Applying Cropping Strategies to Machine Learning and 3D Printing

Dillon Gyotoku

Summary of Research Questions and Results

1. Are there significant differences in image mean brightness and brightness variance across the dataset?
    a. The mean brightness ranged from around 160 to 190 on a scale from 0 to 255, while the mean variance ranged from 0.8 to 1.2 on the same scale. I found that there were two major peaks and one major valley in both mean brightness and variance, and the locations of these peaks and valleys lined up.
2. How does a model trained on random samples from anywhere in the image compare to a model trained on samples drawn from a gaussian distribution centered on a defect region? These models will be compared on mean accuracy and mean intersection-over-union.
    a. The model trained on random samples had a slightly higher accuracy of 98.32% whereas the gaussian model had an accuracy of 97.95%. However, the gaussian model had a higher mean intersection over union of 56.15% compared to the random model's 55.36%. I concluded that the gaussian model was the more accurate model for reasons explained in the results portion of this report.
3. Does a different network model create a more accurate model?
    a. To answer this question, I trained a model using the MobileNet framework and the gaussian cropping strategy. The previous models were trained using ResNet50. The MobileNet model had lower mean accuracy and intersection over union compared to both ResNet50 models, at 97.35% and 54.05% respectively. This made it the least accurate of the three models I trained.
4. How do these models, which use sub-images from the original, compare to a model trained on a resized version of the original image?
    a. The MACS model had the worst stats of the four models, with a mean accuracy of 94.16% and an intersection over union of 48.03%. However, this does not mean that I am able to definitively conclude that it is the weakest of the four models.

Motivation & Background

Laser powder bed fusion (LPBF) is a popular type of 3D printing that has several advantages over traditional methods and other 3D printing strategies. LPBF supports many of the materials commonly used in engineering settings and can easily create shapes that may be difficult for normal 3D printers, such as overhangs. It also keeps material waste and production costs low. However, it currently runs into issues with part quality, which keep it from being widely adopted.

The goal of this project and analysis is to detect defects during the process of the printer's operation using image processing and machine learning in order to improve the overall LPBF process. Reducing the amount of defects present in laser bed prints could allow these methods to become viable for many more products.

The current model being used by the University of Washington Mechatronics, Automation, and Control Systems (MACS) Lab is trained using images resized to 224x224. This method results in a loss of resolution, and in this project I aim to investigate whether keeping the original resolution by simply cropping a 224x224 area out of the original image leads to a more accurate model. When mentioned in this report, this resizing model is called the MACS model.

Dataset

I used two datasets for this project, which were collected with a camera mounted to the 3D printer that took pictures of the area of operation during a run. Both sets were provided by the MACS lab.

The first is available at this link: https://drive.google.com/file/d/1bHjsyqe2iGGjR9H4K0gyz9gON_Al4KHI/view?usp=sharing. The second is available here: https://drive.google.com/file/d/1ljRefs-4h4yR4WjzljrwdxkswlGLbOjT/view?usp=sharing.

Both files are stored as .zip files on Google Drive so that I can access them with Google Colab, the platform I used to do the training for this project. The first file contains 1800 greyscale images taken during one run of the printer, while the second contains 600 black and white "masks" for 600 of the 1800 images that essentially label the melt area for those 600 images.

Method

Question 1- Are there significant differences in image mean brightness and brightness variance across the dataset?

To answer this question, I started by unzipping the 1800 files from the first dataset with Google Colab and saving them to a new directory. Then, I used torchvision's to_tensor method to convert the images into tensors, where each value is a brightness value from 0 to 1 where 1 is the max. I then took the mean of this entire tensor and multiplied the value by 255 to get the mean brightness on a 0, 255 scale like a traditional RGB value. I can find the variance for each image in a similar way. I then used matplotlib to make simple line plots to show how these values changed across the dataset.

Question 2- How does a model trained on random samples from anywhere in the image compare to a model trained on samples drawn from a gaussian distribution centered on a defect region? These models will be compared on mean accuracy and mean intersection-over-union.

