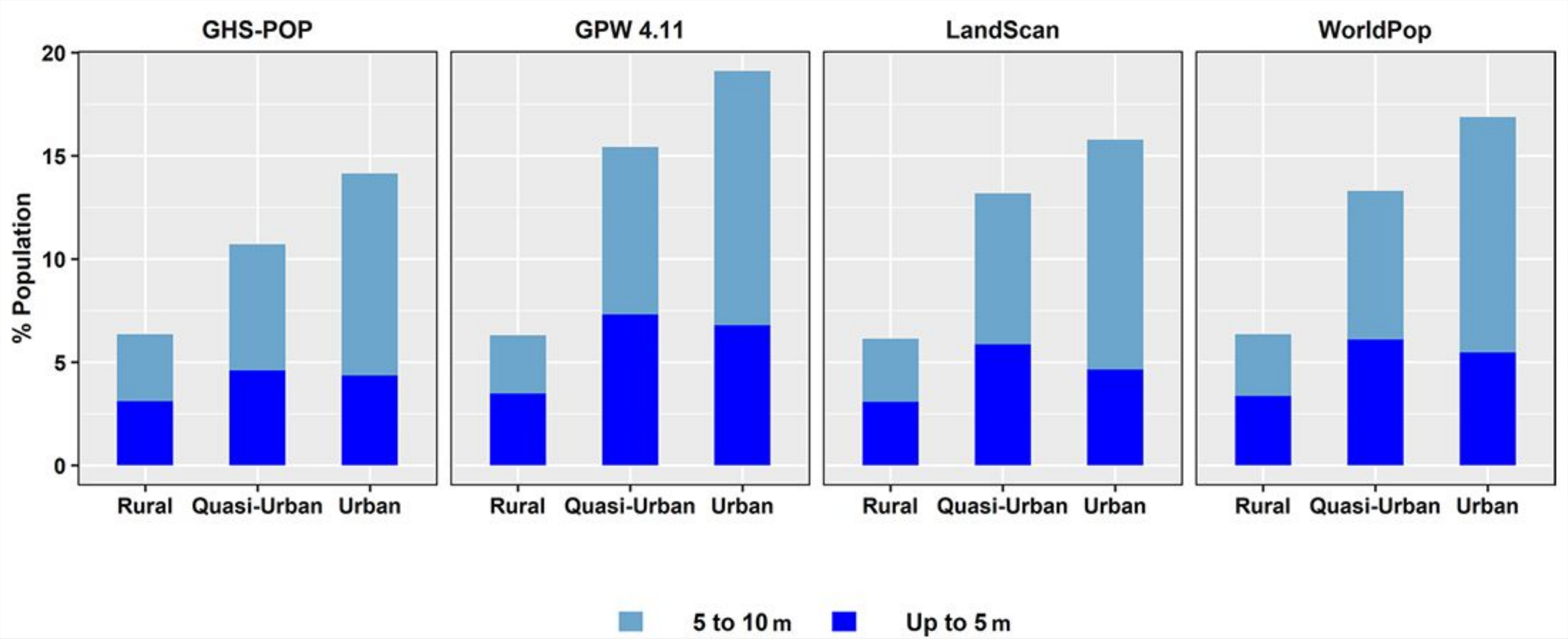
"Much of the world's population, economic activities and critical infrastructure are concentrated near the sea, with nearly **11% of the global population** (**896 million people**) already living on low-lying coasts." - IPCC AR6 WG2, Cross-Chapter Paper 2.



King, 2022: Figure 1

Uncertainties in the data

"Between 750 million and nearly 1.1 billion persons globally live in the 10m LECZ, with the variation depending on the **elevation, population data sources** and **differing population classifications.**" - MacManus et al, 2021.



MacManus et al, 2021: Figure 13

Objective -> Long description

We aim to reduce uncertainty in coastal population density estimates by predicting population using satellite imagery. To achieve this, we compare several machine learning models based on convolutional neural networks (CNNs). Specifically, we train a regional model on images from Malta, another on images from the Netherlands, and evaluate each on both countries. Additionally, we train a “global” model using data from both regions to assess overall performance.

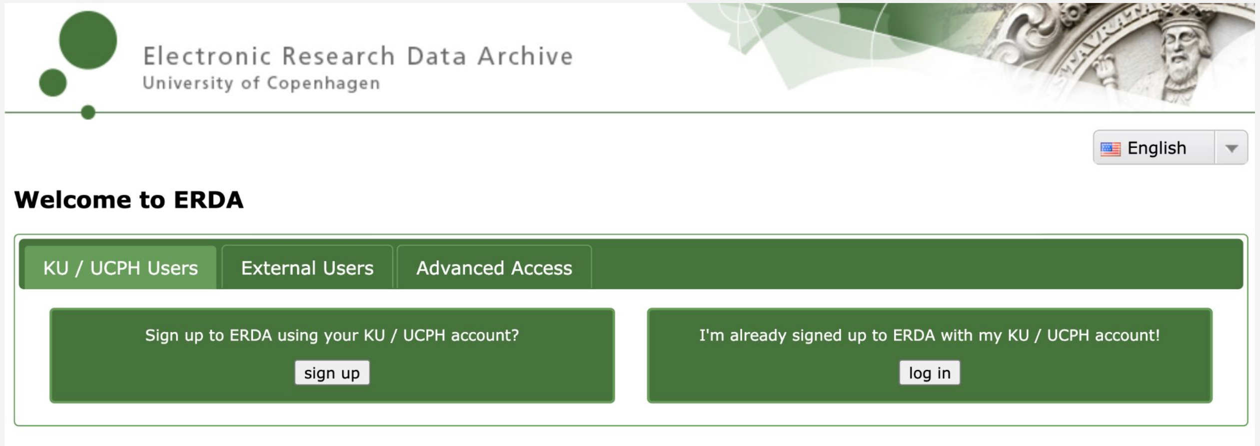
All models are trained twice: once using a ResNet50 foundation model, and once using a combination of AutoEncoders and a PyTorch CNN. A comparison of these two approaches is presented.

Our hypothesis is that regional models will be more accurate within their training regions but less transferable, while the global model may be less precise locally but more adaptable and scalable across different regions.

Computing resources

ERDA

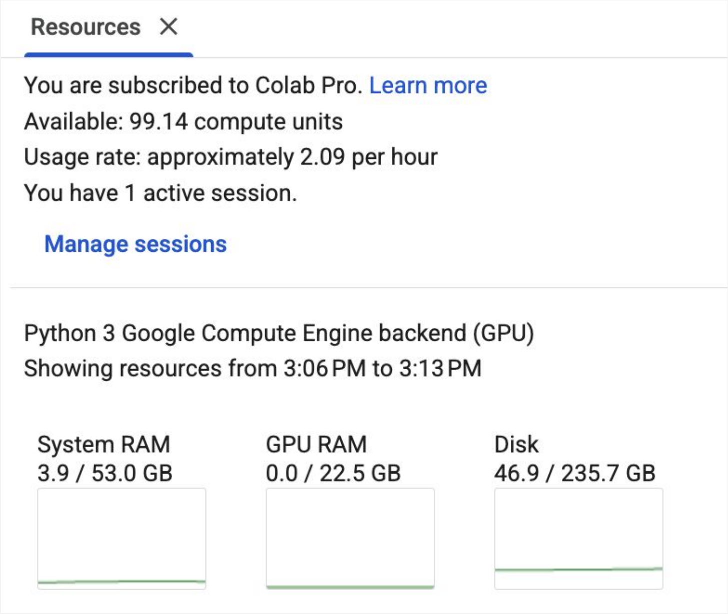
- GPU notebook in DAG.
- Instances have access to 8 compute threads/cores and 16GB of memory.
- Trying to speed training up by increasing num_workers ate up all the RAM.

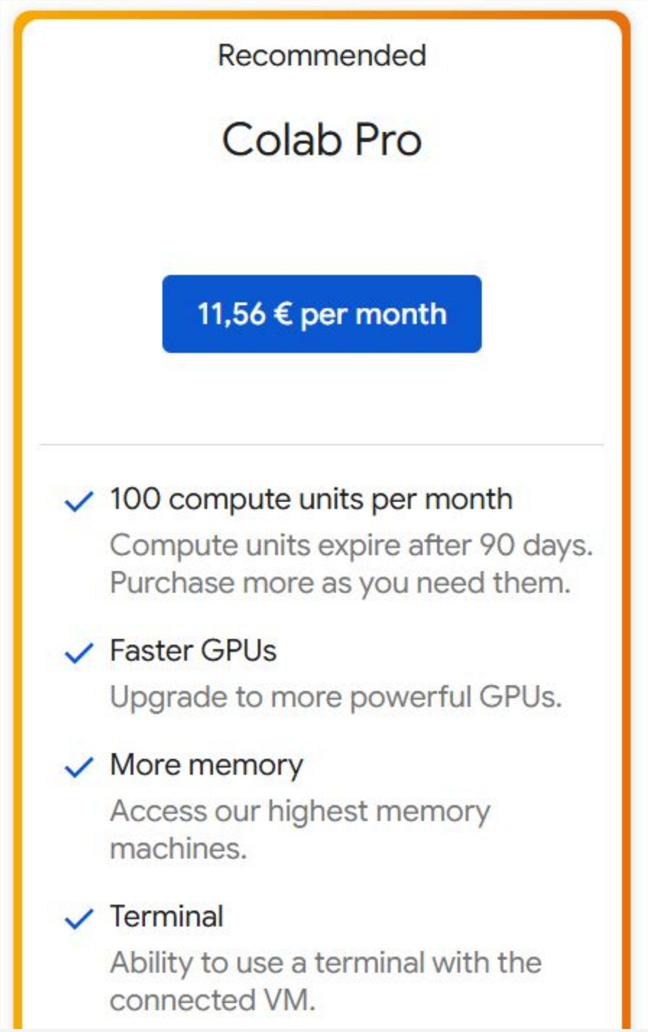



```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 560.35.05              Driver Version: 560.35.05      CUDA Version: 12.6     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
```

Google Colab

- We used the free GPU services from Google Colab very fast.
- Switching to the paid service sped training up, but slow data loading was still an issue.
- Unable to switch between GPU and CPU in the same session → meant wasting GPU runtime.





