Capstone Project
On
Udacity AWS ML Engineer NanoDegree

Developing Deep Learning Model with U-NET[1] Neural Network on Carvana Image Masking Dataset[2] using Amazon Sagemaker

Report
By
Lu. Htin Aung
For
Udacity's AWS Machine Learning Engineer Nanodegree
On Capstone Project

[1] U-Net Neural Network - https://en.wikipedia.org/wiki/U-Net
[2] Carvana Image Masking Dataset from Kaggle - https://www.kaggle.com/c/carvana-image-masking-challenge/data

Submitted
By
Htin Aung Lu

Table of Contents

Submitted
By
Htin Aung Lu

Project's Domain Background

This project is based on carvana competition on the kaggle platform. The image segmentation model is performed on the given data without manual image cutting. This project is solely a part of computer vision field. The given data (photos) will be segmented by using a U-Net Network and given a mask to the desired area. This can be used in further implementation of checking the car's background and prices to more complex model like using to identify cars and people, in production of self-driving vehicles.

In digital image processing and computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as image objects). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.[3]

CNN became the most used in image processing models since 2010-2012, after that, in 2014, there came the R-CNN(Regions With CNNs) for image detection jobs with the use of bounding boxes. Just one year later, the fast R-CNN is develop with ROI(Region of Interest) Pooling and combining all models in single Net. In 2016, there is faster R-CNN with improvement in more accurate bounding box for regions. In 2017, the Mask R-CNN evolved as faster R-CNN for pixel level segmentation. With the improvement of R-CNN, U-Net NN was first introduced in 2015 to be used mostly in biochemical field and it keeps improving with R-CNN timeline.

Problem Statement

As with any big purchase, full information and transparency are key. While most everyone describes buying a used car as frustrating, it's just as annoying to sell one, especially online. Shoppers want to know everything about the car but they must rely on often blurry pictures and little information, keeping used car sales a largely inefficient, local industry.

An interesting part of their innovation is a custom rotating photo studio that automatically captures and processes 16 standard images of each vehicle in their inventory. While Carvana takes high quality photos, bright reflections and cars with similar colors as the background cause automation errors, which requires a skilled photo editor to change.[4]

In this project, we have developed an algorithm using the best practices of Sagemaker that automatically removes the photo studio background.

[3] Image Segmentation - http://en.wikipedia.org/wiki/Image_segmentation
[4] Carvana Image Masking challenge from Kaggle - https://www.kaggle.com/c/carvana-image-masking-challenge/

Datasets and Inputs

Dataset is get from the Kaggle challenge page, to reduce the storage usage, we will only use the non-HD version of the datasets. The dataset consists of a large number of car images (as .jpg files). Each car has exactly 16 images, each one taken at different angles. Each car has a unique id and images are named according to id_01.jpg, id_02.jpg ... id_16.jpg. In addition to the images, it also provides with some basic metadata about the car make, model, year, and trim.In the training set, a .gif file that contains the manually cutout mask for each image.

We will then upload these data into the S3 bucket to use effectively with Sagemaker. The images in the train dataset and train_mask dataset will be feed to the Model for training purpose, while the test dataset will be used for validation during the training.

As this is an image dataset and the Carvana Team already refined the image data, so we don't need much on cleansing the data. In the project, first we download the dataset using kaggle api and then we have split the data into 2 kinds, the Training Images and the Validation Images, with respective mask image data. After that we uploaded the images to the S3 bucket for easily accessible.

Solution Statement

We will use U-Net algorithm to extract the features on each image, eliminate the unwanted part and rebuild the image to get the masking image of the car or the expected output. We will use sagemaker notebook to implement the solution, first we will try hyper parameter tuning and search for the best hyperparameters to use with the algorithm, after that we will train the model using the debugger and profiler enabled to be able to log the situation and loss condition during the process. After that, deploy the model to a sagemaker endpoint and create lambda functions to start utilizing it on the image of the any type of car.