To answer this question, I implemented two methods of cropping. The first cropped out a random square image of the specified size from anywhere in the image. The second used a 2-D gaussian to find a location in the original image to choose as the center point for the cropped image, biased towards a given "target location" in the original. I then trained two models, each

using one of these strategies, for 6000 iterations using the 600 masks from the second dataset and their corresponding original images in the first dataset. I then implemented functions to calculate the mean accuracy and mean accuracy-over-union of each model. This was done with 10 samples from each image in the 112-image test set for a total of 1120 images. Lastly, I compared the results.

Question 3- Does a different network model create a more accurate model?

After training both models, I found that the model trained with gaussian cropping performed better. I then trained one additional model using MobileNet and the gaussian cropping model and used the same analysis functions mentioned previously to compare.

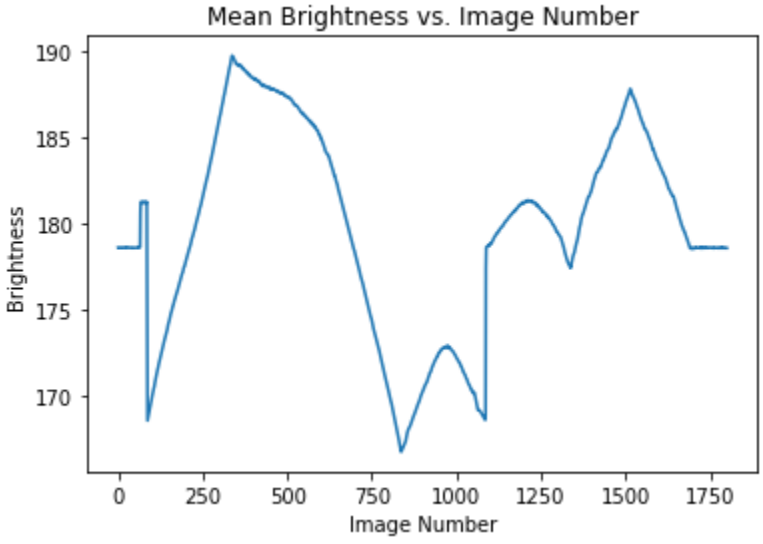
Question 4- How do these models, which use sub-images from the original, compare to a model trained on a resized version of the original image?

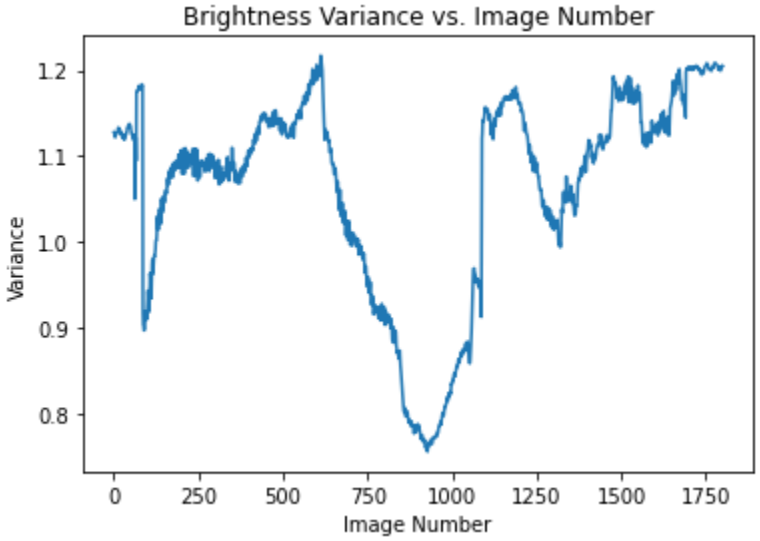
To answer this question, I spoke with my mentor at the MACS lab and asked him to run the dataset that I used through the existing model. I then compared the results to each of the models I created previously.