Loss Functions

LOSS FUNCTION	KEY ADVANTAGE
MAE	Robust to outliers
MSE	Emphasizes large errors
Weighted MSE	Prioritizes certain samples or classes
Huber / Smooth L1	Balances robustness & sensitivity to outliers

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

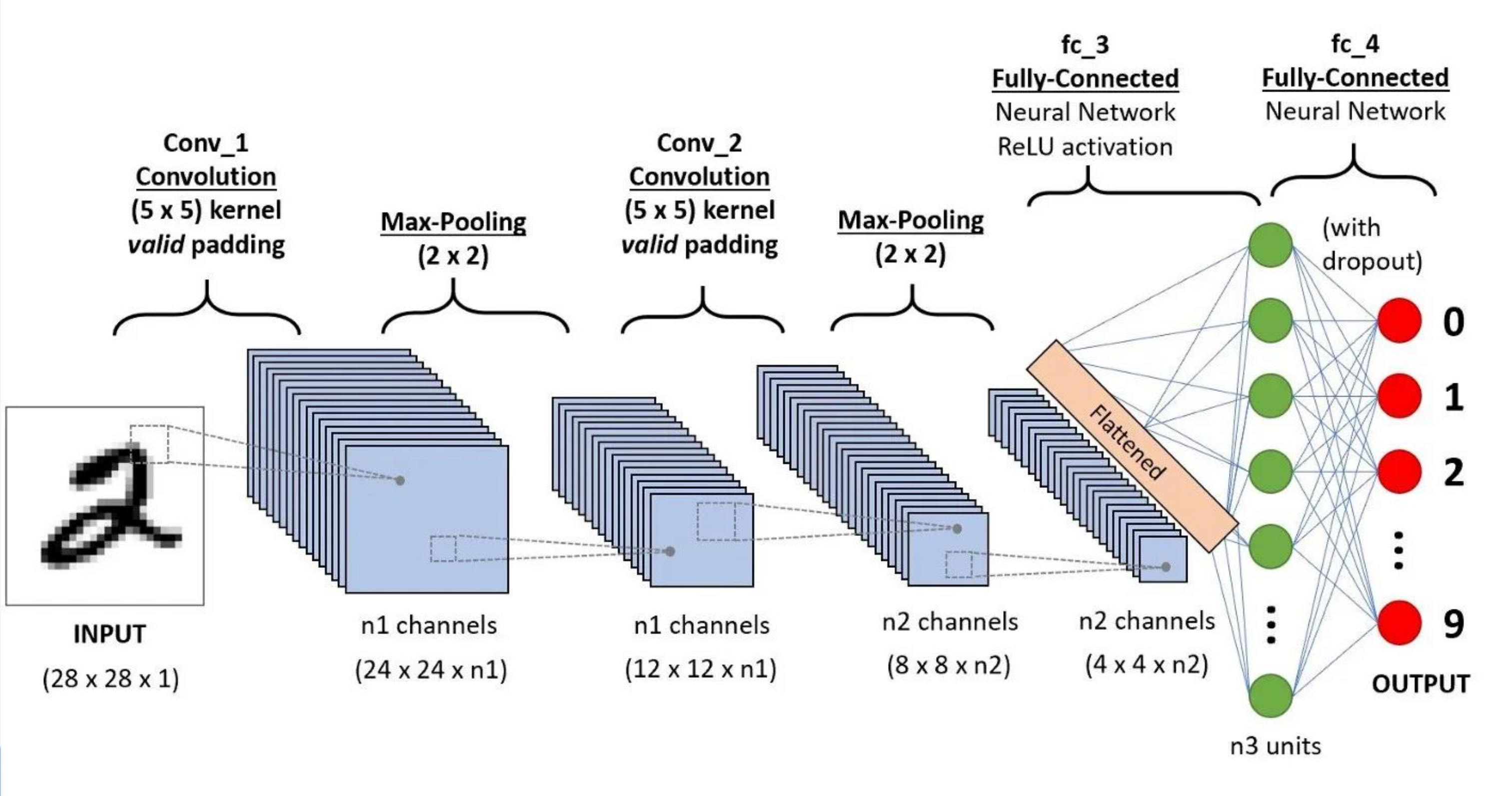
$$\mathrm{WMSE} = \frac{1}{n}\sum_{i=1}^{n}w_i(y_i - \hat{y}_i)^2$$

$$\mathrm{Huber}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2, & |y-\hat{y}| \leq 1 \\ |y-\hat{y}| - \frac{1}{2}, & \text{otherwise} \end{cases}$$

Background Info: CNNs & ResNet

CNN architecture

- Standard for training on **images**.
- Composed of three main types of layers: **convolutional**, **pooling**, and **fully connected**.
- Convolution refers to **filter**/kernel which moves across input and detects features.



Example CNN architecture from towardsdatascience.com

Foundation model: ResNet-50

- CNN model from the Residual Network family (He et al. 2015).
- Solves the vanishing gradient problem through **skip connections**.
- Trained on > **1e6 images** from ImageNet dataset.
- ResNet-50 is the intermediate depth version, and also uses a **bottleneck design** to reduce training time.

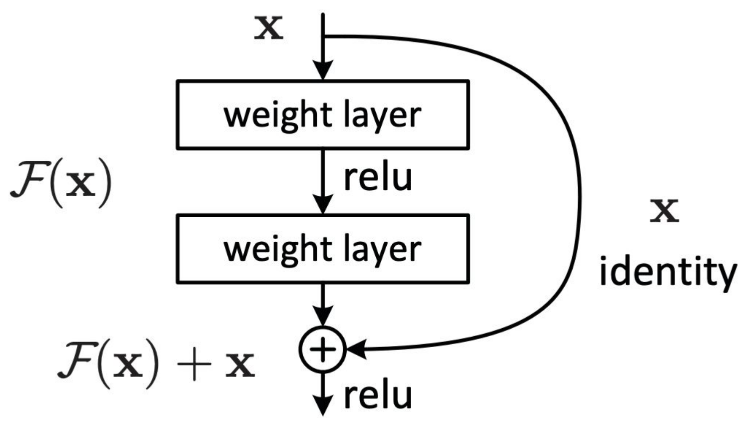
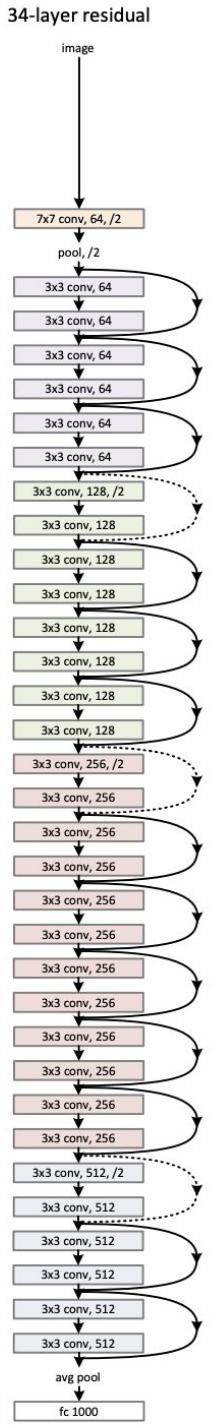


Figure 2. Residual learning: a building block.

Left: Illustration of skip connections.
Right: ResNet-34 architecture.

He et al., 2015



Models - Additional Information

ResNet-50 Europe Architecture

Used default architecture, with fine-tuned weights:

Convolutional Layers:
Multiple Conv2D layers, with kernel_size=3 or 1, and padding=1, arranged in residual blocks.

Increasing Filters:
64 → 256 → 512 → 1024 → 2048

Activation Function:
Rectified Linear Unit (ReLU)

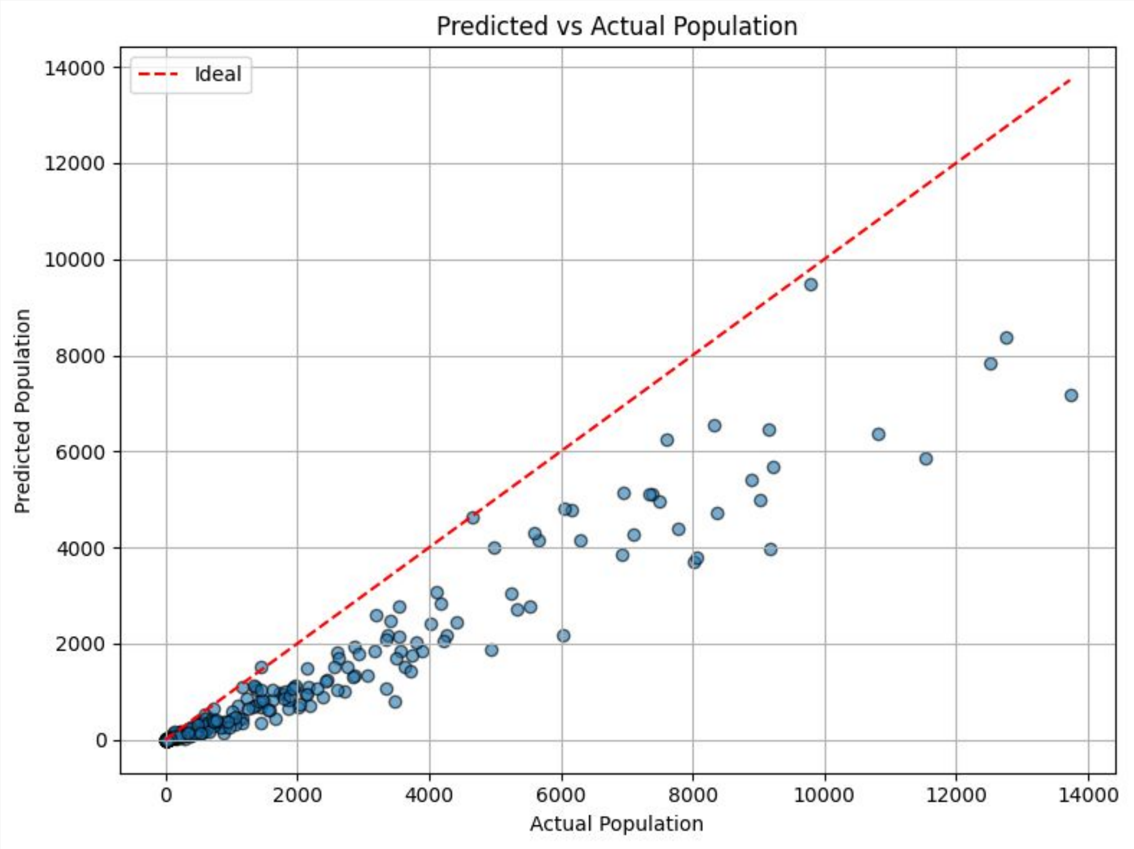
Batch Normalization:
BatchNorm2D after each convolution within residual blocks