To define the success of the outcome, first we have our implemented mean dice coefficient to check the accuracy of the result. Moreover, we can see an output is good or not by our eyes also as shown below.

Fig. The Result Example[5]

[5] Photo taken from Carvana Kaggle Page - https://storage.googleapis.com/kaggle-competitions/kaggle/3333/media/carvana_graphics.png

We can see if the result is as good as the one shown in Manual editing box or not completely masked with model error.

Benchmark Model

The benchmark model is taken with permission from a github repo[6]. The model's output is in the output folder. I have also trained that way and record some of the metrics during the training run as follow.

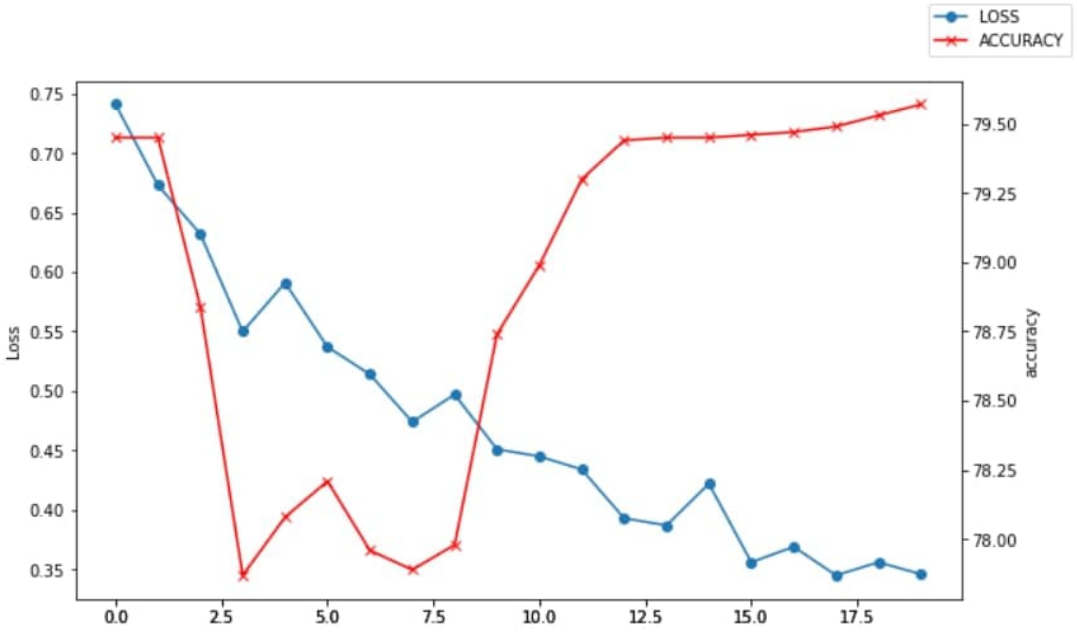

Fig. The Plot Showing the Benchmark Model's Accuracy and Loss

As we can see, the benchmark model is somewhat doing fine, but we might want to reduce the training time, search for the best hyperparameters (number of epochs, batchsize, the learning rate) to get more accurate and smoother model. I will improve the model and operationalizing it using sagemaker's best practices and AWS services.

[6] GitHub Repo of neirinzaralwin - https://github.com/neirinzaralwin/Deep-Learning/tree/main/Neural%20Networks/U-NET%20(Image%20Segmentation)

Evaluation Metrics

We will use the mean Dice coefficient. The Dice coefficient can be used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth. The formula is given by:

$$\frac{2 \times |\mathrm{X} \cap \mathrm{Y}|}{|X| + |Y|}$$

where X is the predicted set of pixels and Y is the ground truth. The Dice coefficient is defined to be 1 when both X and Y are empty. The Metrics is the mean of the Dice coefficients for each image in the test set.

