FB Detectron2 - Facial Recognition Detection Performance *by* Training Data

Shaw Wen
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02120
wen.sha@husky.neu.edu

Abstract

This project paper is about the effect of training data on Detectron2's ability to pick out facial images from pictures or photographs. Here I compare the effect of different amounts of training data on the performance of the detection model from two perspectives: the performance metrics (accuracy, false negative, false positive) and the actual detection rate of faces from images. While the former suggests similar performance irrespective of the amount of training data, the latter demonstrates a classic example of the bias variance trade-off.

1 Introduction

Detectron2 is a Facebook Artificial Intelligence Research software system that implements state-of-the-art object detection algorithms. It is a successor or revision of its predecessor Detectron, which is implemented in TensorFlow. Like its predecessor, Detectron2 also includes Mask_RCNN as part of its benchmark; however, Detectron2 improves from the initial version in many areas including: implementation framework, more features, data customization, and most notably, significantly faster training. Unlike the initial Detectron system, Detectron2 is implemented in PyTorch, which is another deep learning framework that's comparable to TensorFlow but less buggy. The newer version also includes more features such as panoptic segmentation, densepose, Cascade R-CNN, and rotated bounding boxes. In addition, the flexibility of Detectron2 allows the system to be used as a library to support different projects. And this is the defining attribute which allowed me to carry out this particular project.

2 Experiment Setup

I have chosen to replicate the facial recognition project implemented by Venelin Valkov which entails training the system on a set of custom dataset, specifically, a dataset of facial images, to detect faces on images. The first step in the process is data preparation. For this particular project, we need images and annotations of the images as data input. Given the time consuming nature of annotation, a set of pre-annotated facial images provided by Dataturks hosted on Kaggle is used. To train on a custom dataset (facial images and annotations), first we register the new dataset with Detectron2's using DatasetCatalog by creating a dataset dictionary with annotations

in COCO dataset format, then we simply use the *DatatsetCatalog.register(...)* to add new data into Detectron2's data dictionary.

Next is the foundation of the novel part of this project, and that is to split the training dataset into four different subsets with varying amounts of training images and observe the effect on performance of the model's ability to detect faces from random images. The entire dataset is first split into 95% training and 5% testing. Then, we further split the training data into 25%, 50%, 75%, and 100% of the training data to create four separate training sets with varying amounts of data images. The objective is to train Detectron2 on each of the four training sets and compare the performance of the model's ability to detect facial images based on availability of training data.

Once datasets are properly created, we proceed to load a Mask R-CNN X101-FPN model that's pre-trained on the COCO dataset as a prior then train the model on our custom facial image dataset with the following hyperparameters: IMS_PER_BATCH = 4, BASE_LR = 0.001, WARMUP_ITERS = 1000, MAX_ITER = 1500, SOLVER.STEPS = (1000, 1500), SOLVER.GAMMA = 0.05. Depending on the amount of training data, the time used to train the model can vary anywhere from 20 minutes to slightly over an hour on Google Colaboratoy with 25GB of GPU. This phenomenon checks out with our intuition about the direct relationship between training time and the amount of training data.

