Self-Supervised Representation Learning Approach to 6DoF Pose Estimation

By: Maahum Khan

March 28, 2025

ELEC 874 Final Project

Queen's UNIVERSITY

Background: The Direct Method with PoseNet

- February, 2016
- Claimed to be the first use of regression in estimating the pose of a camera from a monocular image
- Used GoogLeNet architecture as a backbone
- Did affine regression in the final layers instead of softmax classification
- Use of transfer learning in 6DoF pose estimation
  - Apply a solution (GoogLeNet) of a task (image classification) to a new task (pose estimation)

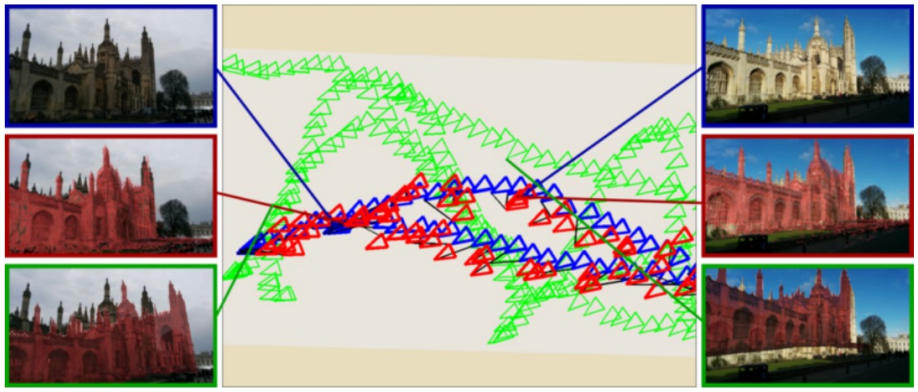


Figure 3: Magnified view of a sequence of **training (green)** and **testing (blue)** cameras for King's College. We show the **predicted camera pose in red** for each testing frame. The images show the test image (top), the predicted view from our convnet overlaid in red on the input image (middle) and the nearest neighbour training image overlaid in red on the input image (bottom). This shows our system can interpolate camera pose effectively in space between training frames.

Problem Statement

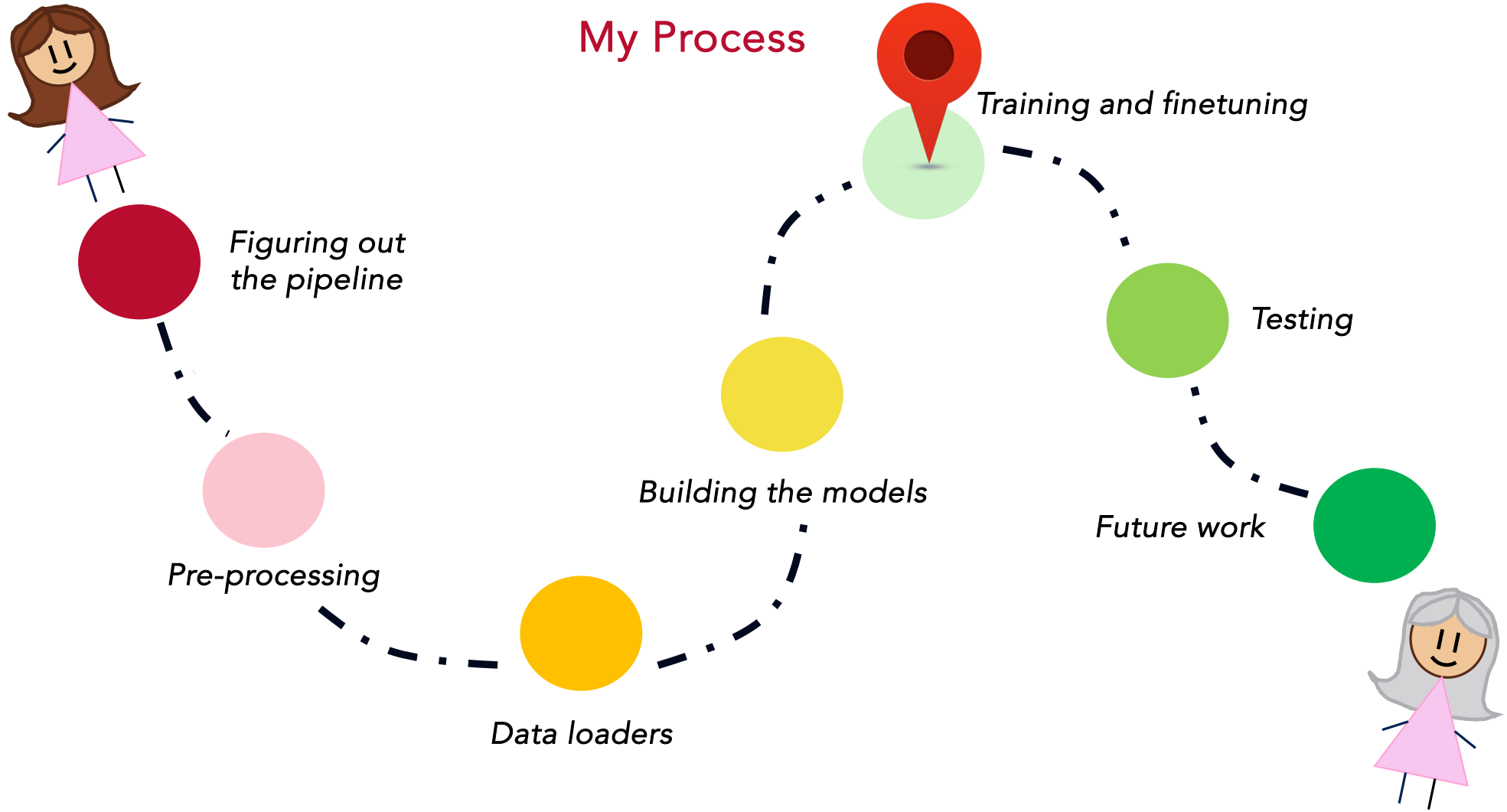
Significant roadblock with 6-degrees of freedom pose estimation:

Difficulty obtaining 6DoF pose annotations

Problem Statement:

Create a self-supervised approach to solving 6DoF pose estimation that requires a significantly smaller amount of annotated data, and takes advantage of latent space organization.

