Paper Reading and Analysis Assignment Report

Jonathan Alcineus

Many deep learning models can accurately classify melanoma with high levels of accuracy. The paper I selected, *"Deep Neural Network or Dermatologist?"* acknowledges this fact and explores what specific segments of nevi (benign skin lesions) or melanoma contribute to the model used to classify between them. The two main frameworks used by the authors are GradCAM (Gradient-Weighted Class Activation Mapping) and KernelSHAP (Shapley Additive exPlanations). Because many skin cancer prediction models are not open source, GradCAM and KernelSHAP serve to uncover more of the inner workings of these models through "weighted linear regression" (*Shap.KernelExplainer — SHAP Latest Documentation*, 2018) or "spatial information" (Jim Run, 2023). Specifically, the Kernel SHAP "uses a special weighted linear regression to compute the importance of each feature" (*Shap.KernelExplainer — SHAP Latest Documentation*, 2018), and GradCAM helps visually represent the areas of the image that the model is focusing on through a heat map. As the paper's deep learning framework is TensorFlow, with GradCAM and KernelSHAP being libraries built on top of it, TensorFlow will be the deep learning framework of choice. I will be using Inception, an image classification model pre-trained on ImageNet, to replicate the accuracy of classifying nevi or melanoma as found in the paper. After examining the limitations and future areas of exploration sections of the paper, I also aim to use a larger dataset, along with HAM 1000 and GradMask, to strengthen regularization and penalize reliance on the same few images with the same visual orientation.

One of the first things that I needed to do was to run the supporting code from the paper while using the correct version libraries that supported the code. For the first file that I had to run, train_and_test_set_creation.ipynb, everything ran as expected. The purpose of the first file I had to run was to organize the downloaded dataset into a specific dataframe, allowing me to run the rest of the tests and ensure that the remaining files in the program reproduced correctly. Rather than using the entire data set with all its attributes, the file, train_and_test_set_creation.ipynb, focuses on creating datasets that store an image as a numpy n-dimensional array and the diagnosis as a string. The paper focuses on the classification between nevi and melanoma, so it was also necessary to exclude any images that had a diagnosis other than nevi or melanoma. One of the things I noticed when examining the size and shape of the data for both the whole dataset and the testing model was that the training dataset has a size of 5617, and the testing model size is 400, resulting in an imbalance in the sizes of the nevi and melanoma classes. Specifically, the authors, Young et al., mentioned that "Our data set contains a total of 6017 images, with significant class imbalance: 5403 naevi and 614 malignant melanoma" (2019, p. 2) and that they had to "retain a balanced set of 200 images of each class as a hold out test set" (Young et al., 2019, p. 2). This class imbalance can lead to a bias in the model's predictions, as it may be more inclined to classify an image as a nevi due to the larger number of nevi images in the dataset. Although the models in the authors' implementation use InceptionV3 as the base model for fine-tuning the image data, there is still significantly less image data for the malignant melanoma class, which can make the model weaker and less accurate in classifying melanoma.

After creating the datasets to be used, when attempting to run the DataSplit_HpSearch.py file to train the models, the program encountered errors. Since the code provided with the paper was almost six years old, I had to update many of the libraries in the program to make it compatible and modify some of the methods as well. One of the first things I looked at was the specific Python version and the versions of the additional libraries that had been installed. After scanning the DermDNN.yml, I noticed that the Python version is 3.7.4. Because I wanted to update my implementation of the experiments in the paper and use a stable, updated version of Python, I chose to use Python 3.11. Python 3.11 also had a faster runtime, updated libraries with syntax that I understood, and optimized bytecode that better handles lists and sums, improving speed. This was crucial programming and training models in TensorFlow. When updating the Python version, many of the libraries I implemented in the program had deprecated methods that I need to review in their respective documentation. One of the deprecated methods, generic_utils from the Keras library, had to be replaced with Progbar, as I had issues running it. It was not compatible with the version of TensorFlow that I had installed, which was version 2.19.00. Another aspect that I wanted to update was the manner in which the images were sized and processed. In the original paper, the authors reshaped and normalized the images using the NumPy resize method, along with dividing all pixels by 255. Although I normalized each pixel for each training image the same way, I used the resize method from the cv2 Python library because it was more powerful for computer vision. There are more methods to make image transformation easier, rather than manually doing it matrix-wise with NumPy. Beyond what was written in the paper, I also wanted to test the compatibility of the GPUs that I would use for training the model and for performing the remaining experiments outlined in the paper. I utilized the TensorFlow library and the config method to view a list of physical devices and determine whether a GPU was connected to my Jupyter notebook. However, this step of checking for GPU devices came after many challenges of figuring out how to approach training my models. Updating the code to make it compatible with the current libraries was a significant part of the process, requiring careful consideration and testing to ensure the changes did not impact the model's performance.