Results

1. Are there significant differences in image mean brightness and brightness variance across the dataset?

The images in the dataset were taken in chronological order during one run of the printer, so analyzing the mean brightness and variance was a relatively quick process. Once the images are converted into tensors using torchvision's to_tensor command, the pixel values are stored as the values in a tensor with the same dimensions as the original image (so the value at a certain x,y in the tensor is the brightness at that location in the image). I can then use the torch functions mean and var to find the mean and variance of the image tensors. The only other notable complication for this process is the fact that to_tensor converts the brightness to a number between 0 and 1, so I multiplied by 255 to get a number on the traditional RGB scale. I then plotted these values using matplotlib.





As shown, there are two clear peaks and one major valley for both mean and variance throughout the dataset. The mean ranged from around 160 to 190, while the variance ranged between 1.2 and 0.8. Finding the standard deviation of both the mean and variance indicated that both of these ranges qualify as statistically significant, with the mean standard deviation being 5.91 and the variance standard deviation being .119.

2. How does a model trained on random samples from anywhere in the image compare to a model trained on samples drawn from a gaussian distribution centered on a defect region? These models will be compared on mean accuracy and mean intersection-over-union.

Before explaining the results for this question, I will quickly explain some background about the training of these models, how they crop, and how they were evaluated.

The second dataset contains masks for 600 images in the first dataset, and these function as the labels of these models. This prevented me from using all 1800 of the images in the first dataset, as labels are required for any useful training to happen. The image is read in, then converted into a tensor, normalized, and cropped to a certain area according to the chosen strategy. The corresponding mask is cropped to the same area so that the pixels still line up. Then, I used torchvision transforms to add variety to the data through horizontal and vertical flipping, changing the brightness, and changing the contrast. The images are then split into a training and test subset, which are then again split into batches of 16. They are then ready for training.

The random model uses a strategy of simply picking any random location in the image as the center of the crop, then cropping out a 224x224 image based on that. The gaussian model uses a

defined center and covariance matrix to create a 2D gaussian based around a specific point, then samples a location in the image from that gaussian. These cropped images are the parts that are used by the model for the actual learning process during training, and each model is trained for 6000 iterations using the same hyperparameters.

After training is completed, the model is evaluated based on two statistics: mean accuracy and mean intersection over union (IOU). These are the two primary statistics used in two-class image segmentation. Mean accuracy is relatively simple, and is calculated as the number of correct pixels divided by the total number of pixels. Mean IOU, on the other hand, is a bit more complicated. For each class, the total number of correctly predicted pixels of that class is divided by the total number of pixels of that class minus the number of correct pixels. Then, you add up the result for all classes and divide by the number of classes.

As a short example, let's say the model predicts [0 0 1] for an image whose correct values are [1 1 1]. For 0, there are 0 correct pixels and a total of 2 pixels of that class, so we have 0 / (2 - 0) = 0. For 1, there is 1 correct pixel and a total of 4 pixels of that class, so we have 1 / (4 - 1) = ⅓. We then add these values together and divide by 2 since we have 2 classes, and this results in a mean IOU of ⅙. For readability, I have converted these values to percentages throughout this report.

When computing the evaluation statistics, I decided to take 10 random crops from each image for a total of 1120 cropped images. This was due to the small test set size of only 112 images, but also only possible because I was essentially using a subset of each image, which means that for the purposes of the model the other locations in the image that were not previously used for evaluation can be used as "new data".

The following table summarizes the results for each model:

	Random	Gaussian
Mean Accuracy	98.32%	97.95%
Mean IOU	55.36%	56.14%

Both models were able to achieve a high mean accuracy and a relatively good IOU. Normally values greater than 50% for IOU are considered good. Although the random model had a higher mean accuracy, I concluded that the gaussian model was more accurate overall because of its higher IOU. IOU is generally considered to be a more useful statistic than mean accuracy, as it takes into account incorrect evaluations. For example, if an image is 95% black and 5% red and a model classifies the entire thing as black, we have a 95% accuracy, but the model is doing nothing of value. The IOU for this hypothetical model would be only 47.5%, which is much

more accurate. Although the differences between the models are small, these differences are significant enough for the settings where this data would be used in practice.
Another thing worth noting is that due to the sampling methods used, both models have unavoidable inflation of their means whenever the cropped image does not contain the melt pool. These images will naturally have 100% accuracy and an IOU of 0.5, which brings up the average for both models. This is why I did not place much weight on the mean accuracy, especially for the random model which is more likely to sample an image that does not contain the melt pool.

