\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MODEL 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

results

<https://github.com/WolffRuoff/Fingertip-Finder/blob/fef7342f4f9acc7d691d883e2f5df8f3cf4d3700/project.ipynb>

**My Initial Experiments in pixel-level segmentation for bounding box**

At first, I was concerned that directly training the model to classify all the pixels in the 480x640 image would be difficult, since the linear output layer would have 480x640 = 307200 nodes. I was especially concerned with the fact that when using resnet18 as base model, the latent space before the output layer has a dimension of only 512, which is sensible for the original ImageNet classification task with 1000 classes, but 307200 is a much higher dimension than 1000. Therefore, I have developed dataloader that would tile the images into smaller square sections, thereby reducing the output dimension to tile\_len^2, which in the case of 80x80 tiles is just 6400.

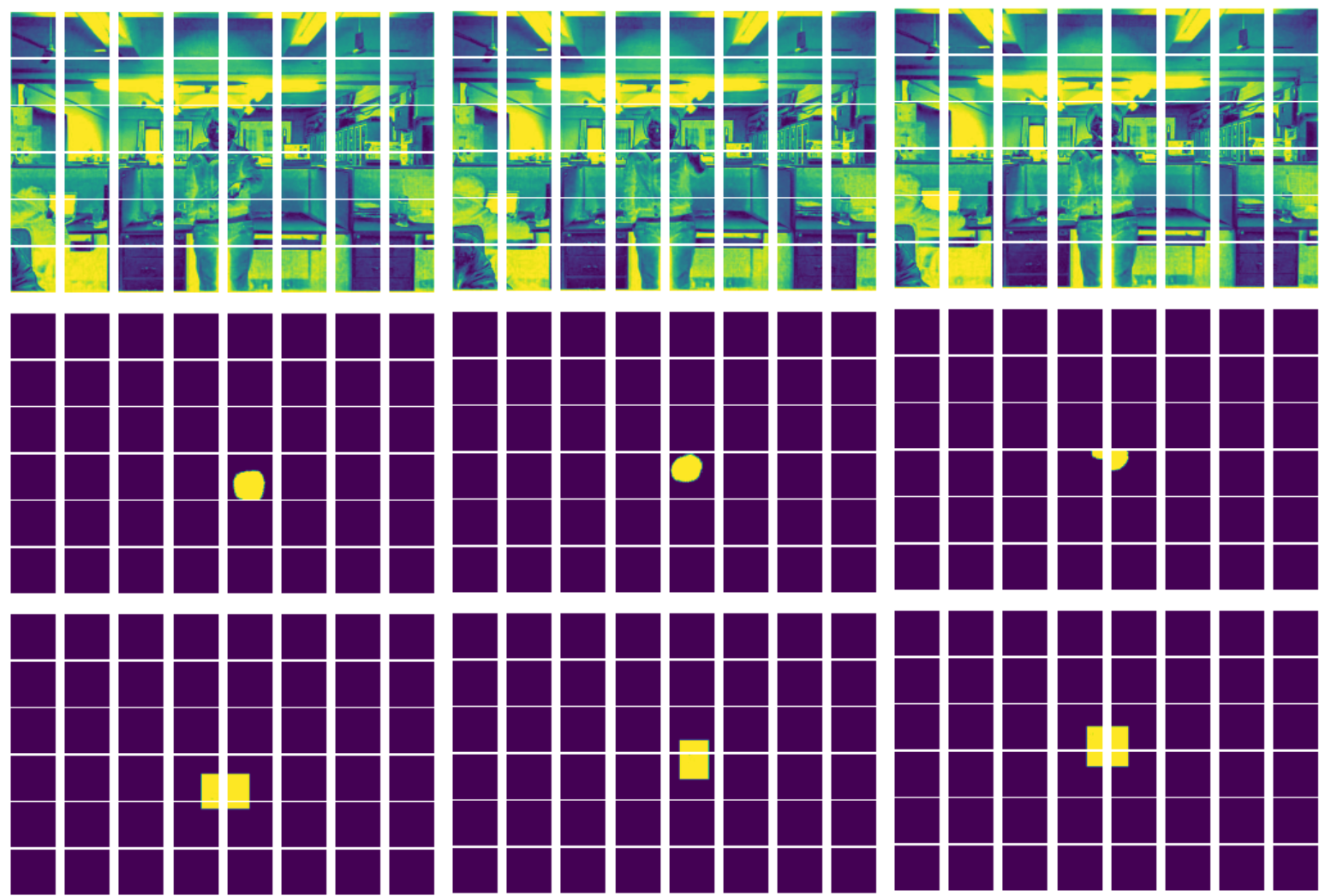
added transform that convert image to smaller tiles

to do pixel level classification or small tile level classification, it is necessary.