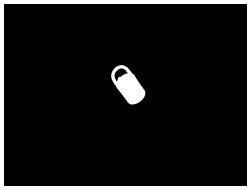
My Process


Figuring out the pipeline
Pre-processing
Data loaders
Building the models
Training and finetuning
Testing
Future work

Pipeline

RGB image

mask



Pre-Processing

BB-centered RGB image

x all images

training

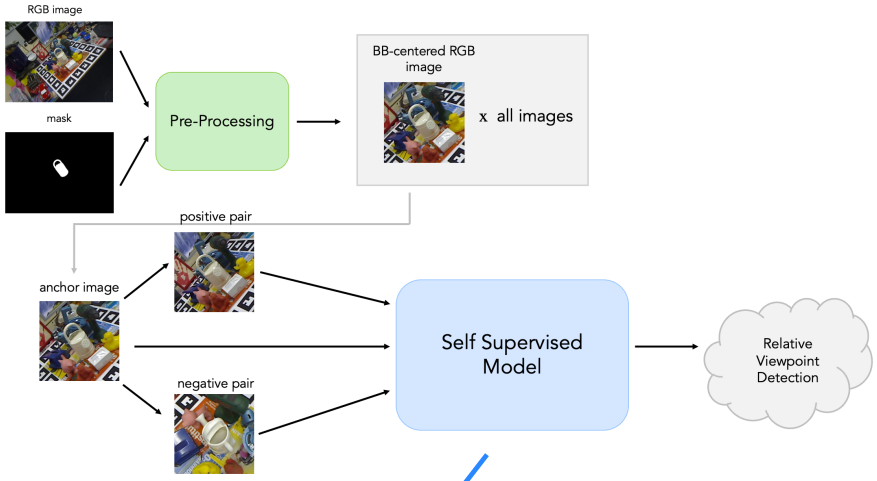
positive pair

anchor image

negative pair

Self Supervised Model

Relative Viewpoint Detection


RGB image
mask
Pre-Processing
BB-centered RGB image
x all images
positive pair
anchor image
negative pair
Self Supervised Model
Relative Viewpoint Detection

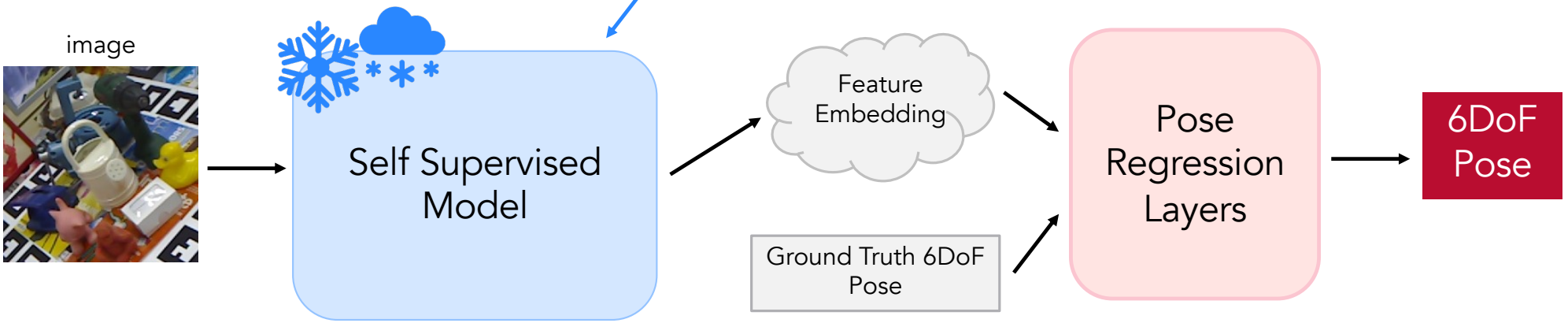
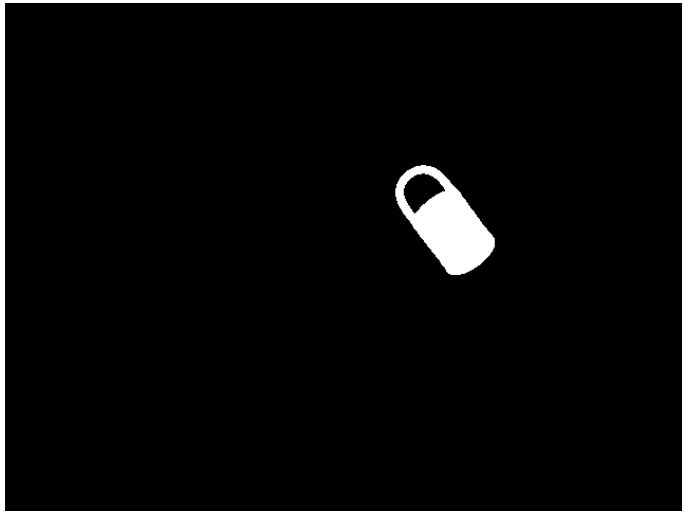


image
Self Supervised Model
Feature Embedding
Ground Truth 6DoF Pose
Pose Regression Layers
6DoF Pose