Project Design

The Project design is straightforward, first we will create a sagemaker notebook instance with decent and fair performance. Download the dataset to the instance's environment and after that upload to S3 bucket to use while training. EDA is not needed in this case because dataset is clean and also include manifest files. After that by using sagemaker tuner, we will tune the model with best possible hyperparameters and when it is obtained, we will train the model using sagemaker estimator on a GPU enabled instance. We will then deploy the model to a sagemaker instance and create lambdas functions with appropriate permission to easily utilize the model.

Preprocessing

For the Preprocessing part, the dataset is given by Carvana Team on Kaggle as image data and we don't need much to handle the dataset. We just download the dataset using Kaggle api and then upload to S3 bucket.

A Sagemaker notebook is created with reasonable ml.t2.medium instance to use during the project. The kernel used for the project is sagemaker's conda_pytorch_latest_p36 because we are using the pytorch modules in many part of the process.

After that, we use the sagemaker sdk and import necessary modules in the Sagemaker Notebook.

Preparing the Training Code

For the preparation of the training code, I have taken references from a github repo as I mentioned earlier. The implementation of the algorithm is as follow. First, the input image is transformed as resizing it to 240p x 160p from original HD version of 1918x1280 to not over throttle the training environment and handled efficiently by the algorithm. We use common image transform methods like rotate, vertical flip, horizontal flip, etc. And normalize the image channels, then sent as the Tensor of pytorch using Albumentation's ToTensorV2 module which Convert image and mask to torch.Tensor. The numpy HWC image is converted to pytorch CHW tensor.

The model's pooling part is using kernel size 2 and stride 2 on the input image tensor. After that we define the Double Convolution as fully connected network on the pooled layers with convolution, batch normalization and ReLU applied twice as the down part of the model. In the Up Part of the model, we reversed the features and rebuild the images from the data with Convolution Transpose.

In the training, we import the dataloaders and model and create a forward training part and calculate the loss using dice coefficient. For the backward part we use the loss as feedback to the model and update the gradient scaler so that model can apply it and run the forward part again to get more accurate result.

Hyperparameter Tuning

I used Sagemaker's tuner to tune our model and search for the best available parameter. To use this I have to create a script for hyperparameter optimization code *hpo.py* that emits required metrics(in this case the loss which is to be minimum) to check for the efficiency. Then defined the hyperparameter ranges, in this case the batch size[7], number of epochs and learning rate. The batch size is categorical parameter, epoches is integer parameter and the learning rate is continuous parameter. When both the metrics definitions and hyperparameter ranges are set, we ran the tuner on the ml.g4dn.xlarge(because this instance provide the CUDA GPU) with 4 jobs to check for the best hyperparameter combinations.

[7] The Batch size larger than 128(included) can cause the CUDA GPU Memory to be insufficient. So Used only 32