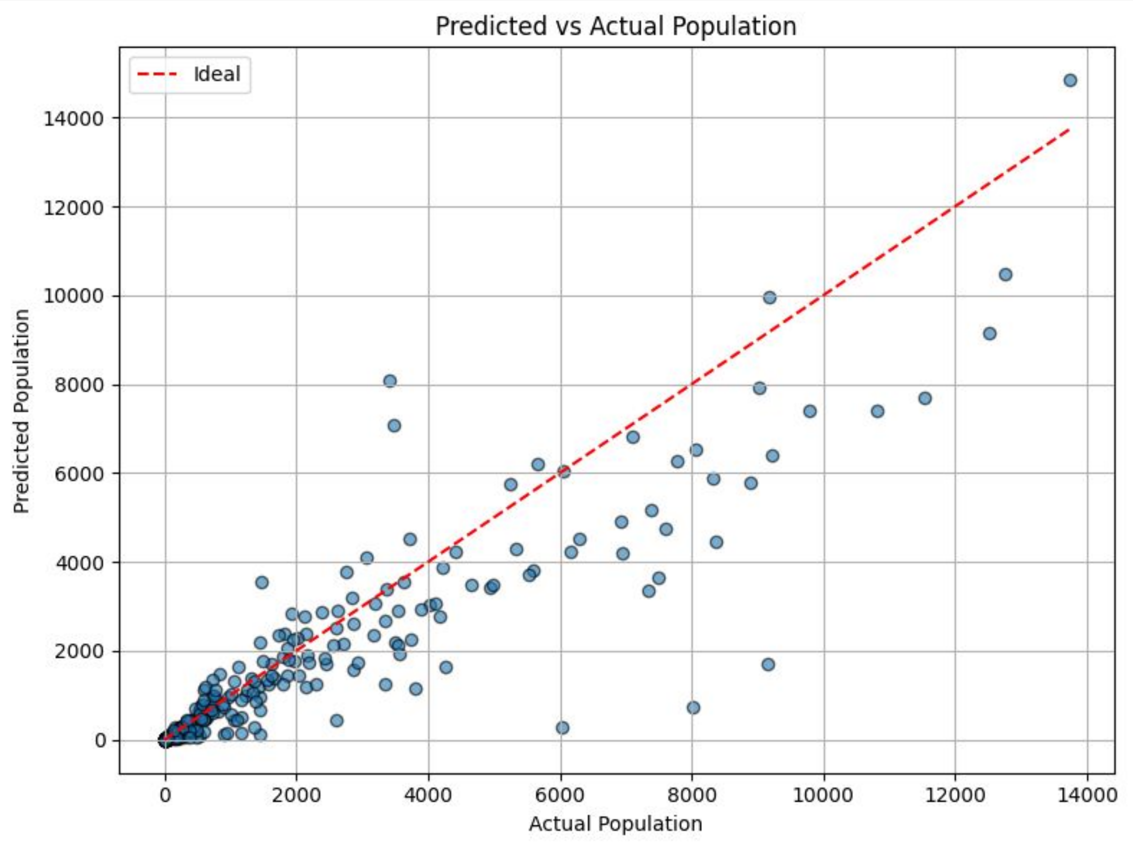
Pooling Layers:
MaxPool2D(kernel_size=3, stride=2, padding=1) after initial conv layer
AdaptiveAvgPool2D before the final fully connected layer

Dropout (Regularization):
None by default in ResNet50 architecture (unless you added manually)

Final Layer:
Fully Connected Linear layer replacing original classification head:
nn.Linear(in_features=2048, out_features=1)

With and without the log Malta

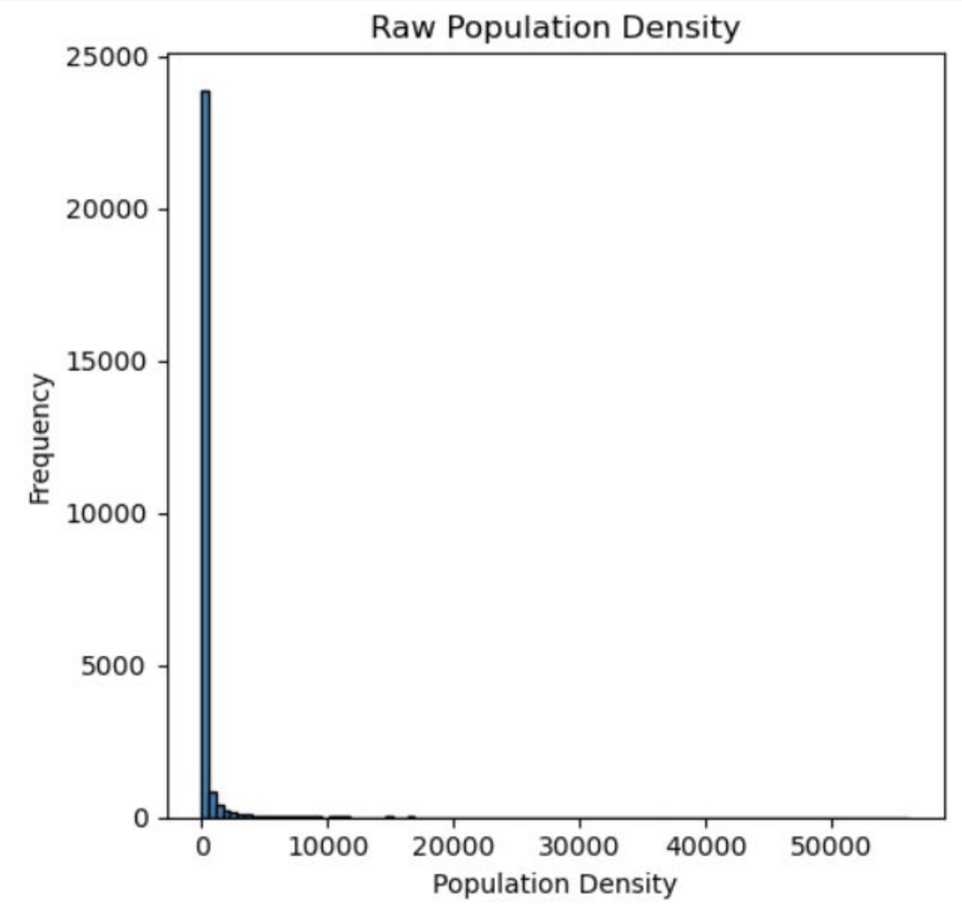


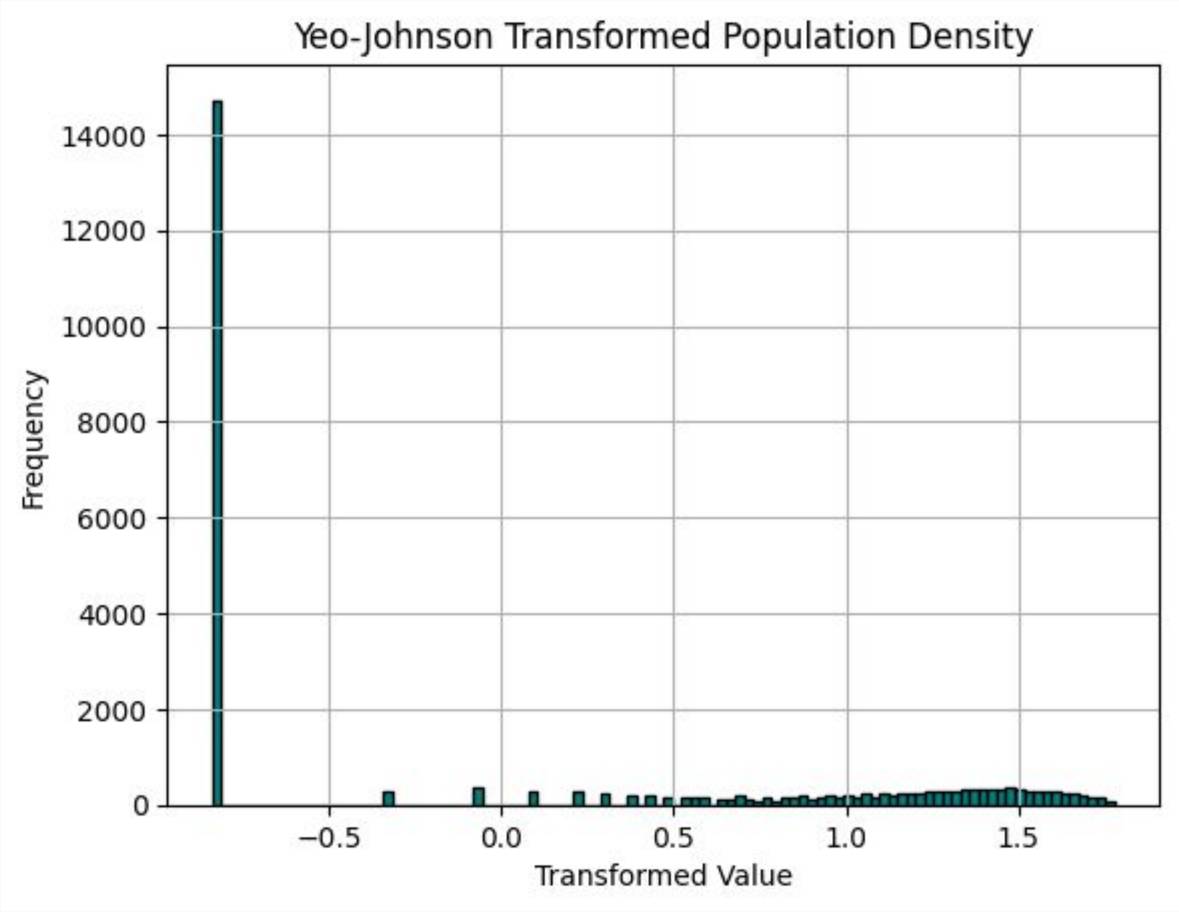


Dataset and transformations

- Same as general distributions. Heavily skewed

```
Total tiles: 26519
0 ≤ population ≤ 1: 15002 tiles
1 ≤ population ≤ 10: 2319 tiles
10 ≤ population ≤ 100: 3874 tiles
100 ≤ population ≤ 1000: 3859 tiles
1000 ≤ population ≤ 5000: 1300 tiles
5000 ≤ population ≤ inf: 611 tiles
```