Pre-Processing

- Used *masks* as object detection alternative
- Masks came with LINEMOD dataset
  - Can also easily be extracted in real-world applications using semantic segmentation methods
- Many existing semantic segmentation solutions, and results are much easier to obtain than annotating 6DoF pose on images
- ***Idea:*** CNN architectures (e.g. ResNet, VGG) require 224x224 input, so use masks to get the bounding boxes of the objects and create 224x224 image focusing on it



Pre-Processing

Initial Solution: Create a bounding box around the object using contours from the mask, resize the bounding box to fit 224x224 input image size

Issue: Distorted image — could consequently distort features, pose, and the model's understanding of the object's nature

Pre-Processing

Final Solution: Create a bounding box around the object using contours from the mask, define a 224x224 image outline, align the center of the 224x224 crop with the center of the object's bounding box

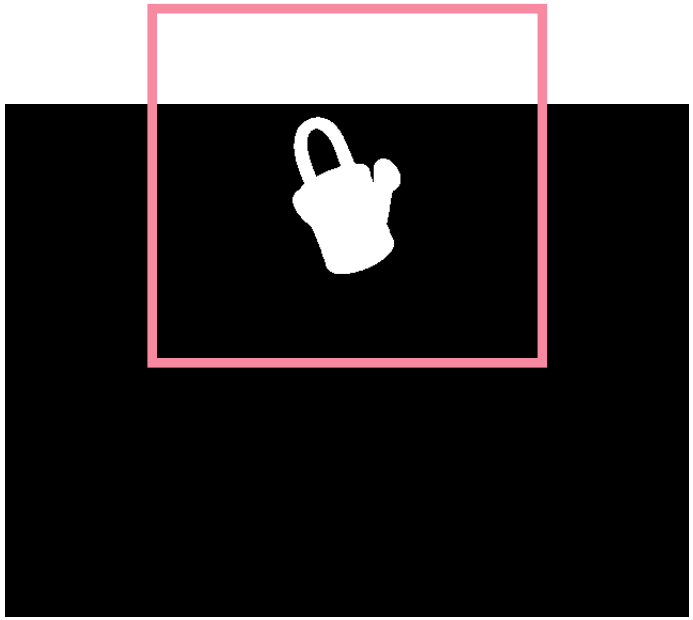
Pre-Processing

Previous Solution

Final Solution

Pre-Processing

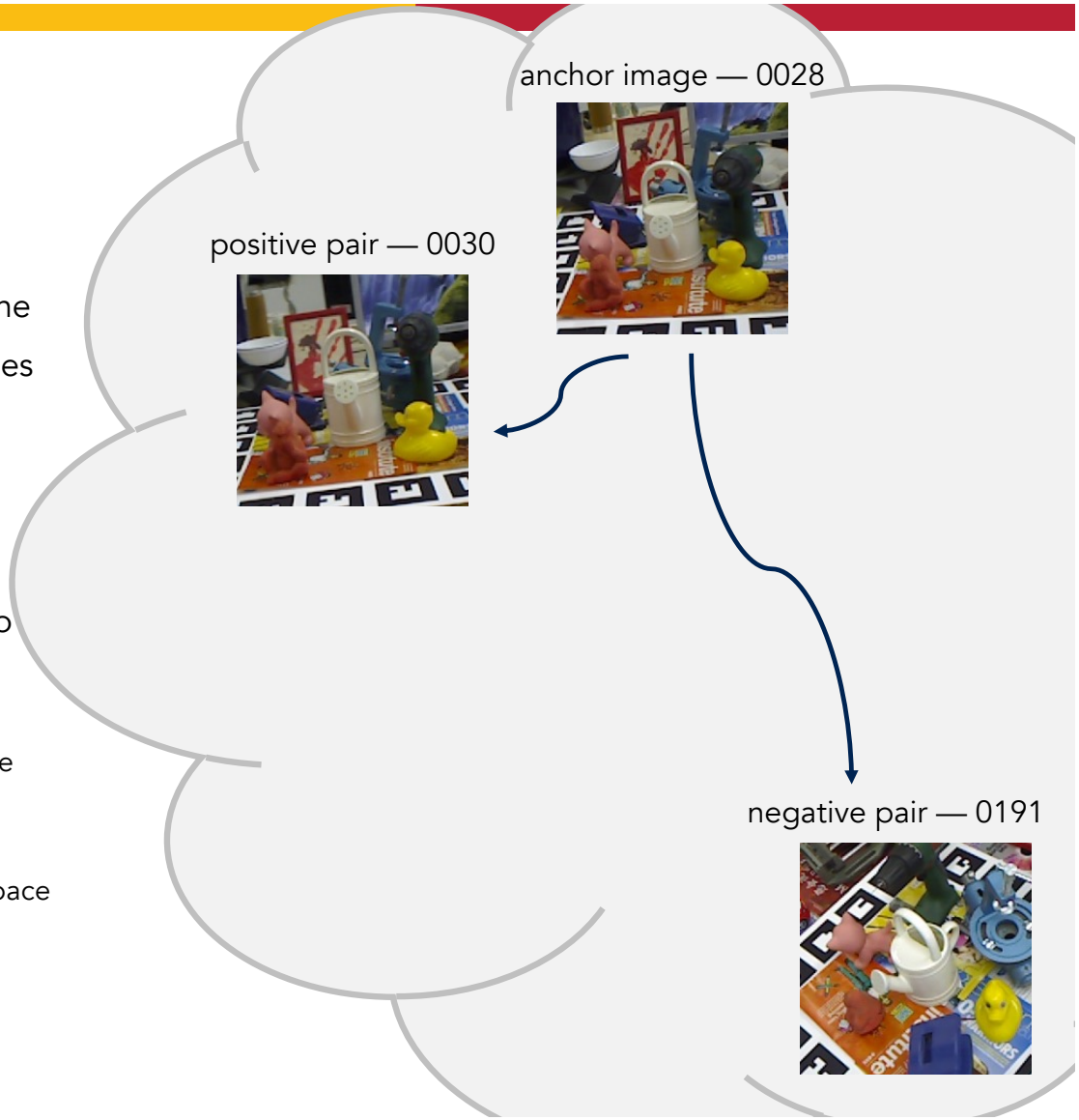
224 x 224



Code to adjust crop accordingly

The Model

- *Self-Supervised Model:*
  - *Task:* Relative Viewpoint Detection — understand the difference between the viewpoints in a pair of images
  - Used a ResNet18 backbone
  - Removed the softmax classification layer
  - Used Cosine Similarity in contrastive loss function to organize the latent space
    - Nearby viewpoints are closer in the latent space (positive pairs)
    - Farther apart viewpoints are farther away in the latent space (negative pairs)
- *Pose Regression Layers:* 2 FC layers + regression head



Selecting Pairs for Contrastive Loss

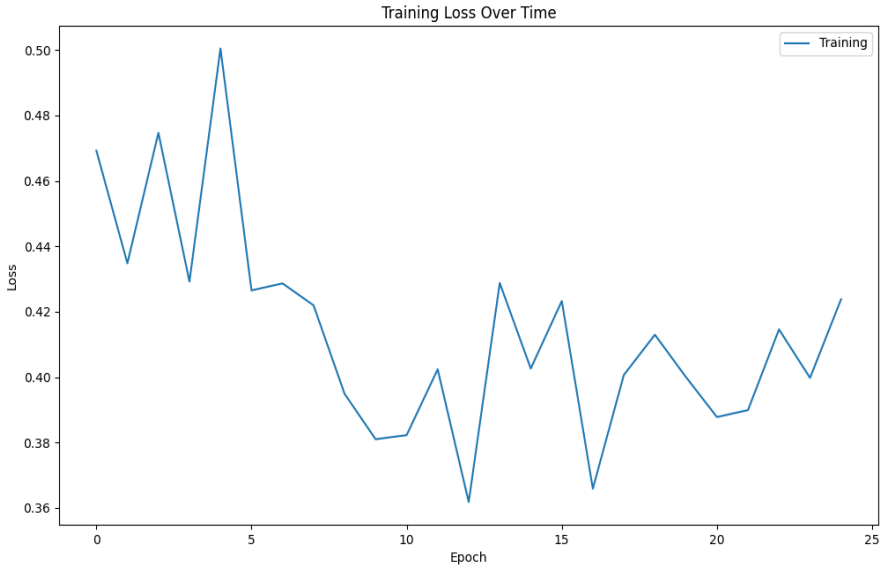
- Used randomly selected distance from anchor image + randomly selected direction from anchor image to increase robustness and variability