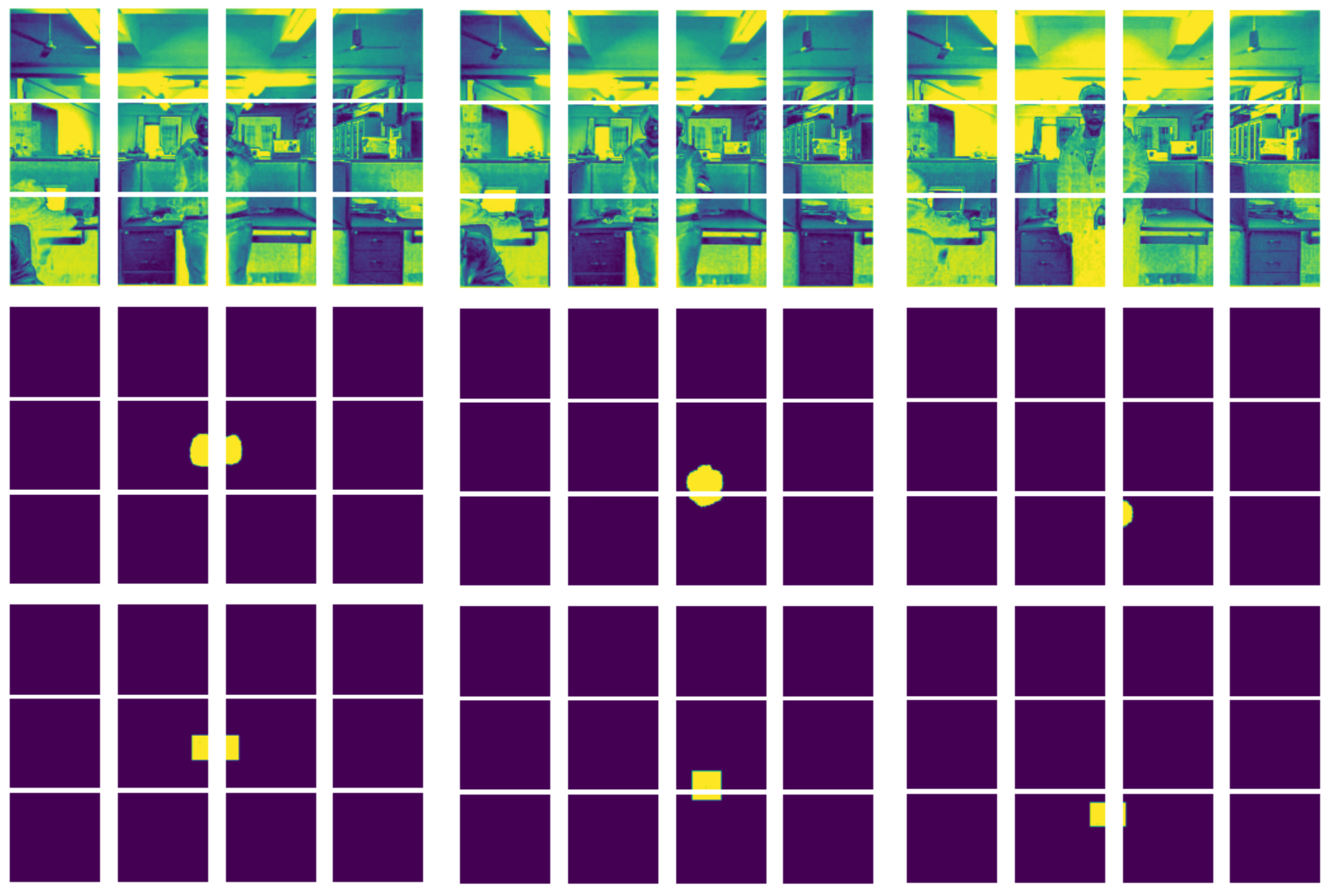
Tile transformation partition each input image and label mask pair to smaller tiles of uniform size, then reshape them to be pixel-level segmentation problem for each tile (effective batch size = number of tiles per img \* dataloader batch size), with a binary classification for each pixel (BCEwithlogitloss | sigmoid -- or equivalently dot(x,w)>0, followed by BCE loss)

It is also possible to do tile-level classification (where each 20x20 tile is assigned a single label indicating whether a hand is within that tile) with the tiling transformation. However, I have decided not to pursue tile-level classification due to poor initial results.

