Toward Accessible, Lightweight, At Home Dermatological Screening

Nikhil Lyles
Stanford University
Department of Computer Science
nlyles@stanford.edu

Abstract

Skin conditions present a significant health burden globally with unmet levels of diagnostic and therapeutic need, particularly in tropical and resource-poor areas. While access to diagnosis by a medical professional and with medical-grade equipment (i.e. dermoscopy) is lacking, smartphone access globally has increased substantially within recent years. This motivates the development of a tool which could easily be accessed and used via smartphone and using smartphone camera imagery to aid in the screening and diagnostics of dermatological conditions, which I attempted in this project.

I experimented with variations on a simple CNN architecture to leverage their high performance and light weight (low parameter count) in my attempts to tackle this problem. While existing machine learning and deep learning models trained for skin condition classification also utilize CNNs, they are typically focused on one or very few skin conditions at a time, they are typically trained on higher-quality training imagery, and they are also often based on heavier, deeper pre-trained models with more complex architectures. For my models, I aimed to achieve diagnostic capabilities across many dermatological conditions simultaneously and opted to use lower-quality, camera-based (as opposed to dermoscopic) imagery with my goal of broader access to diagnostics in mind.

The models achieved around 25% classification accuracy, which leaves much to be desired and room for improvement, but the lighter architectural variations within the tested model architectures were largely able to keep up in performance with the heavier variations, and the experimental results elucidated valuable trends and identified key areas for further investigation.

1. Introduction

NOTE: This paper depicts and discusses ailments of skin, which may be visually uncomfortable or upsetting for sensitive viewers. Reader discretion is advised.

The World Health Organization (WHO) reports that 1.8 billion people globally are affected by skin conditions, with other estimates putting that figure at more than a third of the world population. Within the U.S. alone, as many as 84.5 million people (one in four Americans) are impacted, and skin conditions of varied causes are the leading cause of disease in tropical and resource-poor settings globally. Accordingly, it is clear that skin conditions can benefit from better diagnostics and subsequent treatment than they currently are. Improving treatment quality and/or access is a complicated issue far beyond the scope of this paper, but improving available diagnostic options is certainly within reach. With that in mind, this project sets out to develop an accurate, timely, and *accessible* diagnostic tool which works for a wide-range of dermatological conditions. With accessibility in mind, this model is trained on a dataset of relatively low resolution RGB imagery so that end-users could hopefully use this tool provided access to a smartphone with any (functional) level of camera quality.

The goal of such a tool is not necessarily to be a strictly diagnostic tool. For users (in areas) with access to proper medical care, it is intended to be more of an at-home tool patients can use to determine the nature of a dermatological issue they may be having and to accordingly assess the seriousness of the issue and what kind of treatment they may need to seek. For users (in areas) without as easy access to proper medical care, such a tool could also serve a quasi-diagnostic role for lack of accessible better alternatives.

In this study, I experimented with variations on a relatively simple CNN model architecture in an attempt to develop a lightweight model for many-class diagnostic prediction/classification on camera pictures of dermatological conditions. The models developed for and described in this report achieve test classification accuracy around 25%, so while they are not ready for true deployment, the different architectural and hyperparameter variations used in the experiments yielded valuable information and relationships and elucidated key questions and areas requiring further understanding.

2. Related Work

My paper is certainly not the first to attempt to develop a computer vision (CV) model or tool for dermatological images. A very recent survey paper on computer vision approaches for dermatological condition classification noted that CV approaches have proven beneficial through the techniques of image pre-processing (e.g. visually removing extraneous hair/obstructions from images), image segmentation, feature extraction, outright classification, and combinations of these four fundamental techniques [1]. While these are all valuable CV approaches in their own regards and in clever combination, my project will focus on feature identification/extraction and outright classification jointly, as it will use deep learning (DL) methods (i.e. CNNs) to "automatically extract relevant features from raw image data, eliminating the need for manual feature engineering" [1]. Although I noted my project will focus on feature extraction and classification jointly, the learned features are a means to an end for the sake of my model's classification ability and robustness (as opposed to the learned features being a focal result as well, wherein I could visualize or otherwise study the learned features to assess whether any of the significant learned features possess evident clinical significance or usefulness).