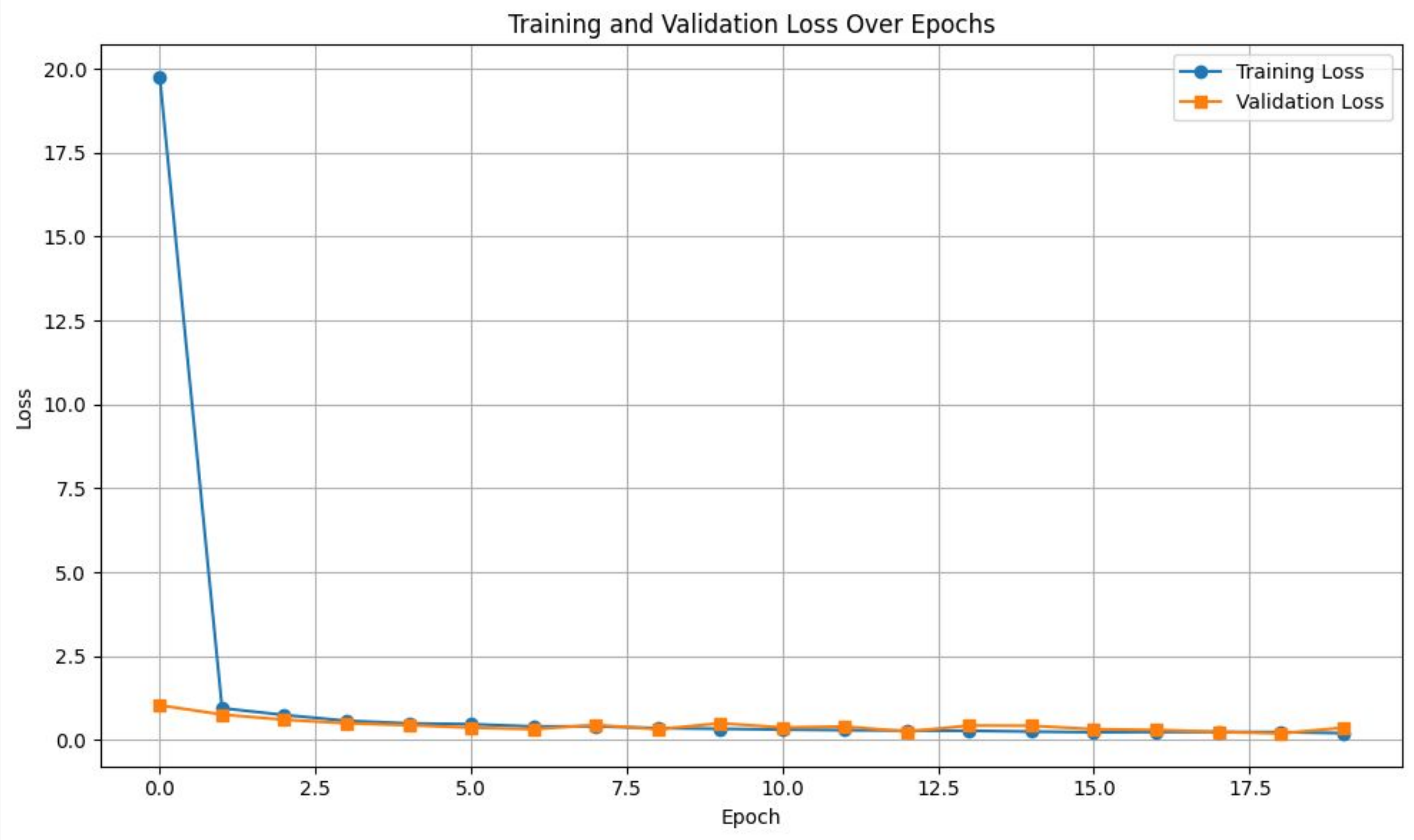


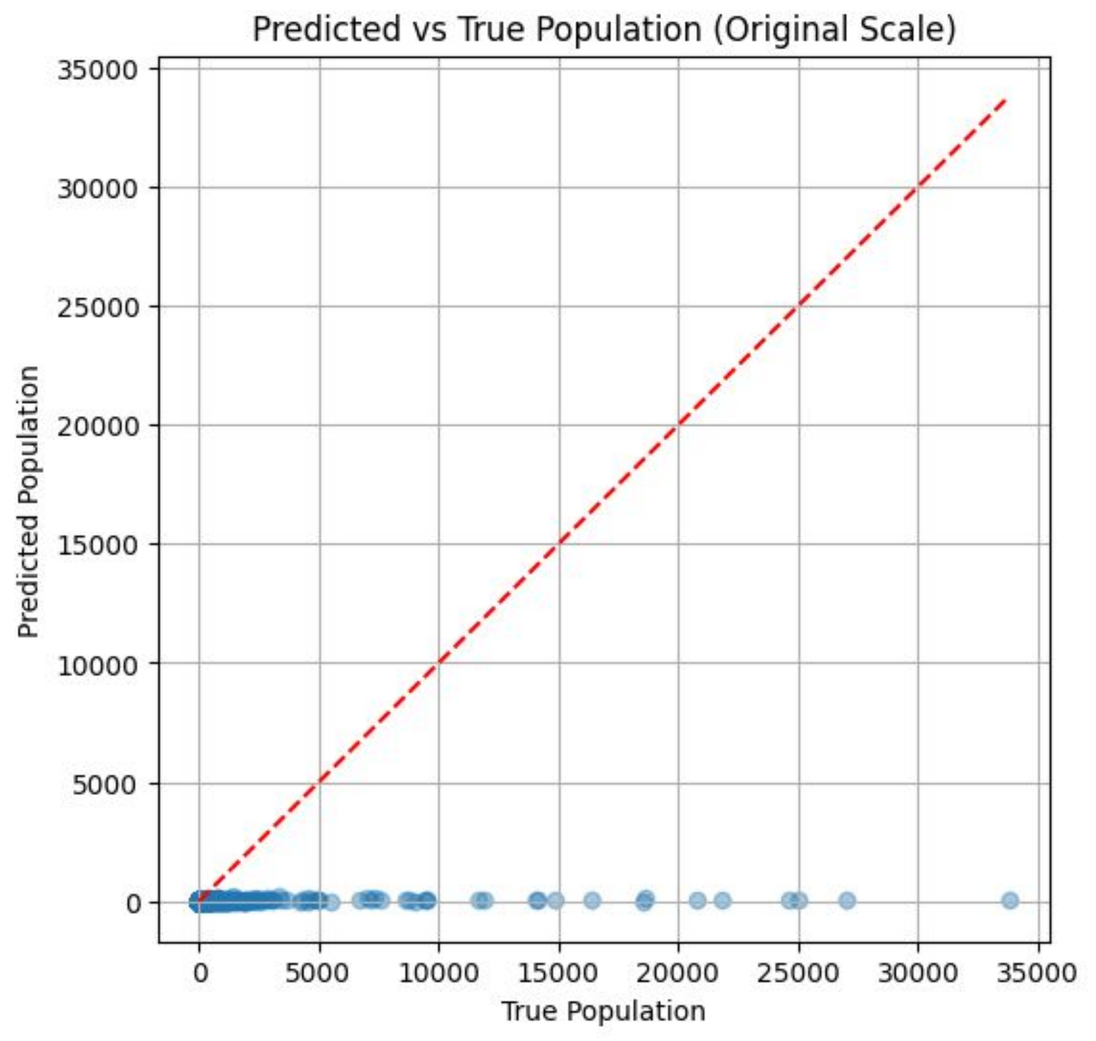


CNN(architecture)

```
Input (4 × 100 × 100)
      ↓
Conv2D (32 filters, 3×3) + BatchNorm + ReLU
      ↓
Conv2D (64 filters, 3×3) + BatchNorm + ReLU
      ↓
MaxPool2D (2×2)                     → 50×50
      ↓
Conv2D (128 filters, 3×3) + BatchNorm + ReLU
      ↓
MaxPool2D (2×2)                     → 25×25
      ↓
Conv2D (256 filters, 3×3) + BatchNorm + ReLU
      ↓
MaxPool2D (2×2)                     → 12×12
      ↓
Flatten
      ↓
Dense (512 units) + ReLU
      ↓
Dropout (p=0.3)
      ↓
Linear → Output: Single Population Value
```