To analyze non-open-source skin lesion classification models, creating a variety of different image classification models will be crucial. Specifically, creating the thirty models enables the authors to analyze precisely where, in the image, the model is making a decision between melanoma and nevi for classification. To strengthen the models, the creation of each model involves a hyperparameter search to determine the optimal parameters for the best model based on the training data. I followed the instructions in the README.md file, which involved running DataSplit_HpSearch.py. My first attempt at running the model was on Visual Studio Code locally on my laptop's CPU. After 10 iterations, my kernel crashed. For context, the original paper required 30 iterations of the hyperparameter search, 10 random starts, and the remaining iterations used the parameters from the previous one.

Additionally, the models were meant to be trained in parallel rather than sequentially. Through researching the issue, it was found that the kernel crashed due to excessive memory usage, as the trained models were stored both as files and in memory. As Python is mostly automatic with its memory allocation, I had to utilize different libraries to make training the model more memory-efficient. I first attempted to make the general code model memory-efficient by including the backend.clear_session() method from the keras library and the collect() method from the gc (garbage collector) library. It made only a small difference, allowing for five more iterations, after which the kernel crashed. Following this attempt, I tried reducing the batch size from 32 to 16. For the first 25 iterations, it was running smoothly until the kernel crashed. I even tried to lower the number of n_calls (iteration) and the number of n_starts. It seemed promising, and it completed all the iterations for the hyperparameter search; however, after completing the first model iteration, the kernel crashed after 12 hours of training. Through more research, I came across Lightning AI.

After many hours of configuring and testing various methods for training the models, I set the batch size to 16 to stabilize the training. I reduced the hyperparameter search from 30 n_calls to 15 and from 10 random starts to 5. Additionally, I utilized the Tesla T4 GPU with 32 GB of RAM to minimize the likelihood of running out of memory during training for each model. Finally, I trained each model on a different model and cleared the memory in between each model. In total, I spent over 15 hours, approximately one hour for each model, on training. Due to computational constraints, it was not possible to explore the entire 30 sets of hyperparameters for the hyperparameter search. The authors gained these results from creating a suite of 30 models. "The mean AUC over the 30 models is 85% with a variance of 1.8%" (Young et al., 2019, p. 3). Although the size of the model and the additions I made beyond the paper differed from those in the authors' implementation, I can confirm an average model accuracy of nearly 85% and a low variance of 1.8%, 0.012210668%, and 86.956522%, respectively. Lowering the number of models in the suite, running each model sequentially, and reducing the number of iterations for the hyperparameters may have impacted the average accuracy and variance in my implementation, skewing the average accuracy to be higher and the variance to be significantly lower.

The authors noted that "seventeen images were consistently mis-classified: at least 25/30 models got the class label wrong" (Young et al., 2019, p. 3) and that the variety in different misclassifications is because of different hyperparameters. Although I reduced the number of models and iterations for the hyperparameter search, I still found that the results obtained by the authors held in my experiment. To keep the model testing consistent with the model training, I decided to switch the encoding that the original test the authors constructed, with 0 being a nevus classification and 1 being a melanoma classification. The authors were initially able to receive their results for the experiment, but I wanted to correct what I believed was an error in the original authors' implementation. I implemented a script that displayed the number of misclassified nevi and melanoma for each model, as well as any common image IDs among all the models. There were no common images between all the nevi images misclassified as melanoma, but there was an image that was present throughout all the models' melanoma images misclassified as nevi, 57. To expand on the authors' findings that seventeen images were consistently misclassified, my implementation found that nine images out of the 400 test images were consistently misclassified, with 13 out of 15 modes misclassifying those images. One of the notable additional findings is that the average number of melanoma classified as nevi is significantly higher than the average for nevi classified as melanoma, at 29.3 and 20.2, respectively. However, there is significantly more variation, specifically standard deviation, for the number of melanoma classified as nevi compared to the number of nevi classified as melanoma, 9.22 and 6.82, respectively. As the analysis for GradCAM and KernelSHAP is primarily visual, I will focus on the first model, using images with indices 228, 231, 57, and 21. This is primarily due to the first two images being misclassified as melanoma and the latter two images being misclassified as nevi.