One thing that sets my project apart from other prior applications of CV to dermatological condition diagnosis is the intended breadth and robustness of my model's classification capabilities. Other CV applications, for example, have often focused on segmentation, prognostics (e.g. stage identification), or positive-negative classification for a *single (or few) condition(s) at a time*, such as melanoma diagnosis or psorias assessment [1]. Additionally, even when DL methods have been previously applied to multi-class diagnosis, they have been predominantly trained on the International Skin Imaging Collaboration (ISIC) challenge datasets or other assorted dermoscopic datasets, which is problematic for multiple reasons [2]. For one, the ISIC datasets have at most 7 classes for diagnostic classification (as of their latest/final 2020 iteration), and other dermatology datasets have that many at most if not typically even fewer classes. This necessarily limits (encodes less) diagnostic specificity into a given model's capabilities if trained on one of these datasets with less granularity of class labels (i.e. more specificity of dermatological conditions as class labels). Given that the Dermnet dataset which I used (described in further depth later) contains 23 class (dermatological condition) diagnostic labels for my model to learn and discern between, it represents a threefold increase in diagnostic granularity/specificity over other dermatological datasets and their respective models. Also, of the widely-known/-accepted publicly available dermatology image datasets, Dermnet is the only one which is based on a clinical image (normal digital photography) modality rather than the "dermoscopic" modality (which is similar to clinical imagery but is taken using specialized equipment which magnifies and illuminates features which aren't visible to the naked eye) [1]. With this in mind, my project's use of the Dermnet dataset uniquely positions it as trained on and focused on commonly available photography, making it more well-suited to my accessibility goal than other, prior DL models trained exclusively on dermoscopic datasets.

3. Data

The data I worked with for this project comes from a publicly available portion of the Dermnet Skin Disease Atlas (previously available through https://dermnet.com and now preserved for present/continued use on Kaggle). This dataset contains roughly 19,500 images, with an approximately 80-20 pre-provided train-test split leaving around 15,500 images for training and the remaining around 4,000 images for testing. I further chose to randomly split the pre-allocated testing images into half (about 2,000 images) each for validation and testing.

Training	Validation	Testing	Total
15,557	2,001	2,001	19,559

Each image in this Dermnet dataset is in the standard JPEG image format and is RGB (i.e. has 3 channels). Notably, the images have varied resolutions, so I did some (fairly minimal) preprocessing to standardize their resolutions before proceeding. Each image in this dataset contains exactly one label corresponding to one of the 23 (class) label options, which I will list below as follows:

1. acne and rosacea
2. actinic keratosis, basal cell carcinoma, and other malignant lesions
3. atopic dermatitis
4. bullous disease
5. cellulitis impetigo and other bacterial infections
6. eczema
7. exanthems and drug eruptions
8. alopecia and other hair loss diseases
9. herpes, HPV, and other STDs
10. vitiligo, light diseases, and other pigmentation disorders
11. lupus and other connective tissue disorders
12. melanoma (skin cancer) and nevi (moles)
13. nail fungus and diseases
14. poison ivy and other contact dermatitis
15. psoriasis and related diseases
16. scabies, Lyme disease, and other infestations and bites
17. seborrheic keratoses and other benign tumors

18. systemic disease-related symptoms
19. tinea (ringworm), candidiasis, and other parasitic/fungal infections
20. urticaria (hives)
21. vascular tumors
22. vasculitis
23. warts, molluscum, and other viral infections

I've provided an example of an image from the dataset in Figure 1, for reference.

Figure 1. An image from the Dermnet dataset used for this study depicting eczema (the class label/diagnosis) with excoriation (a clinical symptom/feature).