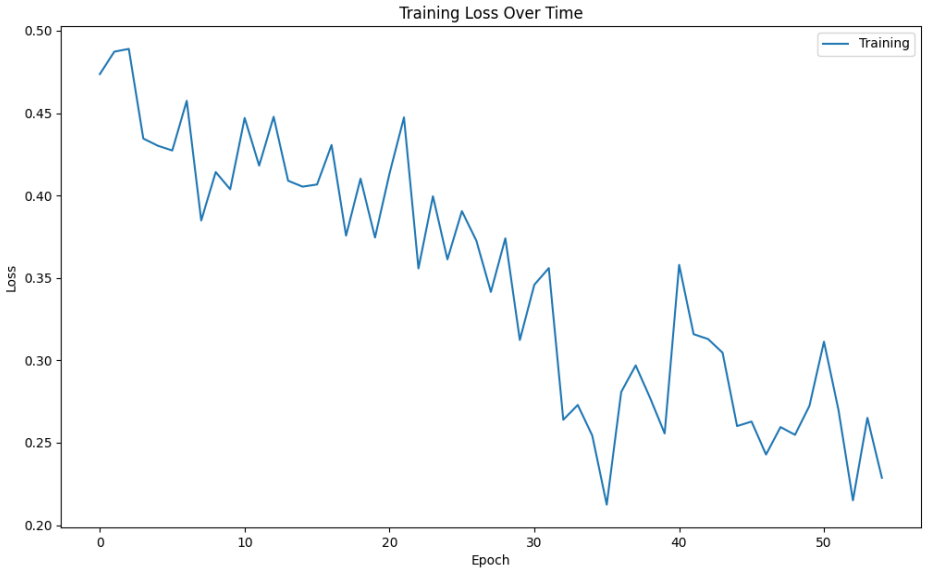
```python
# Get a valid index given the index, difference, and sign
2 usages
def get_valid_index(self, current_index, diff, sign):
    if sign == '+':
        new_index = current_index + diff
        # If the new index goes beyond the bounds, move backward instead
        if new_index >= self.num_imgs:
            new_index = current_index - diff
        return new_index
    elif sign == '-':
        new_index = current_index - diff
        # If the new index goes below zero, move forward instead
        if new_index < 0:
            new_index = current_index + diff
        return new_index
```

```python
# Get current anchor image
img_key = int(self.imgs_list[idx])
anchor_img_path = self.all_img_paths[img_key]
anchor_img = Image.open(anchor_img_path).convert("RGB")

# Select a positive pair (nearby image keys)
pos_diff = random.randrange( start: 1, stop: 4)
pos_sign = random.choice(['+', '-'])
pos_index = self.get_valid_index(img_key, pos_diff, pos_sign)
pos_img_path = self.all_img_paths[pos_index]
positive_img = Image.open(pos_img_path).convert("RGB")

# Select a negative pair (anything 20+ away but stays within index range)
neg_sign = random.choice(['+','-'])
if neg_sign == '+':
    if self.num_imgs - img_key > 20:
        neg_diff = random.randrange( start: 20, self.num_imgs - img_key)
    else:
        neg_sign = '-'
        neg_diff = random.randrange( start: 20, img_key)
else:
    if img_key > 20:
        neg_diff = random.randrange( start: 20, img_key)
    else:
        neg_sign = '+'
        neg_diff = random.randrange( start: 20, self.num_imgs - img_key)
neg_index = self.get_valid_index(img_key, neg_diff, neg_sign)
neg_img_path = self.all_img_paths[neg_index]
negative_img = Image.open(neg_img_path).convert("RGB")
```

