# III. Model documentation and write-up

## 1. Who are you (mini-bio) and what do you do professionally?

I am currently working as a software engineering in a start-up with 10 years of professional experience. I developped real-time bidding algorithms for internet advertising, developped management models for mutual funds, and analyzed insurance claims.

## 2. What motivated you to compete in this challenge?

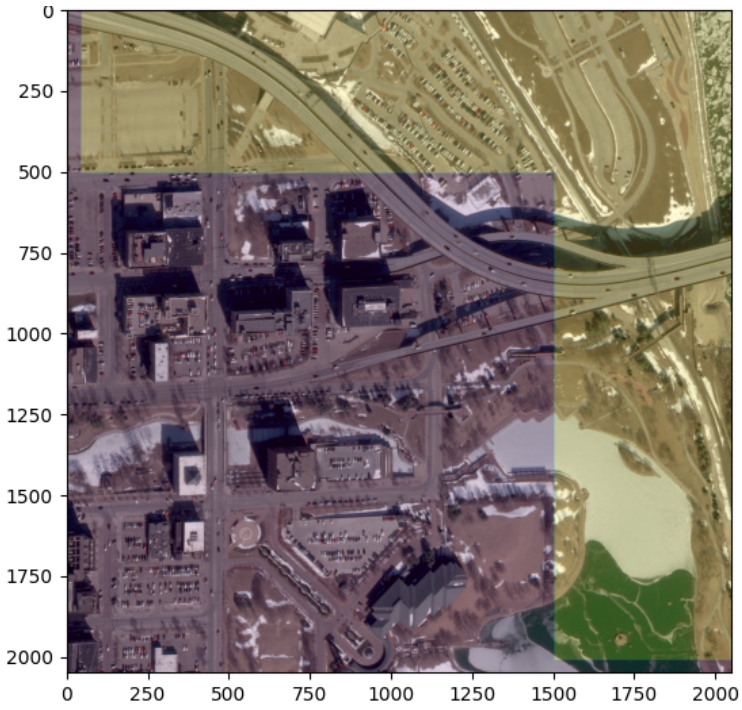
To learn about the techniques for analyzing satellite images.

## 3. High level summary of your approach

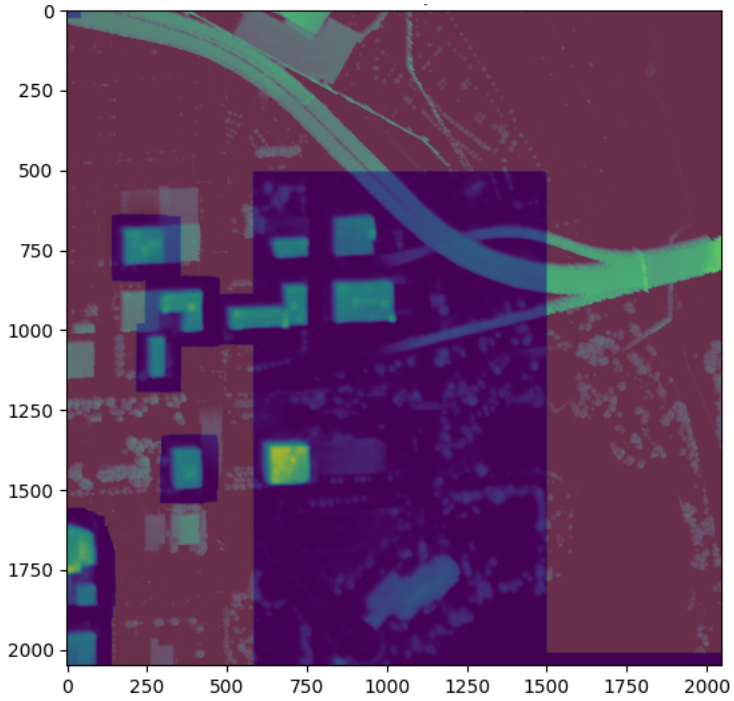
I ensembled the VFlow-UNet model using a large input resolution and a large backbone without downsampling. Better results were obtained when training models with all images from the training set. The test set contains images of the same location as the images in the training set. This overlaps was identified by image matching to improve the prediction results.