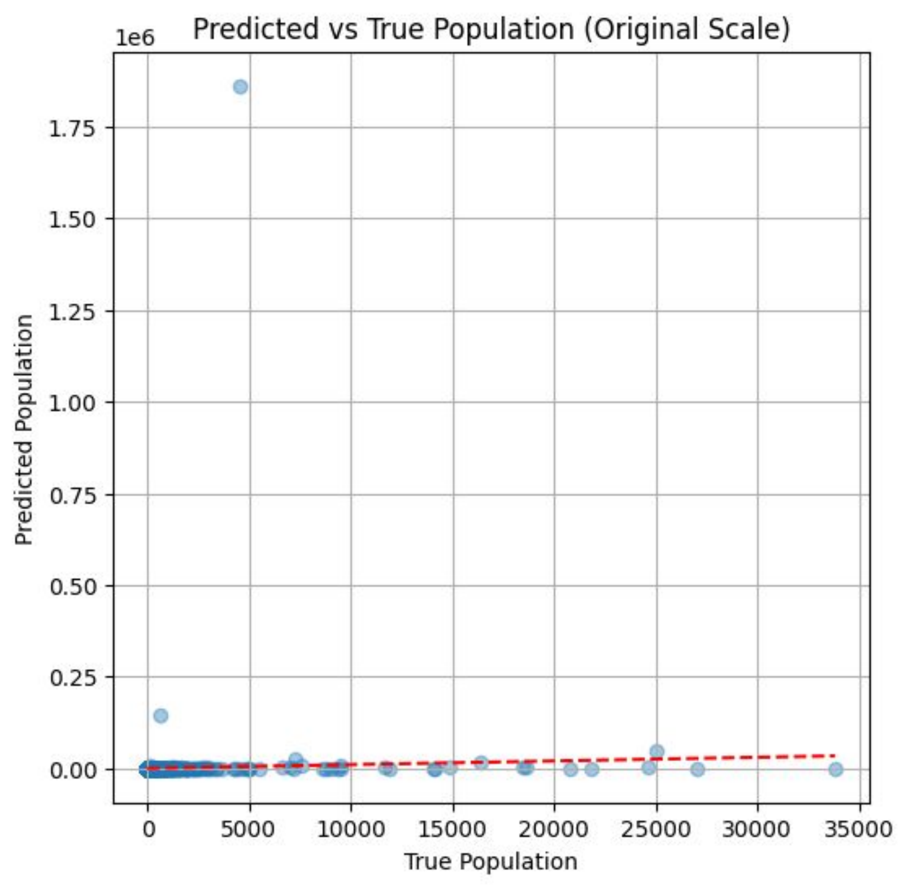
Initial Run

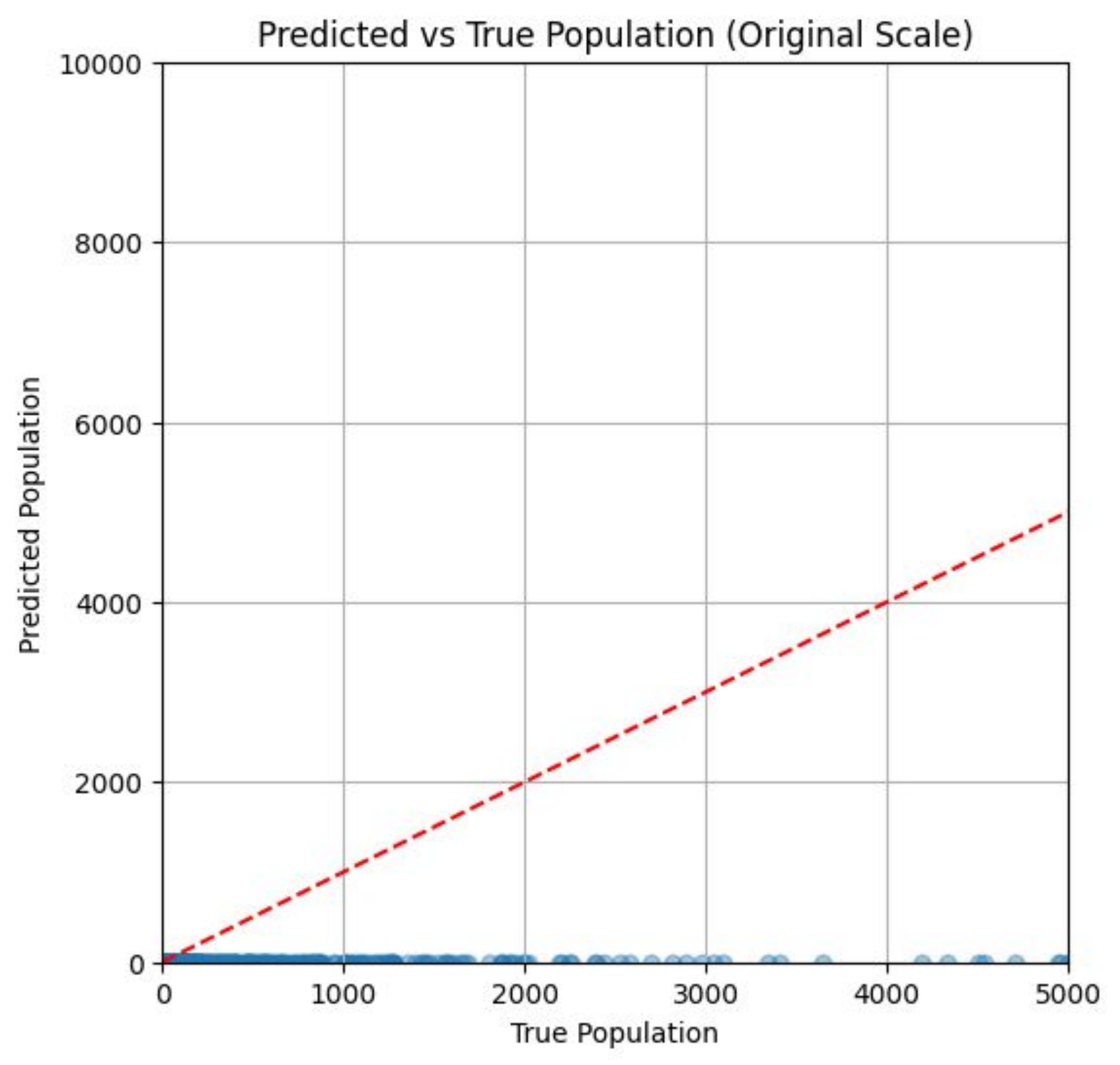




First try: 5600 train samples, 1400 val samples. Lossfunction:MSE,
Transform:Yeo-Johnson
Bands:[2,3,4,8](RGB,NIR)
😅Super good at predicting the 0-tiles!