One noteworthy detail about this dataset is that the images all (as far as I could tell) contain a central "Dermnet.com" watermark, as our example image demonstrates. Although this slightly detracts from the quality of the dataset, I noted that the watermark is relatively unobtrusive and made the decision to proceed with using this dataset. If anything, I believed that my use of particular model architecture(s) (i.e. CNNs) would help computationally learn *features* as well as their relative importances in distinguishing between the identified classes for these images (in which case, it would more or less learn the watermark as a feature and realize its diagnostic insignificance/invariance). More over, while this watermark is a consistent form/amount of "noise" across the images in this dataset, my hope is also that this detractor from the image quality supports if not enhances my model's ability to perform diagnosis on relatively lower quality images (as is one of the central goals of this project).

4. Methods

The convolutional neural network (CNN) is a tried and true DL model architecture within the field of CV, largely due to its ability to pick up subtle, non-obvious image features within the data while respecting and preserving information about the (relative) locality of these features. It's no surprise, then, that CNNs have also been one of the most popular architectures within the application of DL to biomedical CV applications, notably to prior attempts at dermatological image DL classification models. As such, I opted to implement a CNN for this project due to the CNN's track record (both generally and in this application), its relative efficiency (e.g. lightweight-ness) compared to some other model architectures, and its feasibility of implementation.

First, as I alluded to in the Data section, I preprocessed the images of the dataset. Namely, I resized all images in the dataset to a standardized resolution of 256 x 256 pixels (hence the data for each image would be 3 x 256 x 256). Again, since the model being lightweight was a central intended feature (both for the sake of training feasibility and the longer-term goal of portable deployment of my model), I opted for a CNN, and one of only moderate depth at that. The first architecture I experimented with had three cycles of 2D convolution, ReLU activation, and 2 x 2 max pooling (in that order) followed by a flattening step and two subsequent fully-connected (linear) layers (each followed by a ReLU nonlinearity) and finally the classification head. I used an Adam optimizer and cross entropy loss (which is standard for multi-class classification). Due to the relatively small dataset ($\sim$19,500 images), I didn't think that a more complex model (e.g. a vision transformer) would be appropriate in this case, and I opted to stick with a lighter CNN. For the convolutions, the first Conv2D layer used 5x5 filters to try and capture slightly larger features from the images before the max pool downsampling had compressed them away, and I used standard 3x3 filters for the remaining 2 Conv2D layers (with the appropriate padding at each step to maintain original dimensionality aside from channels increasing before the max pool downsampling halved the dimensionality). The two dense linear layers had hidden dimensions of 256 and 64.

For the sake of continuity and simplicity throughout this study, I opted to stick with the original skeleton of my CNN architecture as a starting point for all of my proposed architectures and experiments. With that said, I did make some architectural changes (in addition to hyperparameter adjustments) during my experiments (which I will discuss in more detail in the following section). Although my model(s) may achieve results which leave something to be desired, I uphold and maintain the legitimacy of these architectural and experimental choices as my project is fundamentally still attempting something new that other developed models and investigations have not (namely, wider-scale simultaneous classification with an emphasis on lighter model weight).

Owing to memory constraints, relative familiarity/simplicity, and other factors, I opted to train my models locally rather than in a cloud instance. I implemented nearly all aspects of my pipeline (data loading and processing, model class implementations, training) in a Jupyter notebook, I used a Conda environment based on Python 3.13.4,

and I used a NVIDIA RTX 3070 using CUDA version 12.6. With this setup, it took between 3-4 minutes for the data loading and pre-processing (resizing) step (with some slight variance upon re-running this process), and it took just around 10 minutes for each model training run. Model evaluation typically took negligible time (i.e. only a few seconds). The exception to my local development choice was that I opted to program and execute my code for non-loss evaluation metrics (i.e. accuracy and confusion matrix) tabulation and plotting in a Jupyter notebook ran on a Google Colaboratory (Colab) CPU-based runtime instance. I made this decision after persistent, environment-breaking errors with my Conda environment(s) upon attempting to install the Matplotlib package (and/or other plotting libraries) despite multiple attempts to create a completely fresh Conda environment and carefully install only the strictly necessary packages. As such, an additional methodical consideration for my specific pipeline is that I saved (exported as files) the PyTorch tensors corresponding to the confusion matrices I computed on the test set for each experiment and then loaded in (from their respective files) these PyTorch tensors into the Colab Jupyter notebook for further processing and evaluation.