## 4. Do you have any useful charts, graphs, or visualization from the process?

The yellow overlaid area in the figure below is the area where image matching finds training images with high inlier count and small difference in RGB pixels. In this area, my solution uses the corresponding AGL label to make predictions.




If the images are taken at different incident angles or in different seasons, the difference in RGB pixels will be large even if the inlier count by image matching is high. My solution uses this kind of matching result too. For tall buildings, the difference in AGL is also larger due to the different incident angles. Therefore, if the RMSE in RGB was high, my solution uses the labels of the training image to predict only the region around the low building. In the figure below, the regions overlaid in yellow are predicted using the labels of the corresponding AGL.




## 5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

### Large backbone

In the baseline code, “resnet34” is used as the backbone. I used “timm-regnety_120” and “timm-resnest101e” instead of “resnet34”.

```
args = argparse.Namespace(
    backbone=”timm-resnest101e”,
    <snip>
)

def build_model(args):
    model = UnetVFLOW(args.backbone, encoder_weights="imagenet")
```

### Large input size without downsampling

In the baseline code, the image size is downsampled from 2048x2048 to 1024x1024. Instead of that, I randomly cropped 1024x1024 images without downsampling.

```
image = load_image(rgb_path, self.args)
agl = load_image(agl_path, self.args)
mag, xdir, ydir, vflow_data = load_vflow(vflow_path, agl, self.args)

If self.args.random_crop:
    x0 = np.random.randint(2048 - self.crop_size)
    y0 = np.random.randint(2048 - self.crop_size)
    image = image[y0:y0+self.crop_size, x0:x0+self.crop_size]
    agl = agl[y0:y0+self.crop_size, x0:x0+self.crop_size]
    mag = mag[y0:y0+self.crop_size, x0:x0+self.crop_size]
```

### Image Stitching

Image matching is computationally expensive, so all matching results are saved in an intermediate file. The following code loads the pre-computed perspective projection matrix and use it to update the predicted AGL files.

```
# Load predicted AGL
agl1 = u.load_image(Path(base_dirname) / f"{query_image_id}_AGL.tif")
agl1 *= 100.0

for _, r in df_part.iterrows():
    hit_image_id = r["matched_image_id"]
    fn_cmp_h = Path(f"data/working/match/{query_image_id}_{hit_image_id}_matchH.npy")
    cmp_H = np.load(str(fn_cmp_h))
    mask = cv2.warpPerspective(np.ones(agl1.shape), cmp_H, rgb2.shape[:2])
    agl2 = u.load_image(Path(f"data/input/train/{hit_image_id}_AGL.tif"))
    agl2 *= 100.0
    agl2_transformed = cv2.warpPerspective(agl2, cmp_H, agl2.shape[:2])
    agl2_mask = mask[..., 0]
    agl2_mask = np.maximum(agl2_mask - np.isnan(agl2_transformed).astype(np.uint8), 0)
    agl1[agl2_mask == 1.0] = agl2_transformed[agl2_mask == 1.0] * 1.0

agl1_out = Image.fromarray(agl1.astype("uint16"))
agl1_out.save(fn_out, "TIFF", compression="tiff_adobe_deflate")
```

## 6. Please provide the machine specs and time you used to run your model.

- CPU (model): Ryzen Threadripper 3970X
- GPU (model): RTX3090 *2
- Memory (GB): 256 GB

- OS: Ubuntu 20.04.2.0 LTS
- Train duration: 180 hours. 60 hours with single GPU per model.
- Inference duration: Almost 4 days. Most of it was done with CPUs for image matching.

## 7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

The calculation time for image matching by CPU is very long. Even if you use a powerful workstation processor, this step requires several days.

## 8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

Jupyter notebook is used for exploratory data analysis.

## 9. How did you evaluate performance of the model other than the provided metric, if at all?

To determine the hyperparameters and for model selection, I evaluated model performance of the model with R^2 of AGL.

## 10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

We tried to use similar test images to predict the AGL of the test image, but the score got worse.

## 11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

Ensemble of rectified AGLs of the same location may make the prediction of AGLs more accurate.