Initial experiments used small tiles (80x80) with pixel level segmentation x.shape = (batch\_size x num\_tiles\_per\_img, 3, 80, 80) y.shape = (batch\_size x num\_tiles\_per\_img, 6400) -- only 0th channel used for mask y, the other two discarded. Obtained moderate success with subset of 10000 examples, achieving roughly accurate pixel-level segmentation (but with frequent missing/off regions) val pixel level accuracy = 0.991287, val recall = 0.237542, val precision = 0.596420



Continuing using ImageNet-pretrained ResNet 18 as base model, I have increased tile sizes to 160x160, which led to substantial improvement: much higher precision and recall metrics: val acc = 0.996329, val recall = 0.827894, val precision = 0.797545

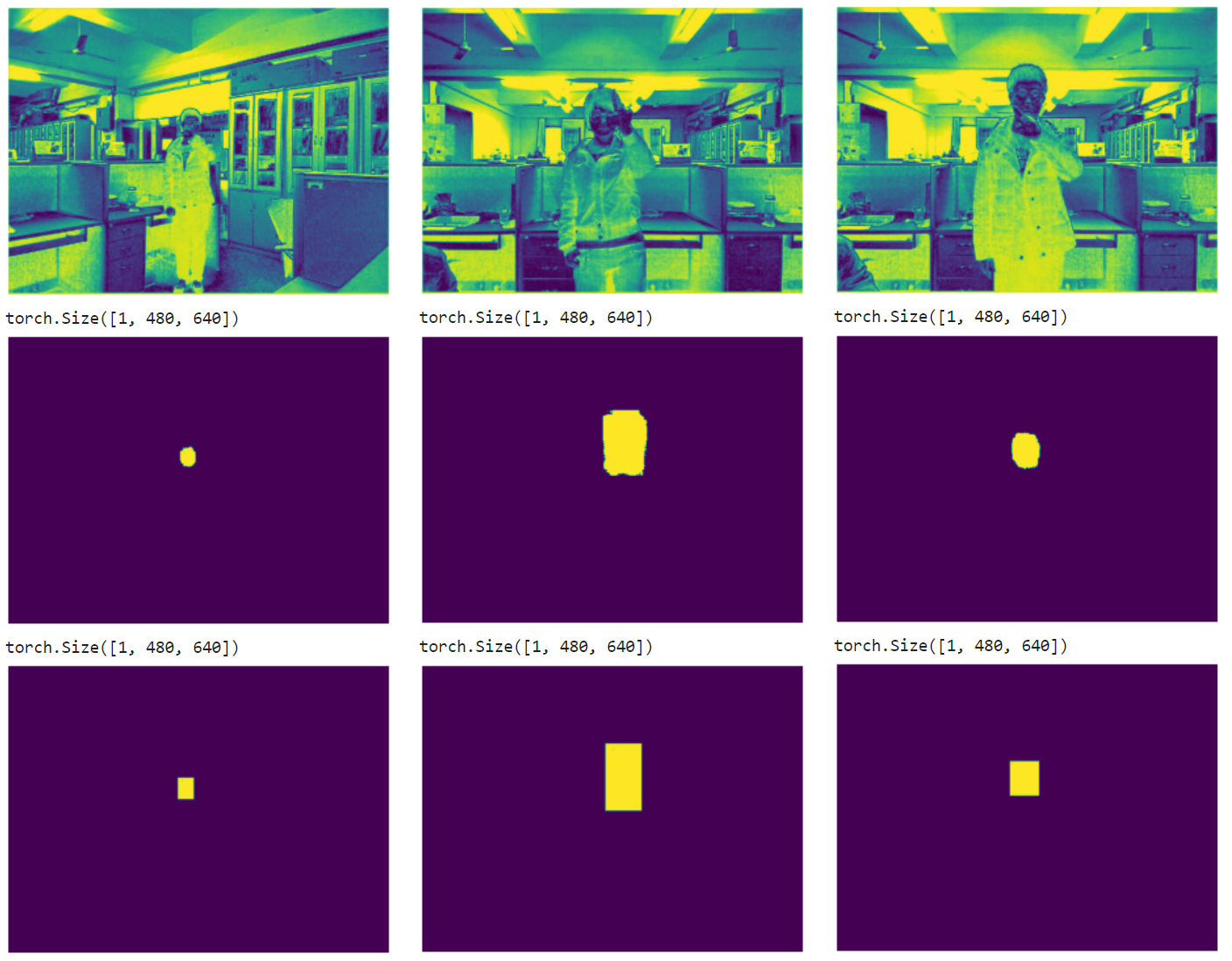


I have hypothesized that the poor performance using 80x80 tiles is like due to two reasons: insufficient context within the smaller tiles and the splitting of hand regions into multiple independently processed tiles, both issues are mitigated by using larger tiles (similar to a problem encountered by google in the streetview house number model, where they experienced issue with the detection module providing images that is cropped too close to the numbers with insufficient context/margins.

https://ai.googleblog.com/2017/05/updating-google-maps-with-deep-learning.html)

From the improvement obtained by increasing tile sizes, I have decided to further experiment with using the entire image instead of smaller tiles. Having a single tile and 480\*640=307,200 output nodes led to even better model performance over using tiling with 160x160 tile size.

val acc = 0.996653, val recall = 0.858869, val precision = 0.809522



Albeit the improvement from increasing tile size comes with the cost of substantially larger model with more parameters (state dict was over 600MB): nearly all of the model parameters are in the very large final dense layer, mainly due to the fact that convnet based models use shared parameters while the dense classification layer does not -- using tiling as experimented earlier can be also interpreted as parameter sharing for the final classification layer: a smaller model that learned how to classify pixels in a tile is applied on each tile parallelly during training and inference.