3. Does a different network model create a more accurate model?

Both the random and gaussian networks I trained previously used a network framework called ResNet50. As the name suggests, this is a 50 layer deep residual network, and the pretrained model is available through PyTorch. It was created by NVIDIA and is a very common model in the computer vision world. MobileNetv3 is a similar type of classification network, but is notably less complex (meaning it has less parameters) as well as lower latency since it was created by Google for use with mobile phone CPUs, hence the name MobileNet.

MobileNet is also built into PyTorch, and so it was easy to change the network type over in the code. I used the Gaussian cropping strategy for this model as it was the better performing of the two strategies, and again trained for 6000 iterations using the same hyperparameters as previously.

The following table compares the MobileNet model to the previous two:

	Random	ResNet Gaussian	MobileNet Gaussian
Mean Accuracy	98.32%	97.95%	97.35%
Mean IOU	55.36%	56.14%	54.04%

As shown, the MobileNet model performed worse than both the ResNet models in both metrics. This is roughly as expected, as I knew going in that the MobileNet framework was less robust. It's possible that if trained for more iterations, the MobileNet could reach similar or better accuracy, but that was not possible since I wanted to keep the number of training iterations consistent.

4. How do these models, which use sub-images from the original, compare to a model trained on a resized version of the original image?

For the data on the resized model, I contacted my mentor at the MACS lab and asked him to run the model with the same training set. The following table shows all of the results:

	Random	ResNet Gaussian	MobileNet Gaussian	Resized MACS
Mean Accuracy	98.32%	97.95%	97.35%	94.16%
Mean IOU	55.36%	56.14%	54.04%	48.03%

Although the numbers for the resized MACS model appear to be the worst among the four, the results of this investigation unfortunately remain somewhat inconclusive for this particular question. If we are purely looking at these numbers, the cropping strategy is clearly the more accurate model- however, we must consider that the metrics for the cropping models are being inflated whenever they sample an image that does not contain any part of the melt pool. The MACS model always includes the melt pool because it does not cut out any part of the image and the melt pool is therefore always present. Whether this is a significant factor in causing its numbers to fall below the three cropping models is something that would require further investigation, and so I am unable to make a definitive conclusion for this question.

Impact & Limitations

An accurate way of detecting defects during LPBF would be a significant development for manufacturers across the world. LPBF processes could be expanded to more operations and become more consistent. As a result, it would become both cheaper and easier to create certain products that have historically been difficult and/or expensive to manufacture due to the required experience, manpower, or any number of other factors. Thus, the results of this investigation are useful to anyone who may be working with LPBF in the future.

Unfortunately, the results of this investigation must also be considered in context, and this slightly hampers their reliability. During the training process, it is very easy to choose a region of the original image that does not contain any parts of the melt pool whatsoever due to the random sampling. This results in an image that is rated as 100% correct and having a low IOU, but the model would never be used on an image that does not contain a melt pool and so this information is relatively useless. This can cause the model to appear more accurate than it actually is. As noted previously, using the gaussian method does make the model more accurate by having more images containing the melt pool, but it is still possible to get images that contain little to no melt pool. This problem is not present with the MACS model, which always uses the complete original image and thus always contains the melt pool in the image used for training, but this also causes it to appear to be less accurate than the models I trained. If there were more time, I would have liked to compare the two models using only images/crops that contain the melt pool.

Challenge Goals

The first challenge goal met by this project is Machine Learning. I believe I met this challenge goal because I learned about two main statistics used to compare image segmentation models, those being mean accuracy and mean intersection over union, and implemented functions to calculate these statistics for my models. I also researched and trained multiple models on different frameworks, in this case MobileNet and ResNet50. These topics were not covered in class and so I believe they qualify for the Machine Learning challenge goal.