Training & Fine-Tuning Self-Supervised Model

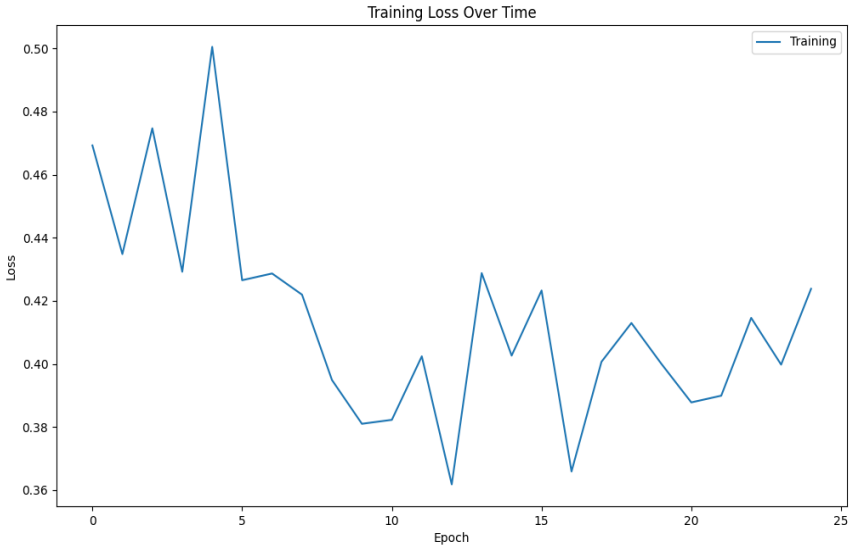
- Hyperparameters:
  - Number of epochs
  - Batch size
  - Learning rate
  - Embedding dimension
  - Size of training dataset
- Losses kept spiking and wouldn't stop



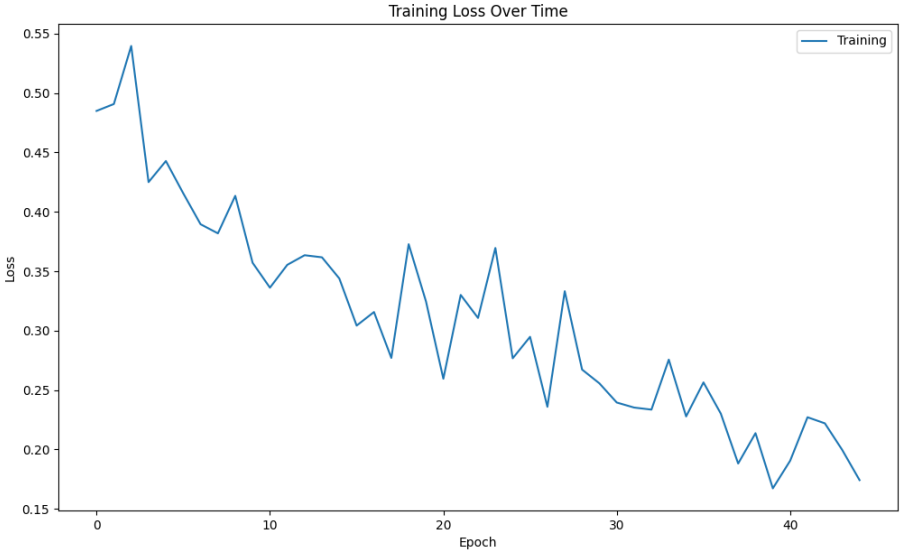


Training & Fine-Tuning Self-Supervised Model

- After many different combinations, reducing the embedding dimension from 128 to 64 helped the model follow a more clear decreasing pattern
- Still very spikey though – typical solutions (changes in batch size, learning rate, scheduler step, etc. did not help)