Examining the GradCAM and Kernel SHAP images for the most commonly misclassified image IDs reveals what the parameters for the model could not capture: specific pixels that the model focused on. Although the data I will be examining is qualitative rather than quantitative, the differences between the GradCAM saliency map and the kernel SHAP saliency map help to understand the limitations of the model architecture in identifying the features to focus on. When reading sections 2.3, GradCAM and Kernel SHAP, and 2.5 Spurious correlation, it was noticed that "the saliency maps show that at this resolution the majority of images do not unambiguously capture clinically meaningful information" (Young et al., 2019, p. 6). For the experiment I ran on the four most commonly misclassified images, I found the authors' notion to persist. Figure 3 shows the saliency maps of two different models: GradCAM and Kernel SHAP. With the first model correctly predicting melanoma with a confidence of 0.999, the heatmap produced by GradCAM focuses directly on the melanoma, with models centered in the middle of it. Similarly to Kernel SHAP, it produces a saliency map with green selections, where the color green represents areas of the image that positively contribute to the model prediction. The saliency map focuses on the entire melanoma section of the image, indicating the features of the melanoma that contributed to its correct prediction. Although both models have an accuracy of 85%, the second model misclassified the melanoma as a nevus with a confidence level of 0.996. The saliency map produced by GradCAM focuses the model on the upper left area of the image, rather than the center of the melanoma, leaving the melanoma outside of the heatmap. The saliency map produced by Kernel SHAP does select part of the melanoma that positively contributes to its prediction. However, the second model's saliency also captures green selections that are not the melanoma, such as discolorations of the rest of the skin. Additionally, there is a red selection, meaning that the selection negatively contributes to the prediction of that model, away from the melanoma. Overall, the second model focused on areas that were further away from the melanoma area in the images. Similar results from the second model in Figure 3 were present throughout the misclassified images in my experiments. The image with the most similar saliency maps to the second model in Figure 3 was the one with ID 23. Similar to the results of the second model, the GradCAM saliency map of the image with ID 23 focused away from the area of the melanoma. The Kernel SHAP saliency map of image 23 features green selections that highlight the melanoma. However, it also has red selection areas focused on areas that are not the lesion, even though they contribute negatively to the model prediction. Similar results were seen among the other images that were most commonly misclassified. To examine how randomizing a layer of a model impacts the GradCAM and Kernel SHAP saliency map, I will implement quantitative methods as well.

The Structural Similarity Index (SSIM) is "a method for measuring the similarity between two images" (Wang et al., 2003), specifically using local luminance and contrast to define a structure of the image (Wang et al., 2004). This is crucial because it provides a quantitative measure for comparing two images. This index is calculated using three separate comparisons: luminance, contrast, and structure. The luminance comparison involves using the average pixel density (0-255) of one image and comparing it to another image. The contrast comparison is calculated by subtracting each pixel density from the average pixel density, then calculating the standard deviation from all the pixels. For the structure comparison, the image is normalized, meaning it is divided by the image's standard deviation. With all of these, the SSIM multiplies each of the comparisons together, yielding a value between -1 and 1, where 1 indicates that the images have identical structure, i.e., they visually appear the same. The randomized layer experiment serves to examine how each layer contributes to the Kernel SHAP and GradCAM saliency maps, and how the randomization of weights in a layer impacts those maps as well. For the randomized layer experiment in the paper, a set of images and percentages of degradation across all layers were used for GradCAM and Kernel SHAP saliency images, utilizing SSIM. Instead of going through the entire test set, I use a sample of 10 images for calculating the SSIM scores. I opted to use the same layers that we mentioned in the paper, except for layer 'conv2d_65', which I replaced with the closest layer, 'conv2d_64'. While the majority of layers that were randomized had their saliency maps, KernelSHAP and GradCAM had them focus on areas near the skin lesion that is classified as melanoma, specifically layer "conv2d_51" had no heatmap for the GradCAM saliency map, even though the Kernel SHAP saliency map did focus on the center of the skin lesion. When examining the changes in the randomized layers quantitatively, we can observe how the SSIM scores degrade or exhibit differences throughout the layers.