Lastly, I'd like to specify the evaluation metrics I used for this project. I mentioned above that I used cross entropy loss as a vital metric for training updates and evaluation. Additionally, I evaluated fully-trained (and seemingly viable) models using both accuracy and confusion matrices. These metrics together give a fairly good, comprehensive understanding of the model's performance on a micro- and macro-level.

5. Experiments and Results

For my first experiment, I trained my initial architecture with a learning rate of 0.001, a weight decay of 0.0001, 10 epochs of training, and a batch size of 100. Unfortunately, this first experiment had quite poor results. It was immediately evident upon looking at the training loss vs validation loss (seen in Figure 2 below) that something had gone awry:

While training loss steadily decreased across the course of the training epochs, validation loss barely decreased for one epoch and then steadily, egregiously increased across the remainder of the training duration. This is a hallmark sign of the model overfitting to the training data, which immediately proved to be an essential problem to mitigate during the subsequent experiments. Moreover, with this significant degree of (suspected) overfitting occurring and given my reasonably conservative hyperparameter choices for the first experiment, I heuristically concluded that this model architecture's performance would most likely not be salvaged/remedied by adjusting the hyperparameters. As such, I decided then to consider experimenting with the model architecture. First, let us consider the number of parameters

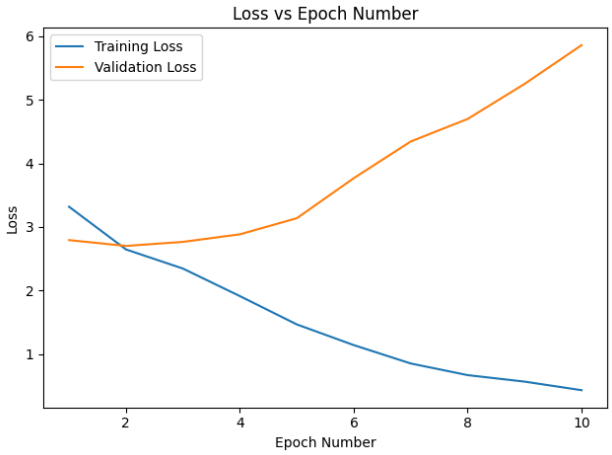

Figure 2. Training loss and validation loss plotted side-by-side as a function of epoch number during training for experiment 1.

(per layer) in our original model architecture below, in figure 3:

Layer	Params
Conv1	1,216
Conv2	4,640
Conv3	18,496
Linear1	16,777,472
Linear2	16,448
Output (Linear Head)	1,495
Total	16,819,767

Figure 3. Number of trainable parameters per unique layer of the original model architecture

Overfitting can occur for a number of reasons, but it seemed apparent to me that it was most likely occurring in this first experiment due to the model actually being *too complex* for its own good! Despite the relative simplicity of my original proposed model architecture, the best explanation for this training behavior in the first experiment was still that the model possessed too many trainable parameters (and possibly too many layers).

From Figure 3, there was one obvious layer to target for modification and/or elimination in subsequent experiments (given it contained the overwhelming majority of parameters): Linear1. Accordingly, I changed the Linear1 layer's hidden/output dimension from 256 to 64, which would overall reduce the model's number of parameters to nearly one fourth of first experiment's amount. Making this change and also increasing the weight decay from 0.0001 to 0.001 (1e-4 to 1e-3) while keeping the remaining architecture and hyperparameters the same yielded the loss graph in Figure 4 as a result.

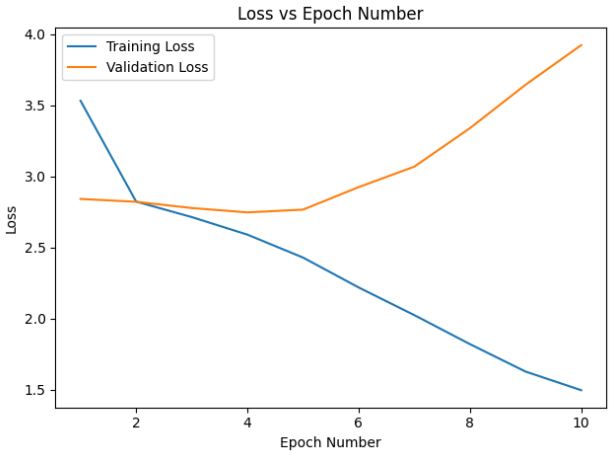

Figure 4. Training loss and validation loss plotted side-by-side as a function of epoch number during training for experiment 2.