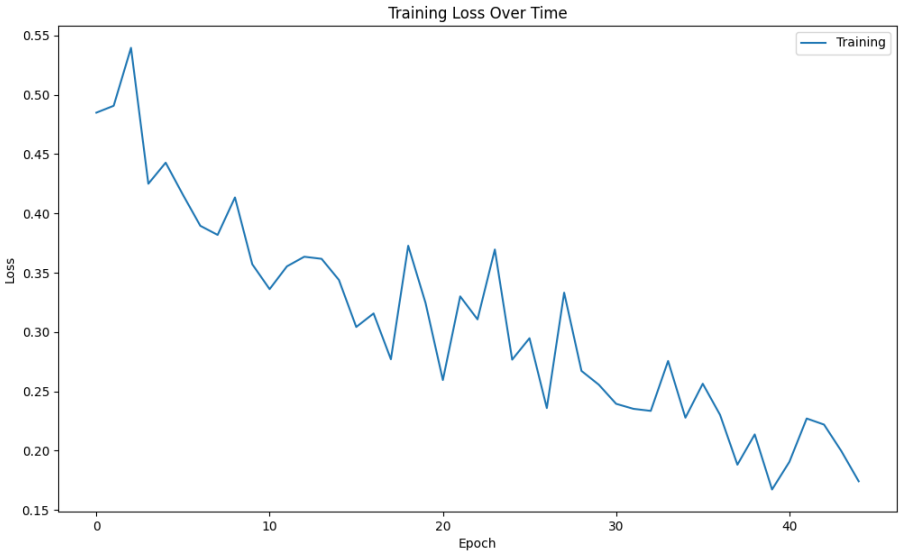
Reduced embedding dimension



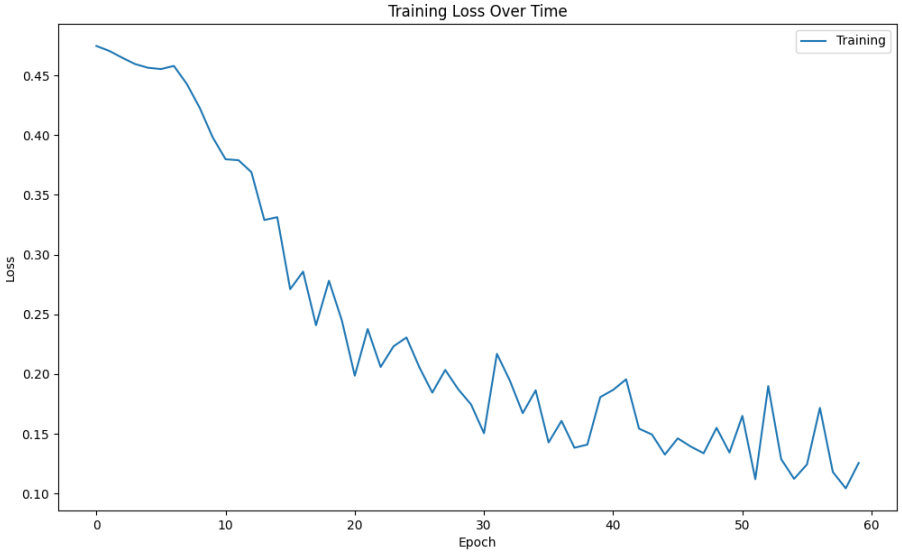
Training & Fine-Tuning Self-Supervised Model

- Using pre-trained weights as starting point for training seemed to help with getting a more traditional converging loss curve