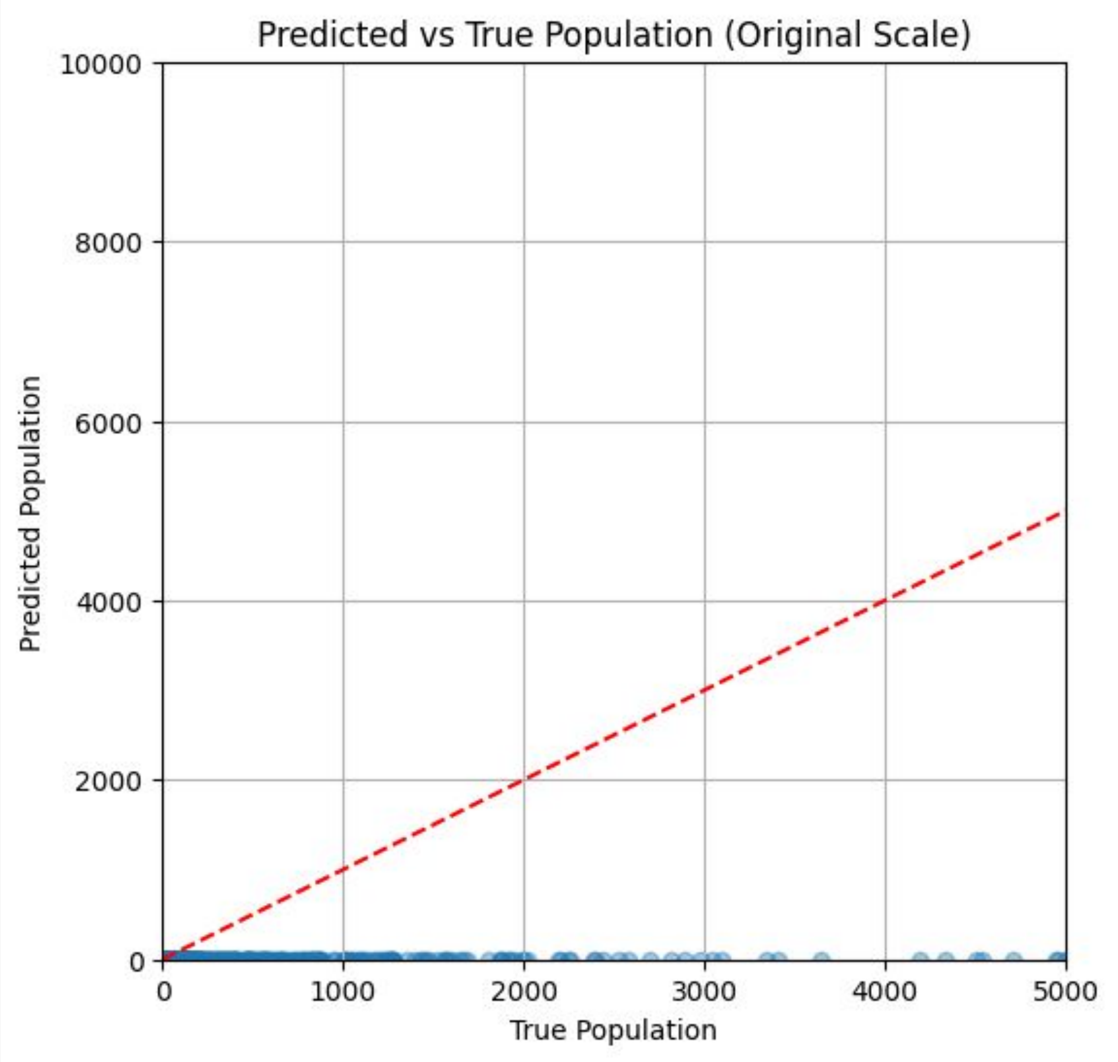
Second run

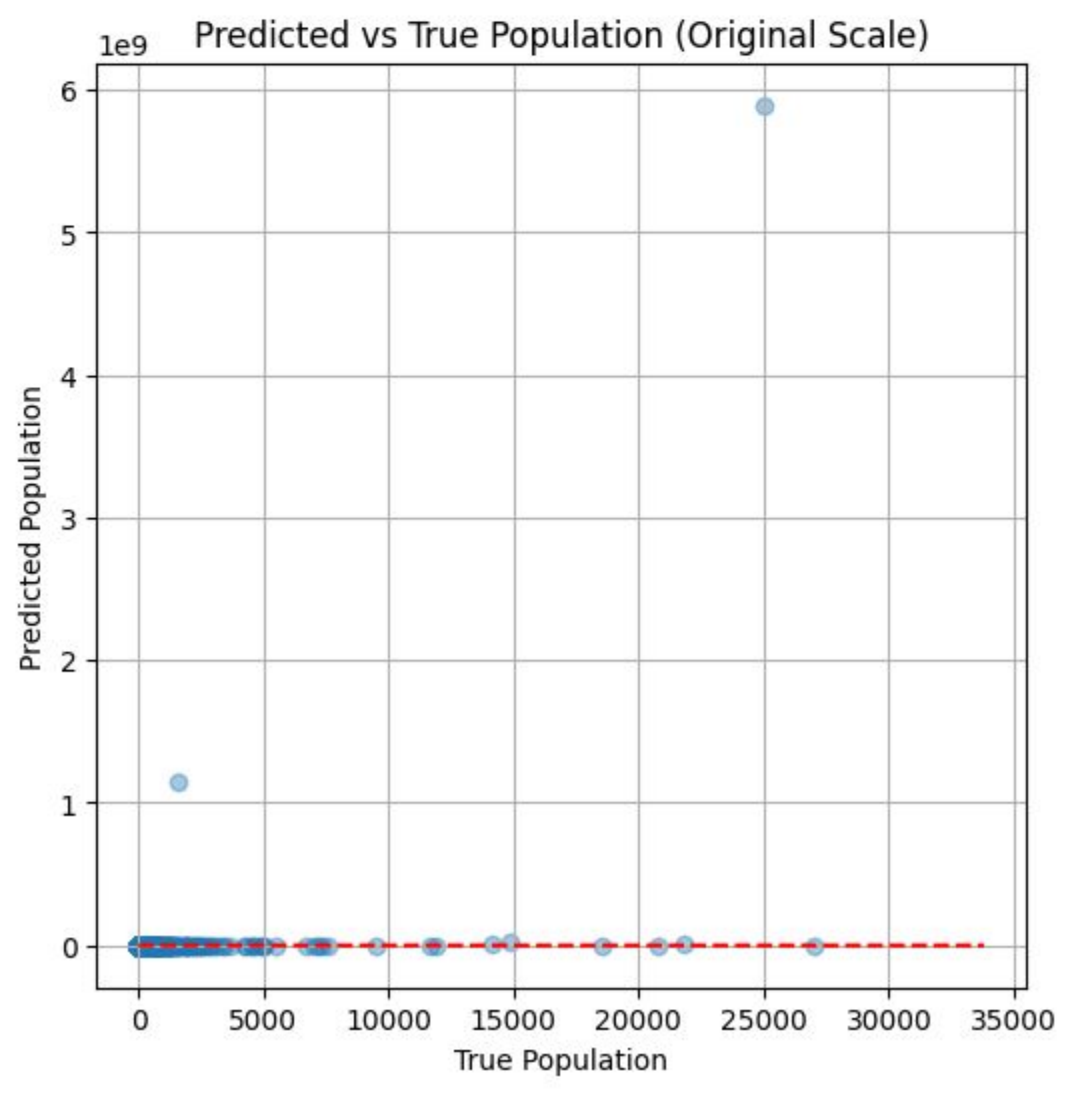


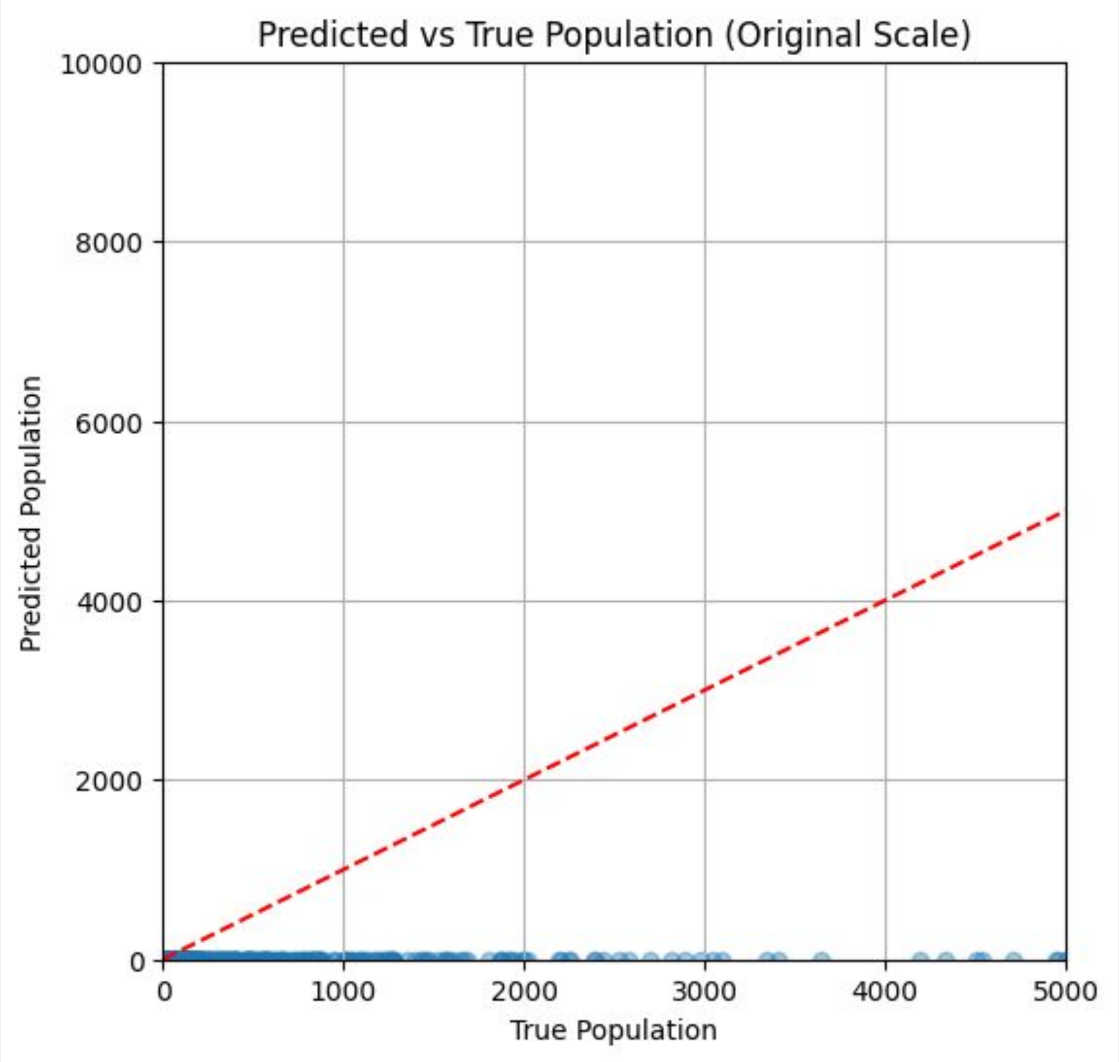


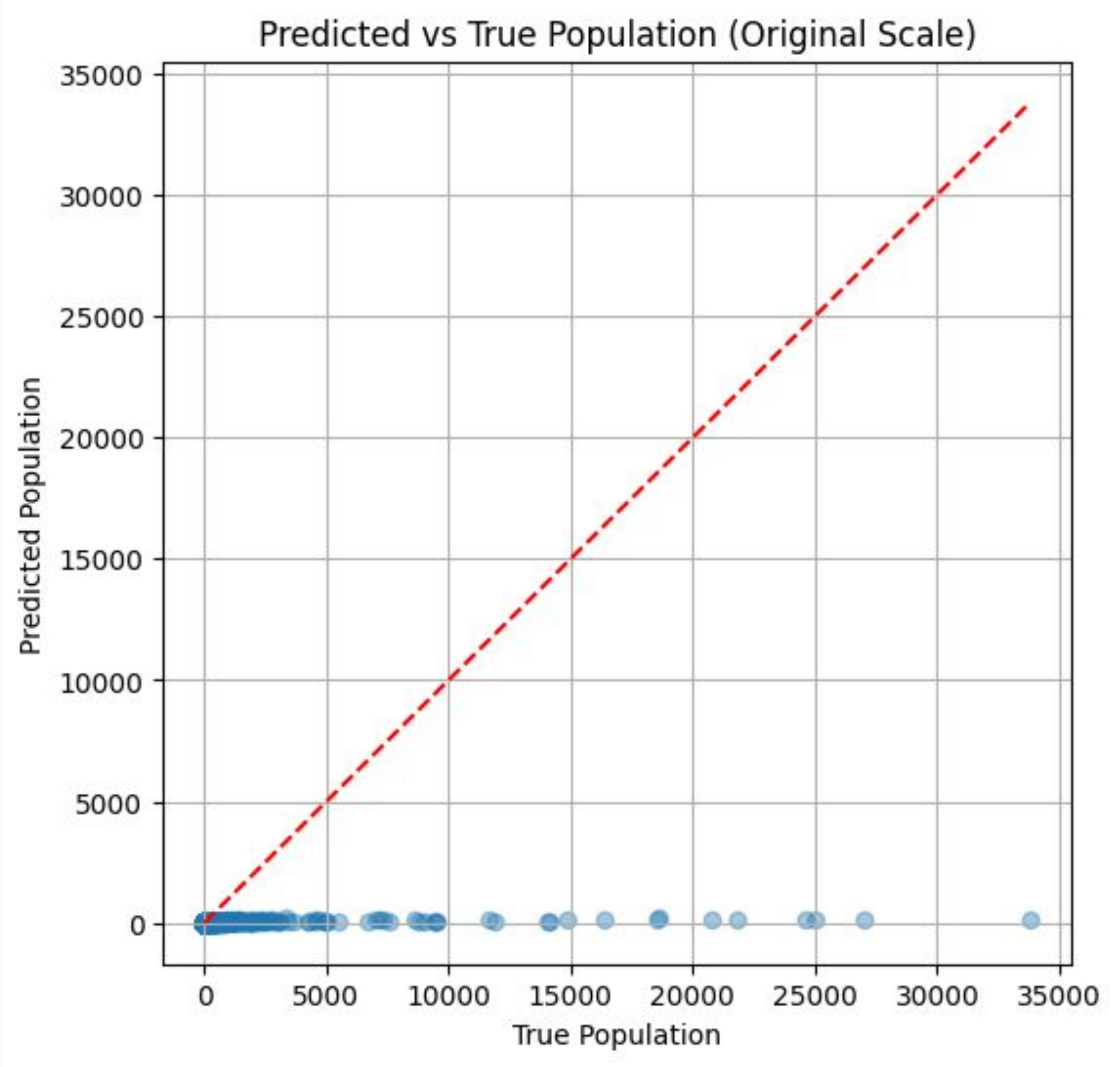
Same as before but now MSE with weights(from 1 to 5). Weights were calculated based on frequency in dataset.