3 Observation

3.1 Performance Metrics:

Figure 1: Mask_RCNN Accuracy, FN, FP using 25% Training Data

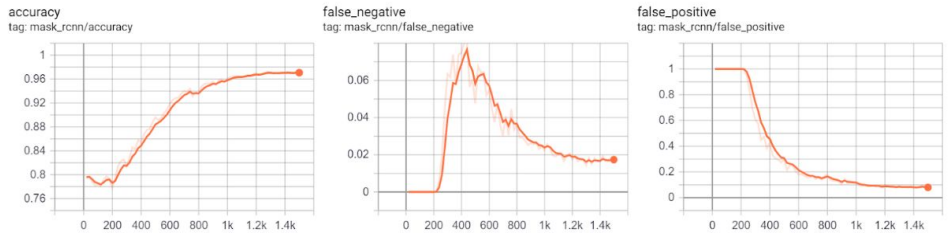


Figure 2: Mask_RCNN Accuracy, FN, FP using 50% Training Data

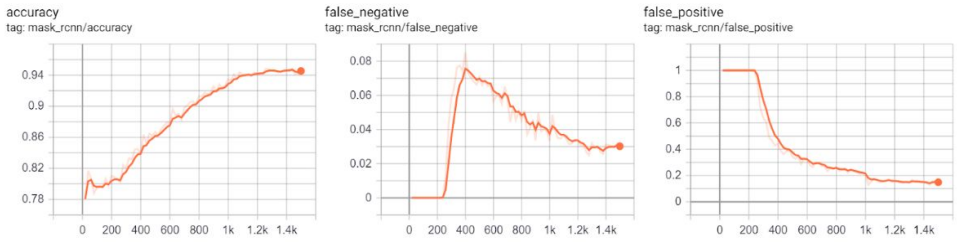
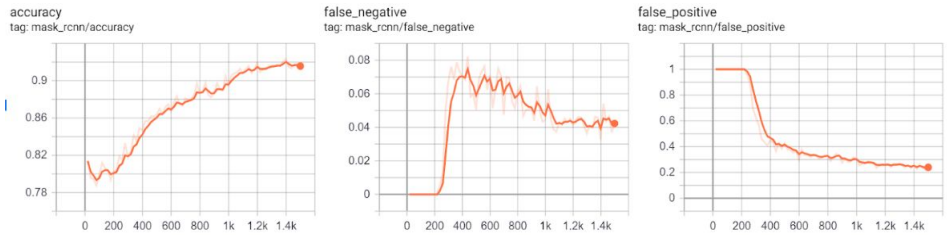


Figure 3: Mask_RCNN Accuracy, FN, FP using 100% Training Data



According to charts above, while the general shape of the curves are very similar, we observe distinctive patterns in the smoothness of the curves. It appears that the more data there is, the more jagged the lines become. One reason that I can think of is that a combination of stochastic gradient optimizer and increased data points creates more "jumps" on the charts. But as far as performances in terms of accuracy, false negatives, and false positives, more data did not necessarily lead to better results. In fact, according to the charts above, one might even conclude that more data leads to higher false negative and false positive rates. So if I were to make a verdict solely based on the charts above, I may be tempted to conclude that the amount of training data doesn't appear to make a material difference in performance. However, as I try to examine the model performance based on the actual detection rate of faces from images, I would observe a very interesting phenomenon.

3.2 Performance in Facial Image Detection:

Figure 4

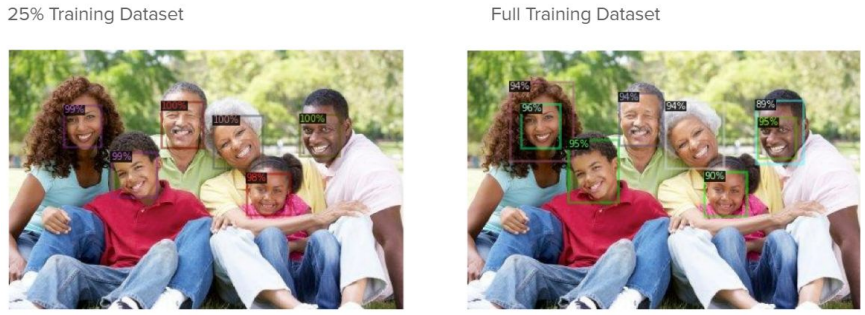


Figure 5

Figure 6

25% Training Dataset

Full Training Dataset

As shown below, while the model trained on less data picks out faces with higher score percentages, the model trained on more data generalizes better which translates to picking up more facial images that are in unusual positions, shapes or forms. This phenomenon is a classic example of the bias and variance tradeoff but with a twist. Wherein model evaluations, high bias is often associated with oversimplified models, or models with less predictors, high bias seems to correspond to more training data. In other words, the same neural network model tends to have a higher bias when trained on more data; whereas, training the same model on fewer data appears to create lower bias or higher variance therefore. Case in point, according to Figure 4, while the model trained on 25% of the training data detects facial images with 98% to 100% accuracy scores and the model trained on the full training set only yields 89% to 96% accuracy scores, the latter (model trained on full training set) appears to pick up a lot more details than that of the model trained on less data. And this phenomenon is further confirmed by Figure 5 and Figure 6 where the 25% model yields higher accuracy scores but the full training set model picks up more faces from the images.

4 Conclusion and Discussion

In conclusion, according to my observation from the above results, I learn that more data certainly has a positive impact on the performance of the Detectron2 model; however, since the impact is rather poorly reflected by the performance curves, it is important to examine the model performance by looking at how well the models can actually detect faces from images in practice. And what I have observed from the latter is that training on more data allows the model to capture more faces from images, particularly, those that are in unusual positions (profiles) or those with peculiar features (X-Men's Hank MoCoy). It'd be interesting to see how Detectron2 would fare in performance with different values for hyperparameters such as images per batch, learning rate, iterations, GAMMA and so on, but that's to be left for those who are interested in exploring more about the performances and optimization of FaceBook's Detectron2 project.

References:

- Project GitHub Repository - https://github.com/isosxwen/DL_Project
- Face Detection on Custom Dataset with Detectron2 and PyTorch using Python
  Fine-tune pre-trained object detection model on a custom face detection dataset
- Facial Detection in Images
- Detectron2 on GitHub
- mAP (mean Average Precision) for Object Detection