While at first glance Figure 4 appears to display similar trajectories for training loss and validation loss as Figure 2 does, a slightly closer look reveals a promising result that this second experiment successfully took a step in the right direction: in the second experiment, although validation loss did eventually start increasing despite training loss consistently increasing, this turnaround started later in the training process than during the first experiment (from the fifth epoch onward rather than from the third epoch onward). This signaled that the second experiment's less complex model was likely less prone to overfitting in the same amount of time as the original model architecture. Additionally, the validation loss exceeded 4 by the seventh training epoch during the first experiment and nearly reached 6 (5.86), but the validation loss increased at a slower rate even after it started increasing during the second experiment and stayed below 4 the entire time (max/final validation loss of 3.92). These improvements and trends in the validation loss across the two experiments thus far were also reflected in the test loss values for these two experiments (i.e. the loss computed for our models evaluated on the previously "unseen" by the model test dataset), as the first experiment's final model at the end of training achieved a test loss of 5.77 while the second experiment's final model achieved a test loss of 3.98.

Looking at loss values and trends alone, it initially seemed that experiment 2 had improved the model insofar as it reduced overfitting. However, somewhat surprisingly, the second experiment attained a slightly lower test accuracy than the first experiment (21.4% as opposed to 26.3%) despite the second experiment having substantially lower test loss. This challenged my previous heuristic assumption that (test) loss more-or-less tracked with (test) accuracy, and I will discuss this observation and potential causes/implications in the following Conclusion section.

Nonetheless, I continued my experiments as planned, undeterred. For the sake of a control, I repeated experiment 2 but with the same original hyperparameters as experiment 1 (the only difference being that I restored the weight decay hyperparameter to 0.0001 (1e-4). This new experiment was experiment 3 in my study, and it achieved summary evaluation metrics between those of experiment 1 and experiment 2 on all accounts (loss trajectories for both training loss and validation loss were similar but between the two experiments), achieving a final test loss of 4.51 and a final test accuracy of 24.4%. To visualize the classification performance and issues of these first three experiments, I've included the confusion matrices for the final trained model evaluated on the test set for each of the first three experiments, in order, in Figure 5.

Based on a cursory examination of these confusion matrices, we see that experiments 1 and 3 have highly similar confusion matrices, but both experiments 2 and 3 make fewer true positive predictions for class 0 ("Acne and Rosacea") while experiment 2 makes substantially more (true positive) predictions for class 16 ("Seborrheic Keratoses and other Benign Tumors") than experiment 1 and experiment 3 makes substantially more (true positive) predictions for class 1 ("Actinic Keratosis, Basal Cell Carcinoma, and other Malignant Lesions") than experiment 1. This suggested that reducing the number of trainable parameters by modifying the model architecture (a fixed change from experiment 1 to experiments 2 and 3) most likely resulted in the model fundamentally shifting its distribution of classification predictions, as evidenced by experiments 2 and 3 skewing their predictions noticeably toward classes 16 and 1, respectively. Exactly *how* the classification prediction distribution shifts seems slightly unpredictable and inconsistent, given that experiments 2 and 3 shifted more noticeably to two distinct classes from one another despite the only difference being the weight decay hyperparameter choice.