Weights from Scratch

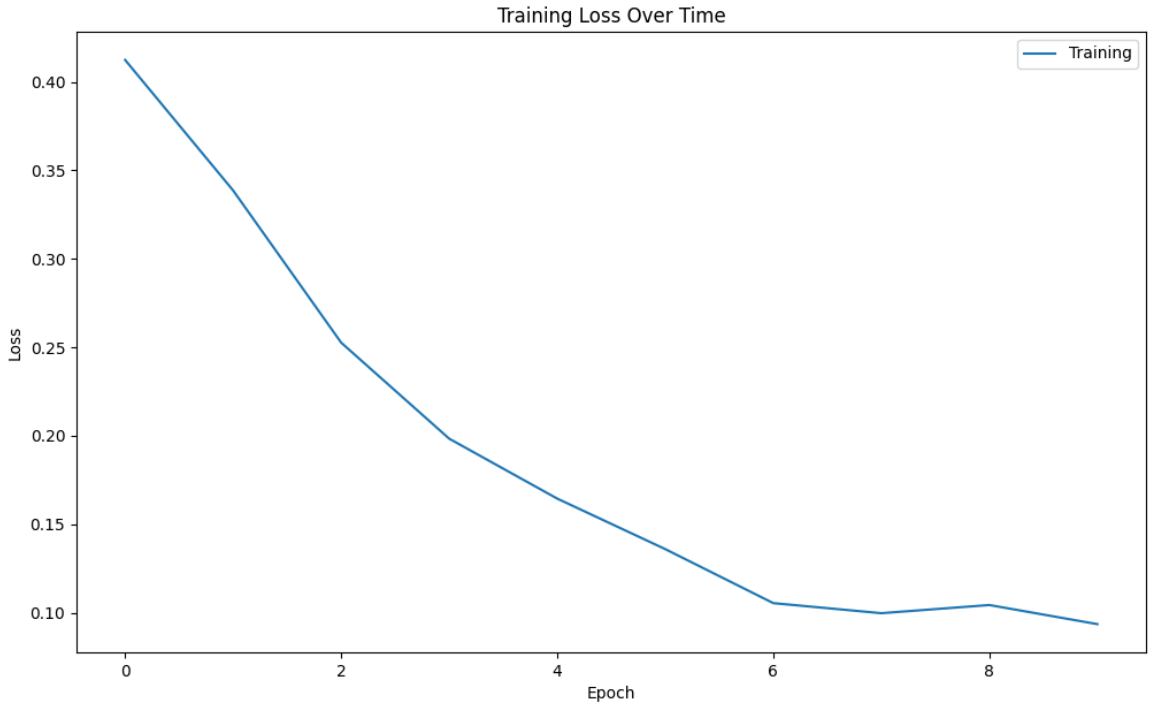


Pre-trained Weights



Training & Fine-Tuning Self-Supervised Model

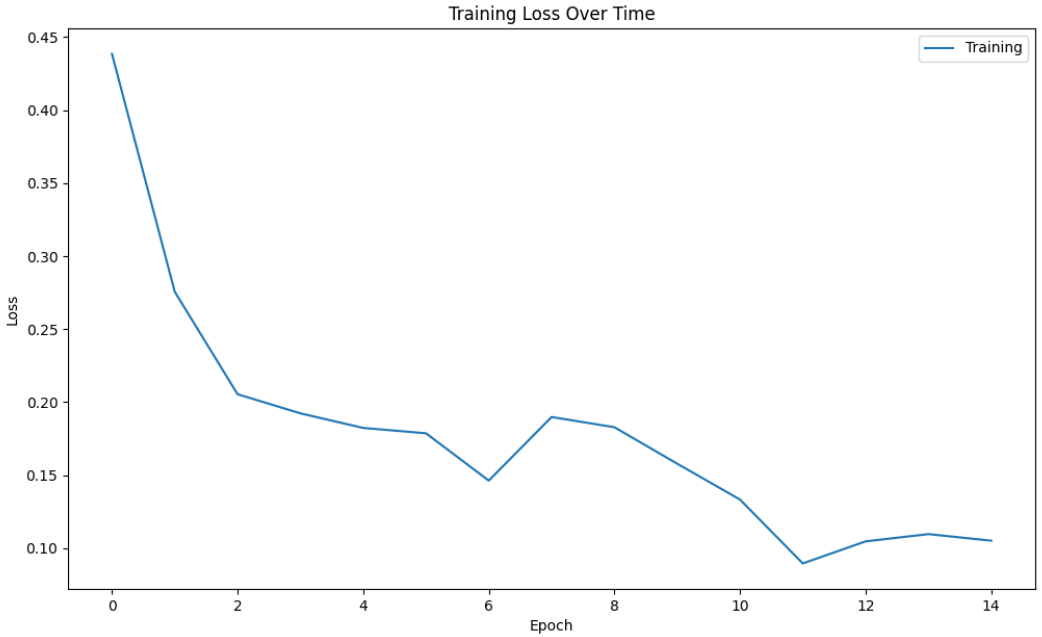
- Best results on pre-processed **bounding-box centered** images: 50% of data used for training (largest training set size I tried), embedding dimension of 128 (largest one tested), 10 epochs (to prevent overfitting), batch size of 128, learning rate of 0.0005



Bounding-Box Centered Images:

Training & Fine-Tuning Self-Supervised Model

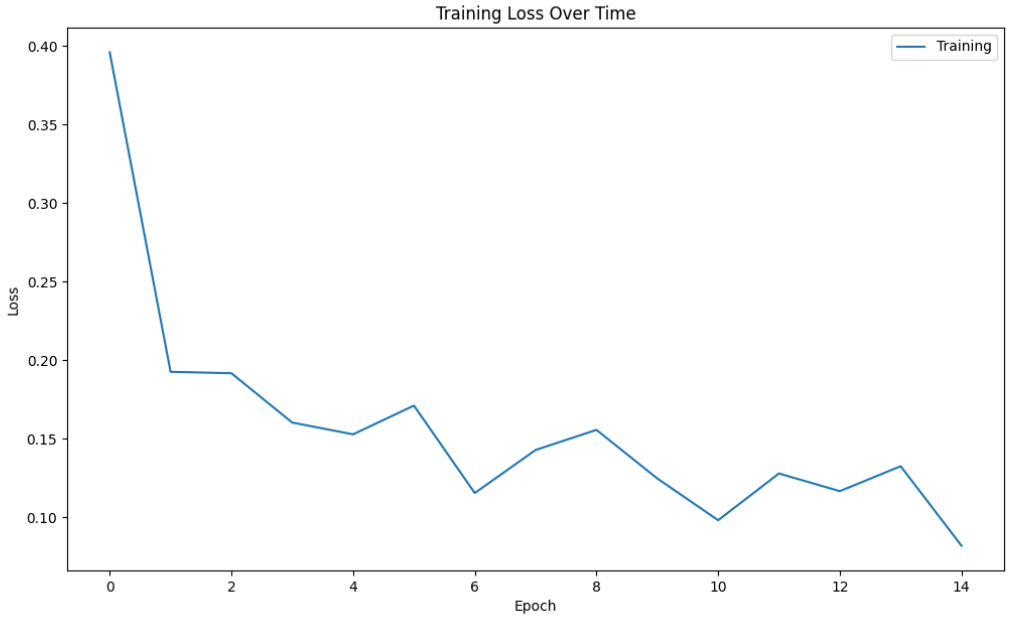
- Results on full, un-cropped RGB images: Loss graph does not have nearly as many spikes as the bounding-box centered image trials had. Note: This contains no centering or focus on the object itself, and thus will perform worse than object-centered image results when pose regression of the object is done using this model
- Ran on: 15 epochs, batch size 64, embedding dimension of 64, learning rate of 0.0005, 15% of dataset used



Full RGB Images:

Training & Fine-Tuning Self-Supervised Model

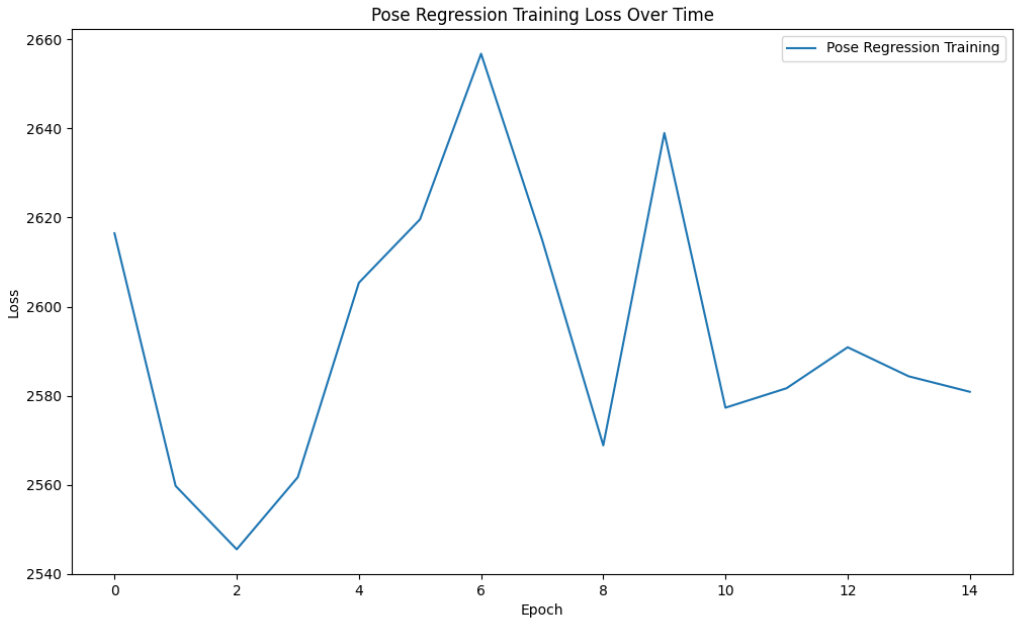
- Results on resized bounding box images: Results are spikier, but appear better and more converging than the bounding-box centered results
- Prediction: This will provide better pose estimation results when used in the full model, since it focuses on the object only and
- Ran on: 20 epochs, batch size 64, embedding dimension of 64, learning rate of 0.0005, 15% of dataset used



Resized 224 x 224 bounding box:

Training & Fine-Tuning Pose Estimation Model

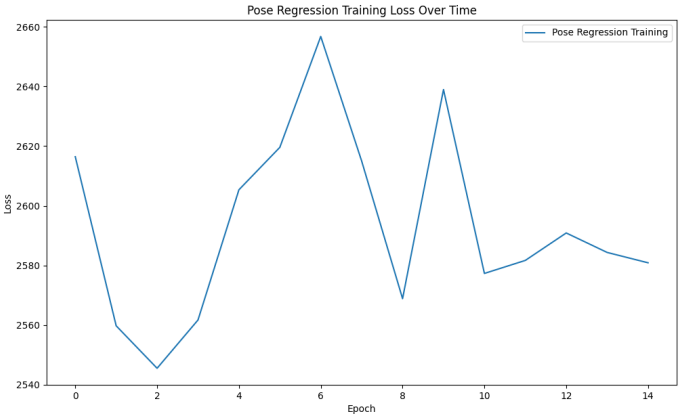
- Was originally planning on fixing the pose estimation training and testing code after my presentation so unfortunately I don't have successful results yet
- Code successfully executes and finishes but results (not finetuned) are currently bad
- Also not sure if my rotation loss calculation is correct, so that could be a big part of why the results are not good (incorrect updating)



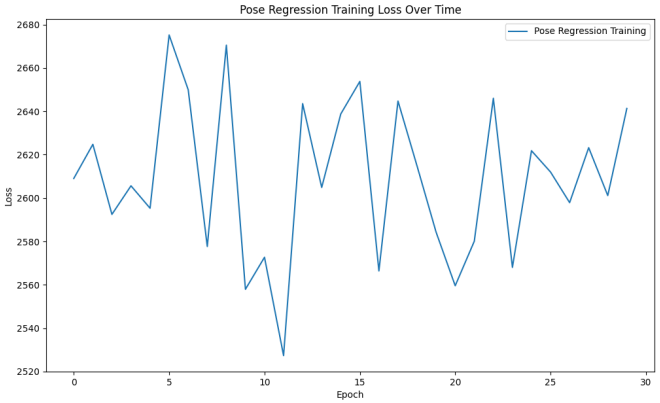
Training & Fine-Tuning Pose Estimation Model

- Only had time to test a few options for epoch hyperparameter — did not help
- Pretty sure it's cause the function calculations were wrong

15 Epochs



30 Epochs



70 Epochs

Future Work

- Fix my pose regression model training and testing code
  - Was able to make training execute and finish by presentation date as planned, but the functions I wrote for the loss, ADD metric, and rotation matrix to quaternion conversion don't seem to be mathematically or logically correct
    - In the future, I would take time to investigate how they work and sort out the logic myself
- Train with many more hyperparameter combinations to optimize my results
- Build upon the architecture over time:
  - Try a different CNN architecture backbone instead of ResNet18 (i.e. VGG, Resnet34, GoogLeNet)
  - Change the architecture of the pose regression layers and make them more complex
  - Do object detection instead of use the masks and cropped image
    - Fine-tune an existing model (i.e. YOLO-v2) for my problem
- Make the latent space distance between anchor images and their negative pairs proportionate to the difference in their image keys (i.e. if anchor image is image 0005 and negative pair is 1205, it would be that much farther away than if the negative pair was 0050)
- Much later in the future: Use self-supervision to estimate 3D bounding box coordinates and use them as keypoints and apply PnP algorithm or rigid transformation to model correspondences to get pose, instead of directly regressing the 6DoF pose — didn't have ground truth keypoints so wasn't able to attempt this

Thank you!

Queen's UNIVERSITY