The second challenge goal met by this project is use of a new library. Every single research question used PyTorch/torchvision functions to arrive at the answer, and the torchvision built in models were used for training. I feel that this qualifies as significant use of a new library and thus meets this challenge goal.

Work Plan Evaluation

- Setup
    - Est. 1 hr, Actual 1hr
    - This estimate turned out to be approximately correct. There were several issues to address and things to learn about getting all the data set up in Colab.
- Create visualizations of average brightness and variance across the dataset.
    - Est. 3 hrs, Actual 2 hrs.
    - I underestimated the time here since plotting this turned out to be relatively easy. The difficult part was integrating the mean and variance calculations for each image into the data loader, which ended up becoming a recurring theme with this project. I had issues getting the tensors into the proper shape and format, but once I had it was easy to plot with matplotlib.
- Implement first cropping strategy
    - Est. 3 hrs, Actual 2 hrs
    - I slightly overestimated the time here. This portion was challenging because I had never worked with tensors of this particular form before, so I was constantly printing out tensors and their shape throughout this process. However, I knew that the actual cropping part would be relatively quick once I got the tensor into the right form, and this turned out to be correct.
- Train and evaluate segmentation model to segment the defect region from the image
    - Est. 10 hrs, Actual 6 hours.
    - Overestimated but not for the reasons I expected. Training turned out to be a relatively simple process that essentially ran in the background while I was doing other parts of the project. However, I had a lot of problems integrating the cropping strategy with the data loader (essentially making it so that the cropping was no longer an external function but instead built into the dataset and called

when you wanted to get its information). I also had some issues getting the correct formulas for the evaluation and handling images that did not contain the melt pool. I also wrote a function during this part of the project that applies a train/test split based on a random seed.

- Implement second cropping strategy
    - Est. 3 hrs, Actual 3 hrs.
    - Turned out to be approximately correct. Gaussian cropping was much more complicated since I had to determine an appropriate covariance matrix to control how much deviation would appear in the sample, select an appropriate mean x and y, and actually sample a coordinate from the resulting 2D gaussian. Integration with the dataloader also proved challenging and I had to handle edge cases where the chosen point would result in a crop that is outside the bounds of the image.
- Train second model
    - Est. 2 hrs, Actual 2 hrs
    - After I had everything implemented, training took approximately the expected time.
- Compare to first model
    - Est. 3 hrs, Actual 1 hr
    - The statistics I chose to compare with are not good to visualize, so I instead opted for a summary table using a large sample size of 1120 images (10 from each image in the test batches). This was naturally much faster.
- Train third model using new network framework
    - Est. 3 hrs, Actual 3 hrs
    - It took me some time to find a network framework that was suitable for the problem and would be both easy to implement and compare to ResNet50. However, other than that, most of the time in this segment was taken up by training.
- Compare third model and MACS model to first two models
    - Est. 1 hr, Actual 2 hrs
    - During this comparison I found a few things that needed to be fixed in the evaluation code for all three of the models I had trained, so I had to go back and troubleshoot that to fix the issue. The MACS model data was provided by my mentor at the lab.
- Write report
    - Est. 6 hrs, Actual 7 hrs
    - The report ended up being longer than I initially predicted, and therefore took longer to write.

Testing

Since I did all of the coding for this project in Google Colab, I was able to easily test using new code cells and printing out the tensors, iteration numbers, etc. throughout the process. This allowed me to use smaller tensors here and there and log various things during the operation to see that everything was working properly. It also allowed me to do things like manually save the model during training in case I had to stop midway.

During the training process, I had two things that helped me see if the training was working. I had the ability to manually save the model during testing and check out the results as it was training on a graph using TensorBoard.

Collaboration

The dataset and some code was provided by my mentor at the MACS lab. These sections of code have been clearly marked in the relevant files. I also spoke with my mentor Jason several times about some technical issues I encountered in this project, as well as to get the evaluation metrics for the MACS model.

Additionally, I consulted the PyTorch/torchvision documentation as well as the occasional StackOverflow thread for information and troubleshooting.