One other interesting result was that a lower weight decay value (1e-4 in experiments 1 and 3 vs 1e-3) was associated with better test accuracy despite higher test loss. Accordingly, I opted to repeat experiments 1 and 3 *without weight decay* (0 for the weight decay hyperparameter) as experiments 4 and 5, respectively. While I did continue to keep an eye on and record the (training and validation) loss histories and test loss, I opted to mostly abandon them as my *primary* concern/evaluation metric for the remainder of my experiments, electing to focus instead on test accuracy and manual inspection/analysis of the confusion matrices. Experiment 4 showed a very slight decline in performance compared to experiment 1 (final test loss of 6.02 and final test accuracy of 24.4%), but experiment 5 showed the most promising results out of any of the models on the smaller

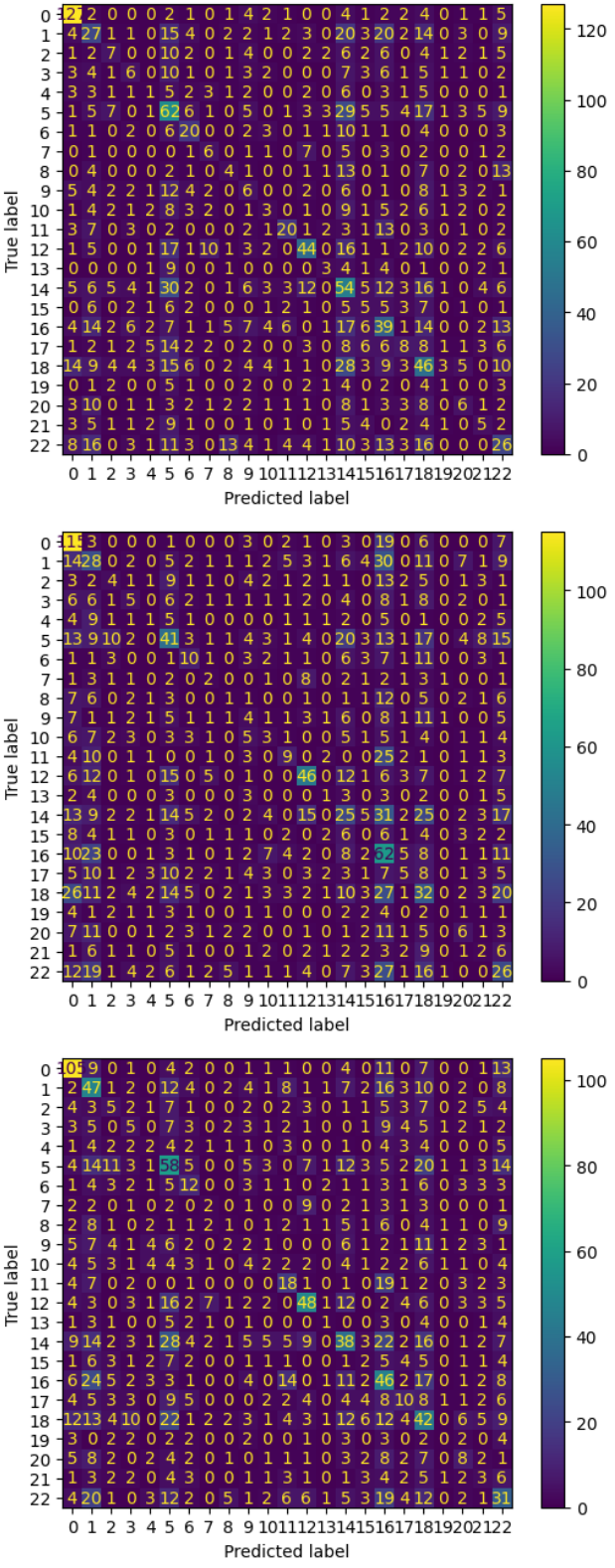

Figure 5. Confusion matrices for the first three experiments, in order. Each confusion matrix was obtained by evaluating the final epoch's trained model from the corresponding experiment on the pre-determined, unseen test dataset.

architecture used in experiments 2 and 3 (final test loss of 5.01 and final test accuracy of 25.6%). While the experiment 4 result was disappointing in comparison to the analogous experiment 1 (higher final test loss *and* lower final test accuracy), experiment 5 upheld and maintained the trend for the second, lighter architecture that reducing the weight decay hyperparameter value contributed to better final test accuracy despite increasing final test loss.