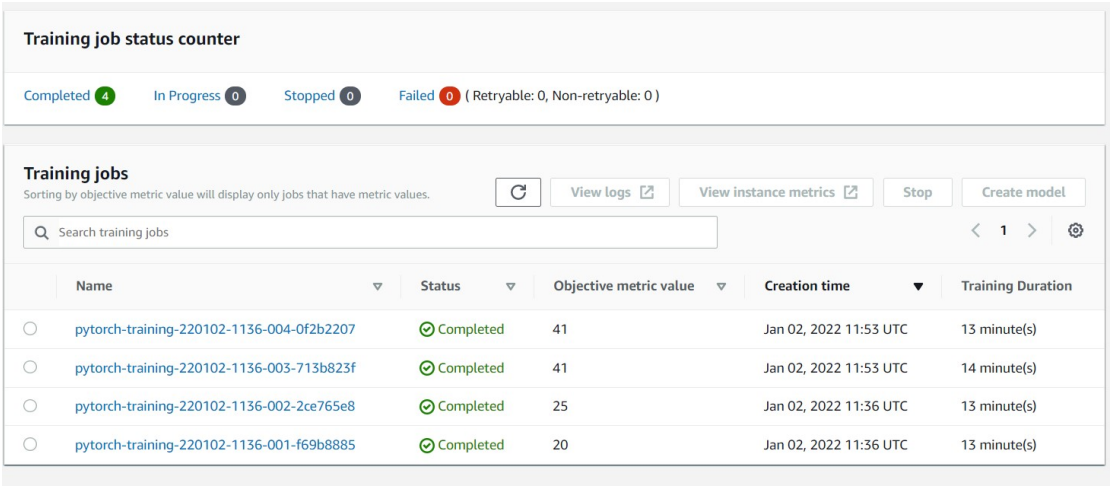

Fig. Hyperparameter tuning job with 4 training jobs

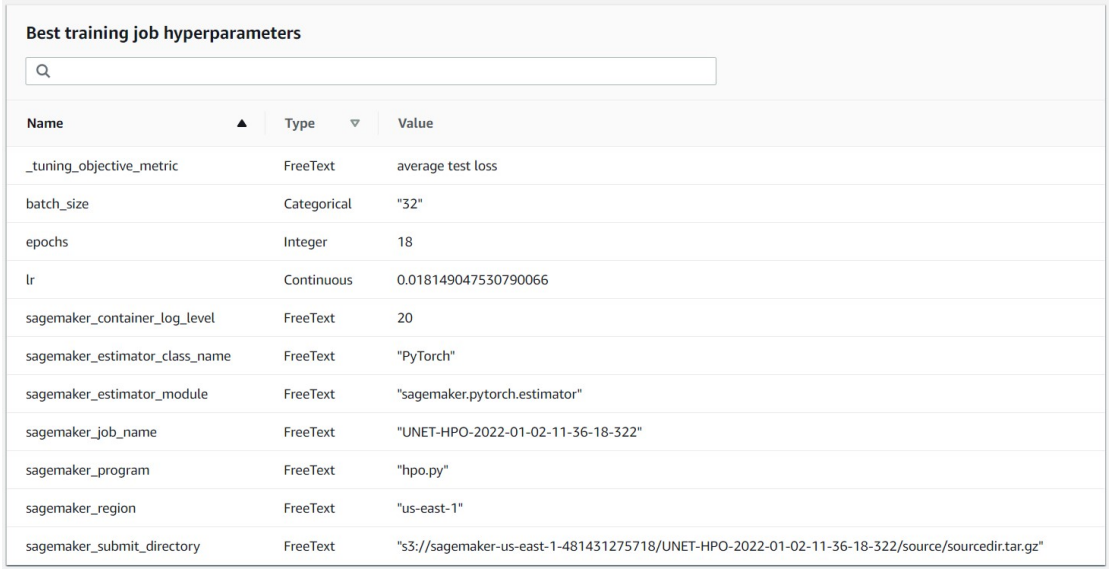

Fig. Best Training Job

Training on Sagemaker Container

After getting the best hyperparameter combination for the training job. We start the training job on a sagemaker container using the estimator function. To match the hyperparameter optimization job, ml.g4dn.xlarge is used again for this training with the *training.py* script.

To use sagemaker training we need to map the environment variables to be used as input channel, output channel, etc. in the sagemaker notebook. I use four channel as input for example, SM_CHANNEL_TRAIN, SM_CHANNEL_TRAIN_MASKS, etc. Which map to the S3 location of the images.

For the training we need to install the required packages in the training environment, container so I have included a requirement.txt in the script folder which we will map in the estimator module later as source_dir. Adding the sagemaker debugger hook to save the tensors we need helps us in plotting the loss output later. When the training is complete, I print out the tensors and see the loss. The loss of the validation seems to be a little over training. But in the cloud watch, I check for the metrics and the validation accuracy and loss are doing fine. So I use this model to create an endpoint.