SSIM scores degrade by 23, 4, 3, 2, and 4 percent for the GradCAM saliency maps. The SSIM scores for the Kernel SHAP degrade by 17, 3, 5, 3, and 7 percent (Young et al., 2019, p. 5). Rather than focusing on an image that was correctly classified, I chose images with the ID of 57, which was misclassified as a melanoma. One of the most significant findings in my results was that many of the degradations were not as varied as the degradations in the paper. I noticed a 9.6, 0.107, 2.83, and -1.69 percent degradation in the average Kernel SHAP SSIM scores. I also noticed a 64, -73, 24.3, 1.93 percent degradation for the GradCAM SSIM scores. While both my experiments and the paper's experiments start with the highest degradation and then gradually decrease, I observed significant degradation between the first and second images, which differs from the paper's results. The variability within the degradation was likely the result of a smaller dataset and sample images for the degradation.

Continuing to the use of the SSIM, it will also be a crucial part of detecting visual differences between models and the same iteration of the model. The authors conducted two additional sanity experiments to explore performance in terms of reproducibility and sensitivity. The authors ran two models that had the same image and the same model to determine similarity. While the author achieved results with an SSIM of 0.92, meaning that the images were similar to each other, my implementation of this experiment resulted in an SSIM of 1. This was a result of a small dataset for the run of two models. The authors also ran the same images for three different random models. The models that I selected randomly were models 8, 9, and 11. The authors reported an average variation of SSIM at 20% for the GradCAM and Kernel SHAP models, with all three models achieving an accuracy of 85%. (Young et al., 2019, p. 5) Through my experiments, all models achieved an accuracy of 87%. The GradCAM had an average variation of 24% across all models, while the KernelSHAP had an average variation of 6%.

Three of the crucial aspects that I would like to change in my implementation of the paper are. One aspect that I would like to amend is my understanding of the tools I was working with. I initially thought of training the machine learning model locally on my device. After the kernel crashed multiple times while training, I realized that I needed to use software that allowed me access to a GPU to run faster with sufficient memory until I saved each of the models. Another aspect I would amend is the selection and generation of data for training the models.

While the test dataset was equally balanced with 400 nevi images and 400 melanoma images, the training data was unbalanced with 5203 naevi and 414 malignant melanoma images. While the augment method from the scikit-image library helped in adding more melanoma images to the dataset, using the ImageDataGenerator from the TensorFlow preprocessing library would allow for additional augmentation beyond just exposure or rotating images horizontally or vertically. The final aspect I would change is trying to keep my experiments as close to the original paper as possible. Although I had used limited computational resources, experimenting with a larger GPU would be interesting to see how the variation in the predictions would change.

References

Jim Run. (2023, February 11). *GradCAM explained.* [Video]. YouTube. https://www.youtube.com/watch?v=eyKUqZOMfo0

shap.KernelExplainer — SHAP latest documentation. (2018). Retrieved September 5, 2025, from https://shap.readthedocs.io/en/latest/generated/shap.KernelExplainer.html

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. In *IEEE TRANSACTIONS ON IMAGE PROCESSING* (Vol. 13, Issue 4). https://www.cns.nyu.edu/pub/lcv/wang03-preprint.pdf#/

Wang, Z., Bovik, A., Sheikh, H., & Simocelli, E. (2003, February). *The SSIM Index for Image Quality Assessment*. Retrieved September 5, 2025, from https://www.cns.nyu.edu/~lcv/ssim/

Young, K., Booth, G., Simpson, B., Dutton, R., & Shrapnel, S. (2019). Deep neural network or dermatologist? In *Lecture notes in computer science* (pp. 48–55). https://doi.org/10.1007/978-3-030-33850-3_6