Third run, same as second but with double data amount

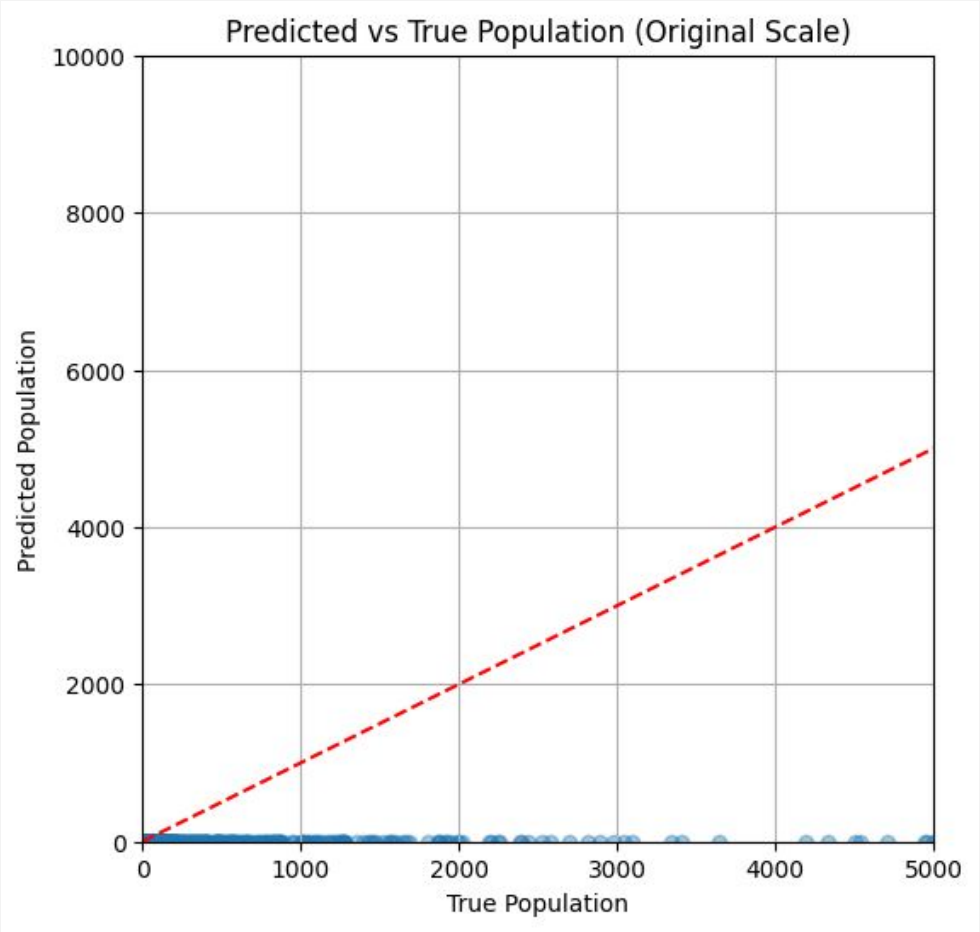


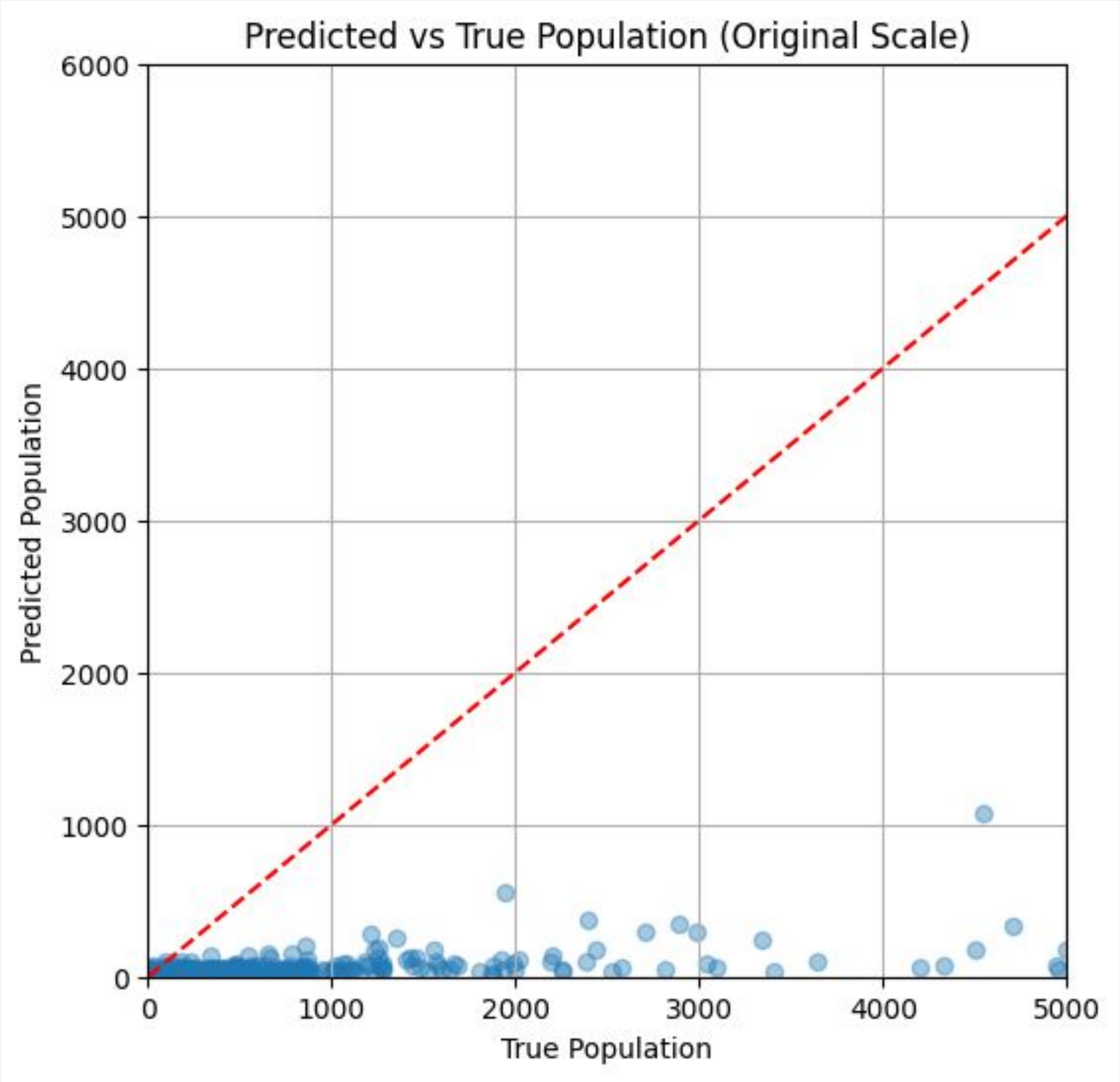






Same data amount as third run.
Standard MSE, Stratified sampling(Downsampled low population tiles,
upsampled high population tiles by data augmentation

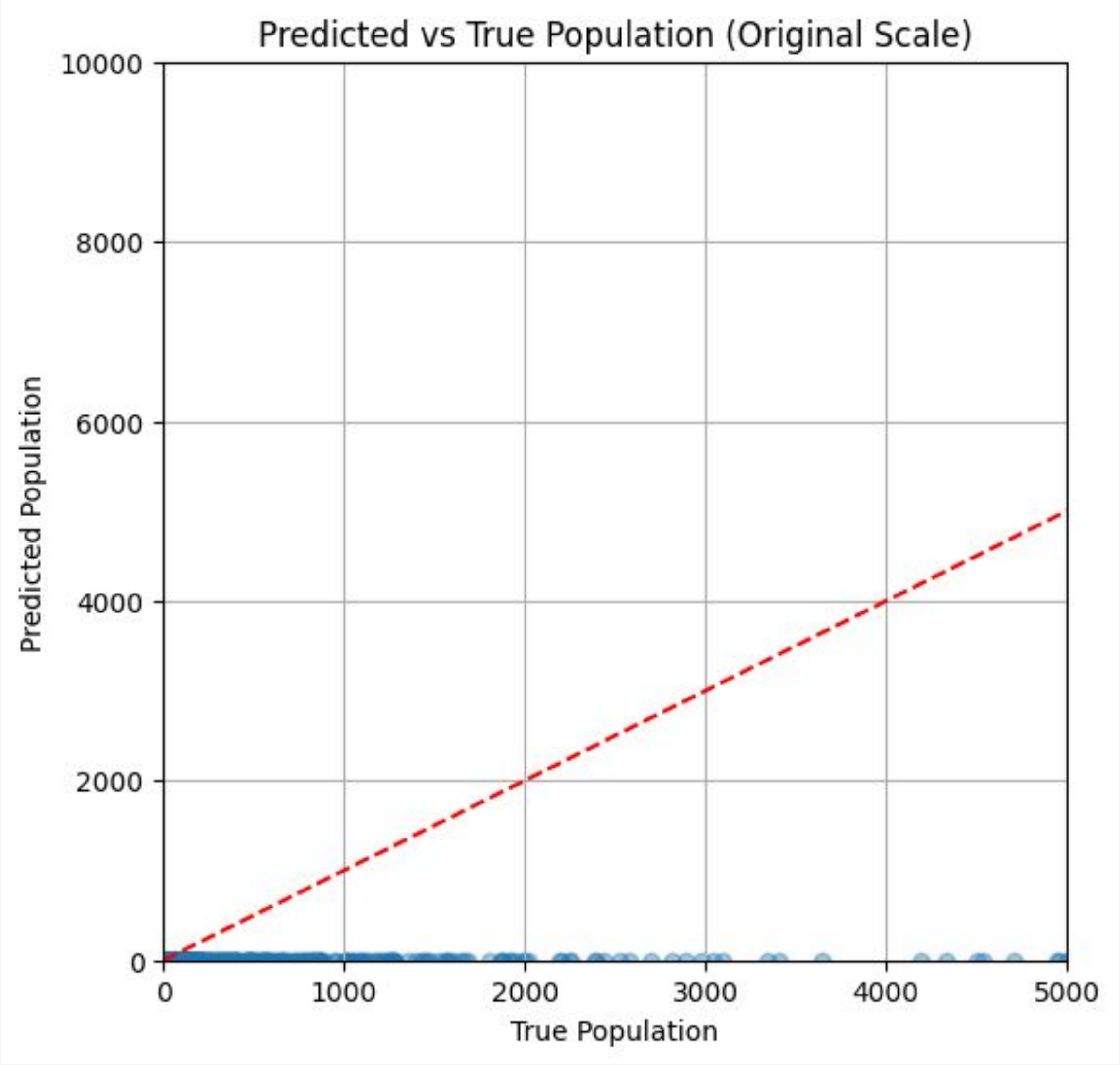


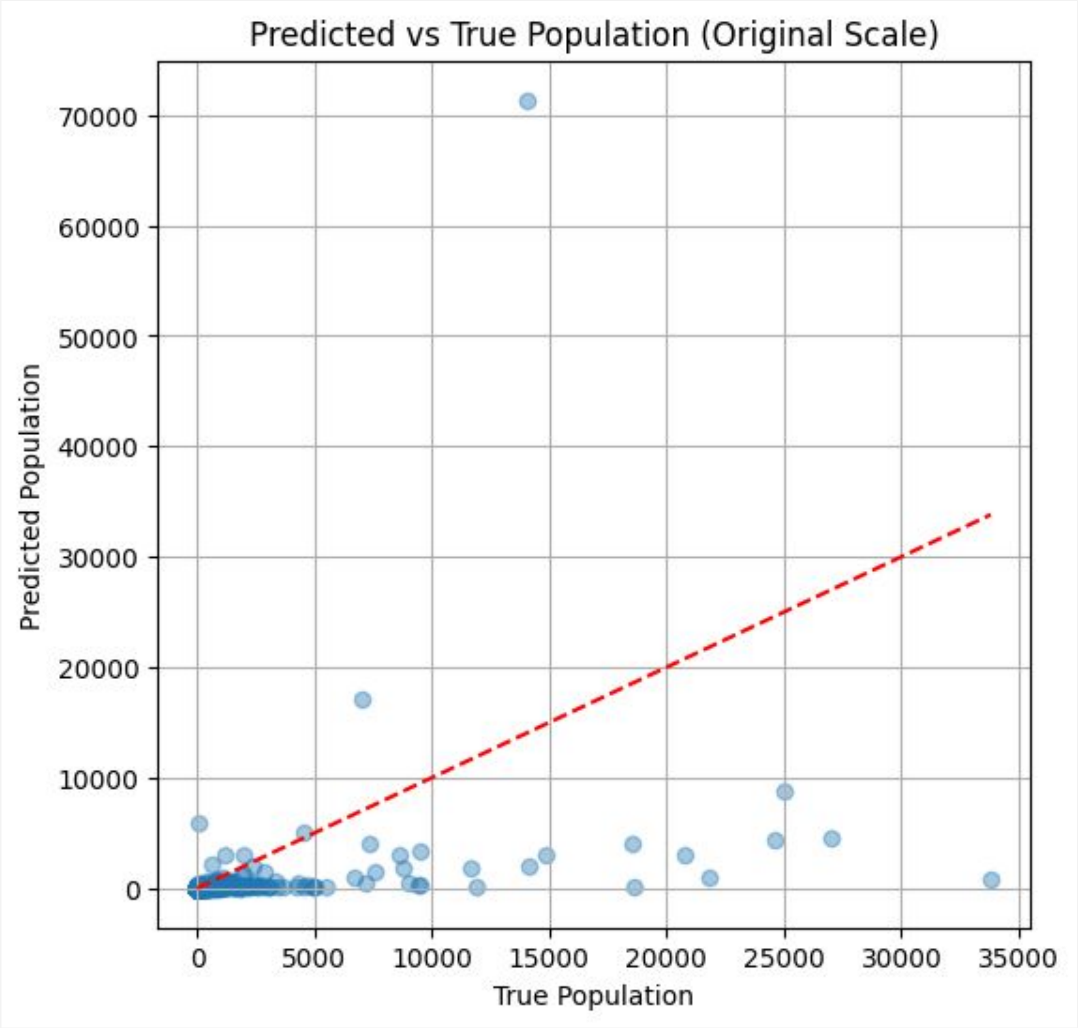


Same as previous run, but now with Huber loss function instead of MSE.