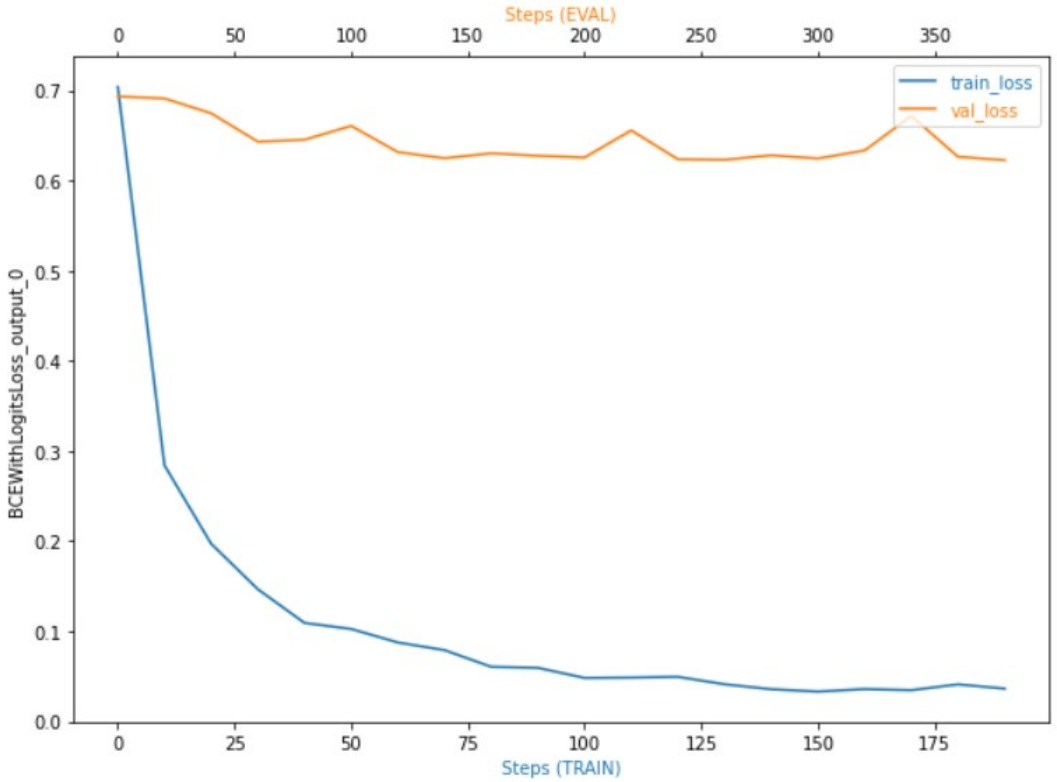

Fig. BCE Loss Plot

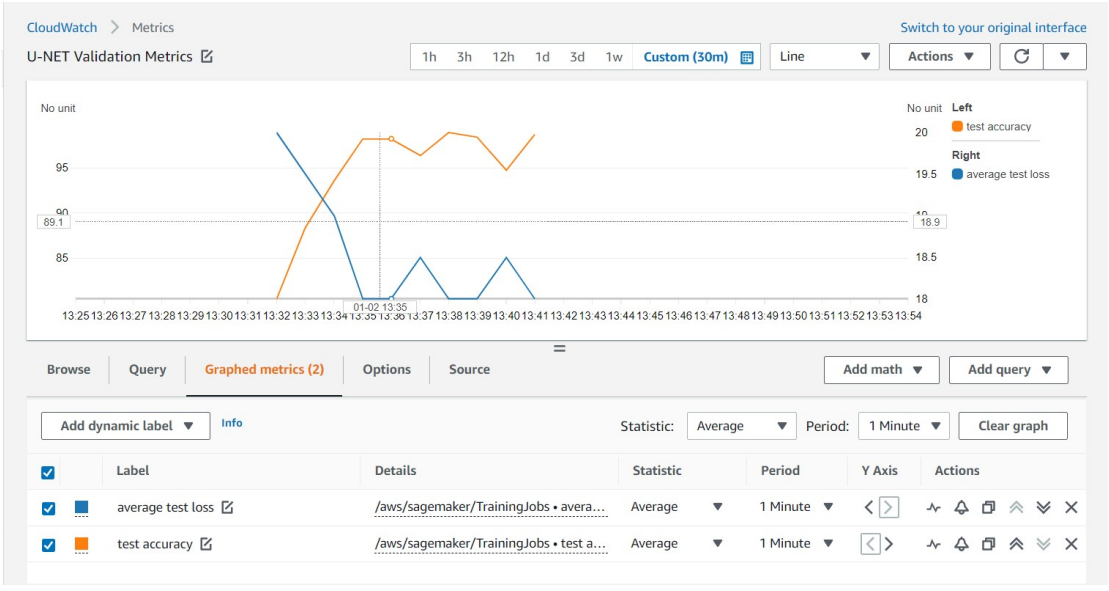

Fig. Validation Metrics in Cloudwatch

Deploying an Endpoint

Now the model is trained and it's time to deploy it to an endpoint. To do that we use sagemaker Endpoint to deploy our model with *endpoint.py* script.

Note: *There was an error with Sagemaker Endpoint and get internal 500 error.*

I have checked my script and script is not wrong so I tried to deploy it somewhere. To get the prediction results, I created another sagemaker instance on ml.g4dn.xlarge (to utilize the GPU) and run my script on this machine and able to get the prediction.

I have included both the first *endpoint.py* script and *endpoint_local.ipynb* notebook script in my repo, the result are pretty accurate and satisfying on *never-tried-or-validated* test data.