This improvement demonstrated that my prior assumption about the infeasibility of using 480\*640=307,200 number of output nodes for binary classification was simply incorrect.

Another observation from the improvement using no tiling and substantially larger classification layer is that the increase in performance from having more parameters might indicate the tiled models might be under-fitting. From the initial results, we have decided to use no tiling for the remainder of the project.

In the initial experiments above, the models were trained on a randomly sampled subset of the EgoFinger dataset to speed up the process of prototyping and iterative improvement. Later in the project we have leveraged the entire dataset and obtained improved validation performance in terms of accuracy, precision, and recall **(results here)**

Due to the class imbalance in the pixel-level segmentation task (>99% of pixels are class 0, that is, they are outside of the bounding box), the models typically begin to classify all pixels as class 0 in the first few epochs, then begin to classify some pixels as class 1 afterwards, gradually improving recall after the first 3-8 epochs, with the timing depending on preprocessing, choice of hyperparameters, and random initialization. The class imbalance is also the motivation for my implementation of precision and recall as additional evaluation metrics, since accuracy can be deceiving and opaque in such cases.

To help address the class imbalance, I have implemented weighted loss using built-in functionality of PyTorch’s loss functions. In the case of BCEWithLogitsLoss, I specified the pos\_weight argument to be a scalar greater than one, which is equivalent to assigning a higher cost to false negatives in the modified loss function. The addition of class weight, which added another tunable hyperparameter, affords greater control over the balancing between better recall and higher precision. Further experiment showed that while it is not essential, the use of weighted loss notably speeds up model convergence towards a good solution in terms of precision and recall.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MODEL 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Create hand masks from Kinect depth map and bbox label using adaptive thresholding. (which Ethan improved with convex hull detection).**

**Figure for img, mask, product**

**Figure for the process and output**

**Discuss caveat of long cpu-bound processing time due to thresholding parameter space search and how I mitigated it by saving processed hand-masks to disk during the process and directly load saved masks if found.**

**Pixel-level segmentation for the hand region (in contrast to the rectangular bounding box)**

**Pixel-level segmentation for the fingertip (after taking the hand bounding box and resize to 99x99)**

**Coordinate regression for the fingertip (in contrast to pixel-level segmentation)**

**This method is more similar to what was used in the paper**

In addition to our methodology, we also experimented with Huang et. al.’s coordinate regression method for fingertip detection after finding the hand bounding box.

The loss function used here is MSE loss on the x and y coordinates of the fingertip, which makes it a regression task rather than a pixel-level classification task, This training objective is equivalent to the Euclidean loss function used by Huang et. al.

Tried 1) whole image and 2) after taking hand bounding box and resize to 99x99

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PIPELINE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Putting together a pipeline that find bounding box and find fingertip coordinate**

**Use Ethan’s figure in the graphics folder**

**This part is done by Ethan, he combined our models from earlier and put them in a two-step process (similar to the paper) where the bounding box pixel-level segmentation model is used to find the bounding box, then the bounding box is extracted, resized, and fed to the cropped fingertip coordinate regression model, producing both the hand bounding box and the fingertip coordinate in the process. In addition to the preprocessing for the fingertip model (such as bounding box extraction from pixel-level mask and resizing the hand region), He wrote the visualization code for this section as well.**

**Use opencv to access webcam for real-time segmentation**

**Using opencv, I have implemented the code necessary to run our trained models using video feed inputs and produce real-time outputs. We found that our models are sufficiently fast during inference to handle real-time input and produce output at a decent frame rate. …**

**Re-train model with IPN dataset to perform better on webcams**

**(Ethan found and pre-processed the IPN dataset, which consists of webcam videos and frames with pixel-level segmentation labels)**

**Both of us have spent time training the model with this new dataset.**

References:

Y. Huang, X. Liu, L. Jin and X. Zhang, "DeepFinger: A Cascade Convolutional Neuron Network Approach to Finger Key Point Detection in Egocentric Vision with Mobile Camera," 2015 IEEE International Conference on Systems, Man, and Cybernetics, 2015, pp. 2944-2949, doi: 10.1109/SMC.2015.512.