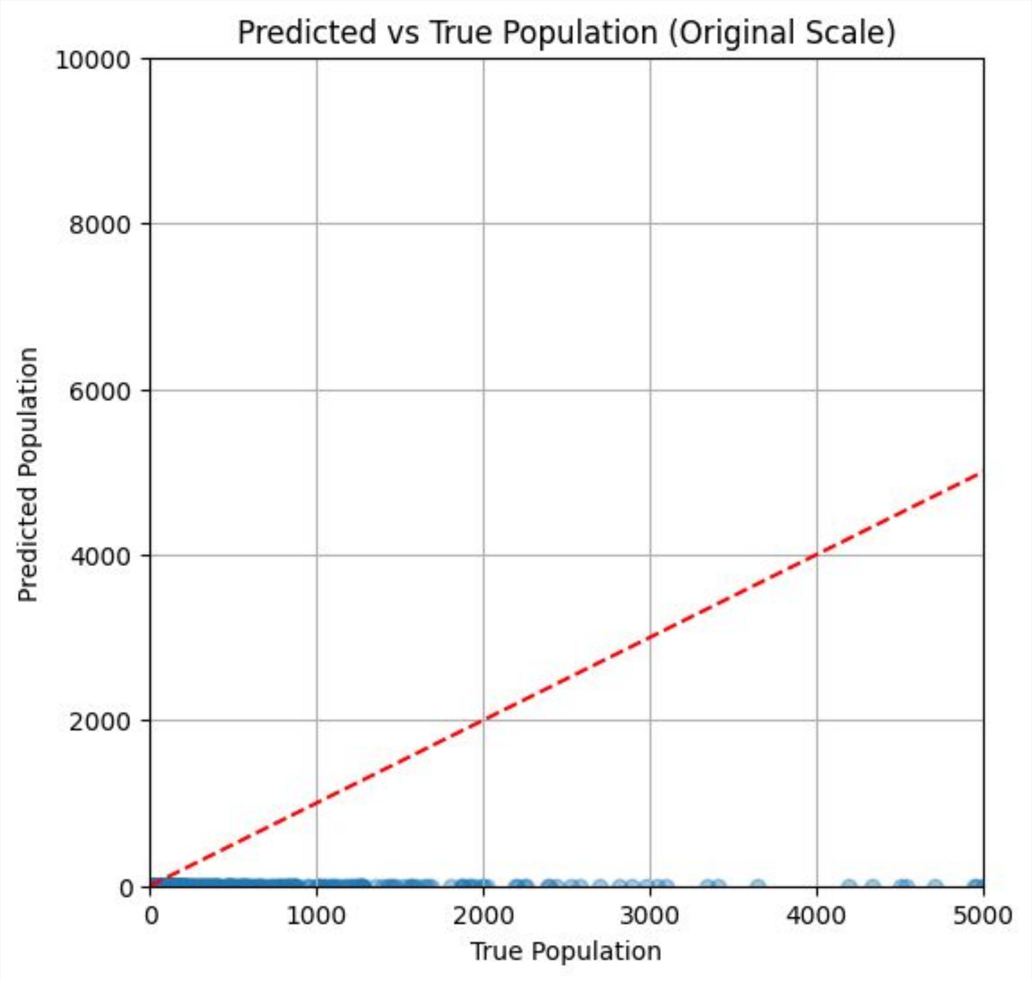
Lets try with this new simplified CNN

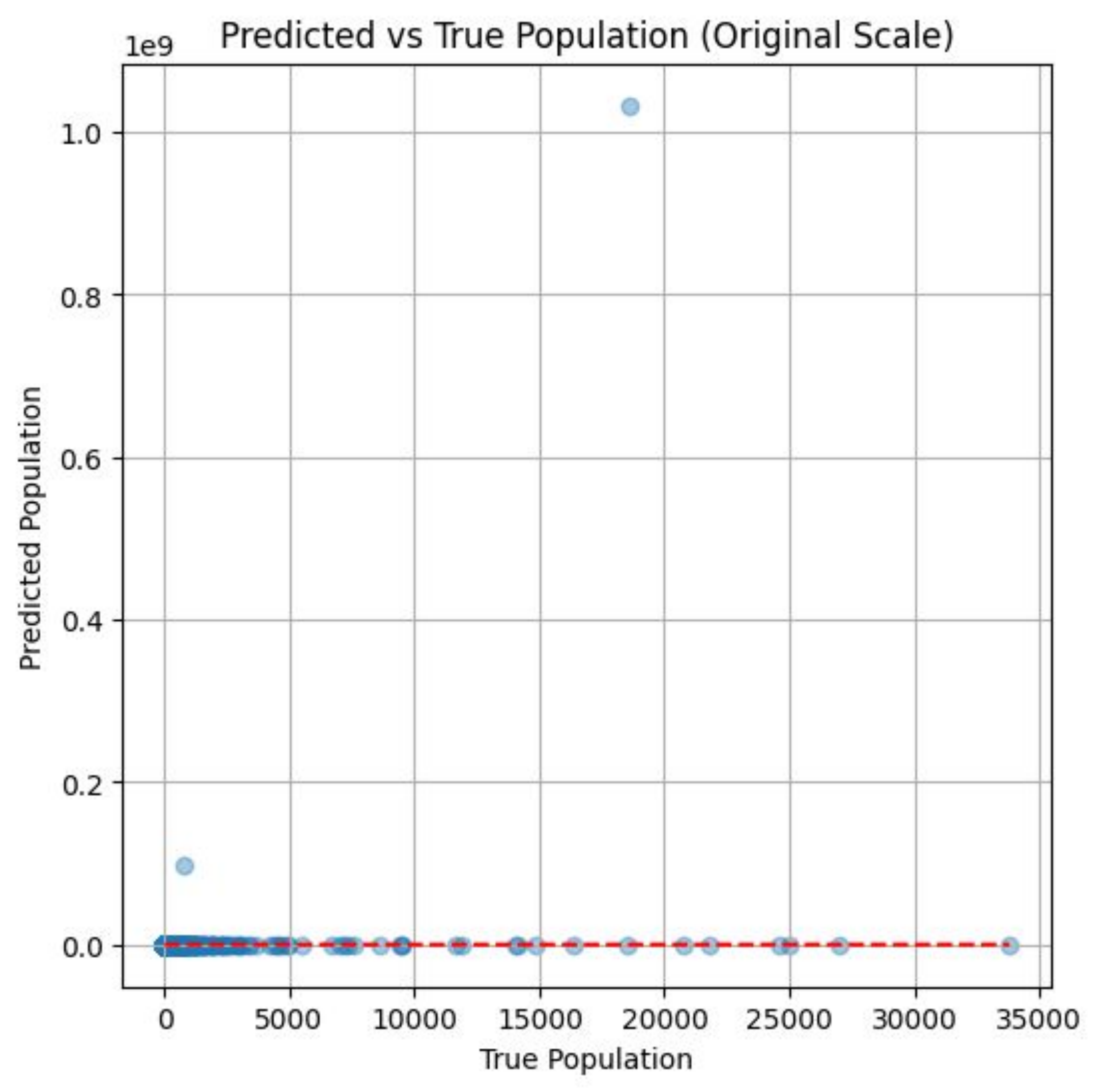
```
Input (4 × 100 × 100)
     ↓
Conv2D (32 filters, 3×3) + ReLU
     ↓
MaxPool2D (2×2)                → 50×50
     ↓
Conv2D (64 filters, 3×3) + ReLU
     ↓
MaxPool2D (2×2)                → 25×25
     ↓
Conv2D (128 filters, 3×3) + ReLU
     ↓
MaxPool2D (2×2)                → 12×12
     ↓
Flatten
     ↓
Dense (256 units) + ReLU
     ↓
Dropout (p=0.3)
     ↓
Linear → Output: Single Population Value
```





Same as before but now with a simplified CNN.





Same as before but now with all 9 Sentinel bands.

We did do many more runs than the ones showed, also with all 9 bands.

We could not get the model to perform well….

Take away

-Maybe with more parameter experimentation we could have gotten a model that worked.
-Probably use more data, if your computer have space and you have infinite time.
-Training time is tedious with images, so CNN from scratch is tough.
-Consider using ResNet instead:)

Hyperparameters and architecture 1.0

Hyperparameters

- Learning Rate : `1e-4`
- Batch Size : `16`
- Number of Epochs : `20`
- Optimiser : Adam optimizer (`torch.optim.Adam(model.parameters(), lr=1e-4)`)
- Weighted Huber Function Loss: `Smooth L1`

Model Architecture

- Convolutional Layers: `Conv2D(kernel_size=3, padding=1)`
- Increasing Filters: `64 → 128 → 256`
- Activation Function: Rectified Linear Unit (`ReLU()`)
- Batch Normalization: `BatchNorm2D()`
- Pooling Layers: `MaxPool2D(2)`
- Dropout (Regularization): `Dropout(0.4)` and `Dropout(0.3)`

Hyperparameters and architecture 3.0

Hyperparameters

- Learning Rate : `3e-5`
- Batch Size : `128`
- Number of Epochs : `15`
- Optimiser : Adam optimizer (`torch.optim.Adam(model.parameters(), lr=3e-5, weight_decay=1e-4)`)
- Number of workers: `8`

Model Architecture

- Convolutional Layers: `Conv2D(kernel_size=3, padding=1)`
- Increasing Filters: `64 → 128 → 256`
- Activation Function: Rectified Linear Unit (`ReLU()`)
- Batch Normalization: `BatchNorm2D()`
- Pooling Layers: `MaxPool2D(2)`
- Dropout (Regularization): `Dropout(0.6)` and `Dropout(0.5)`

CNN: From scratch vs ResNet-50

<u>CNN:</u>

- Standard for training on **images**.
- Built with **convolution**, **pooling**, **activation layers**.

<u>Foundation model ResNet-50:</u>

- Pre-trained model on **>1e6 images** from ImageNet dataset.
- Solves the **vanishing gradient problem** through skip connections.
- **50-layer deep** CNN with **bottleneck** design.