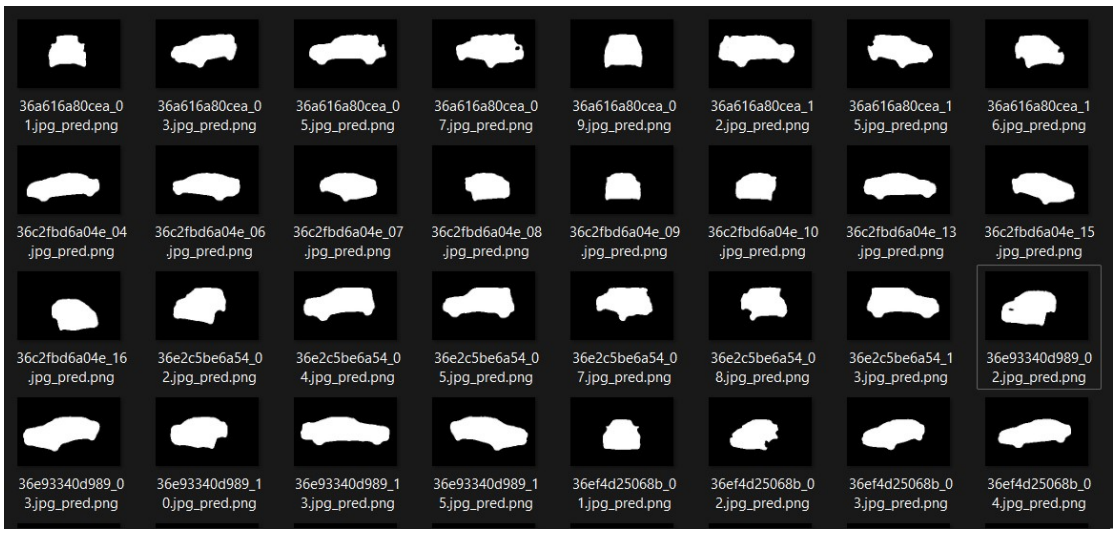

Fig. Inference result

Conclusion

This project is really interesting and fun. Learning the algorithm of U-Net architecture and implementing it into a real world problem makes me improve my understanding on the Machine Learning, and of course AWS's services.

The model may need a little refining as I think the training dataset is too large and got some over training issues. Currently the process from Data collection to the result is complete as it should be.

The implementation of image-image algorithm are so useful and practical in the real world problems like autonomous vehicles, person segmentation, and in cancer cell detection on medical field also. Thank you.

~EoF~

Submitted
By
Htin Aung Lu