I also decided to further modify my model architecture as a form of regularization and experimentation. For example, I wanted to try adding a training-time dropout layer (and consider options for the dropout probability), and I also wanted to try removing the second linear layer to see what effect it would have. For all remaining experiments, I started with the already pruned model (used in experiments 2, 3, and 5) as the starting point for these model architectural modifications. For experiment 6, I removed the second linear layer from this model architecture. As for experiments 7 and 8, I left the second linear layer of the model architecture intact and added a dropout layer after the final output of the convolution phase of my model (for simplicity, after the flattening step), using the default dropout probability 0.5 for experiment 7 and a reduced dropout probability of 0.2 for experiment 8. Experiments 6, 7, and 8 all used the same hyperparameter configuration as experiment 5 (i.e. 0 weight decay and all other prior selections). Experiment 6 achieved a final test loss of 5.98 and a final test accuracy of 22.9%, which represented a decline in performance relative to experiment 5. As for experiments 7 and 8, experiment 7 achieved a final test loss of 3.52 and a final test accuracy of 24.6% while experiment 8 achieved a final test loss of 5.00 and a final test accuracy of 24.1%. It was very interesting for me to see that dropout didn't effect the final test accuracy that much, but it did have a substantial effect on final test loss. Additionally, removing the second linear layer had a significantly detrimental effect, so I opted to run two final experiments starting from the experiment 5 architecture/paradigm: one (experiment 9) with the hidden/output dimension of the second linear layer increased to 256 (as opposed to its current value of 64), and one (experiment 10) with that change plus the dropout layer used in experiment 7 (default dropout probability of 0.5). Since this second linear layer takes as input tensors of length 64, the "cost" (i.e. increase in number of parameters) is very light to increase the output/hidden dimension as opposed to the first linear layer. Experiment 9 achieved a final test loss of 5.26 and a final test accuracy of 24.7%, while experiment 10 achieved a final test loss of 3.69 and a final test accuracy of 24.8%.

Before I proceed to discuss and analyze my results further, I'd like to recapitulate what experiments I have performed. For the sake of space in this table and one more to follow summarizing the results of my experiments, I use the abbreviations WD for weight decay, EN for experiment

number, FTL for final test loss, FTA for final test accuracy, BTL for "best" test loss, and BTA for "best" test accuracy, where the final metrics come from each experiment's model at the end of training time while the "best" metrics come from evaluating each experiment's saved "best" model (based on lowest validation loss) on the test set. Before seeing the "best" model metrics for the experiments, I expected that they wouldn't necessarily be better than the final model metrics given that we had already seen little to no correlation between final test loss and final test accuracy already. I also refer to the architecture used in experiments 2, 3, and 5 as 1st pruned architecture. The experiments are summarized below:

EN	Description
1	Original Architecture & 1e-4 WD
2	1st Pruned Arch. (1PA) & 1e-3 WD
3	1PA & 1e-4 WD
4	Original Arch. & 0 WD
5	1PA & 0 WD
6	Exp. 5 w/o 2nd Linear Layer
7	Exp. 5 w/ 0.5 dropout
8	Exp. 5 w/ 0.2 dropout
9	Exp. 5 w/ augmented 2nd Linear Layer
10	Exp. 9 w/ 0.5 dropout

I summarize the results of my experiments in Figure 6 below, highlighting some of the more (in my opinion) notable results.

EN	FTL	FTA	BTL	BTA
1	5.77	26.3%	2.74	19.6%
2	3.98	21.4%	2.71	20.2%
3	4.51	24.4%	2.75	19.1%
4	6.02	24.4%	2.74	21.6%
5	5.01	25.6%	2.73	19.5%
6	5.98	22.9%	2.80	17.3%
7	3.52	24.6%	2.77	19.3%
8	5.00	24.1%	2.76	20.1%
9	5.26	24.7%	2.76	17.7%
10	3.69	24.8%	2.71	19.6%

Figure 6. Summary of evaluation metrics for the experiments ran in this study.

6. Conclusion

6.1. Discussion

One noteworthy takeaway from this study was that, in the case of this problem and the models developed for it at each experiment, test loss did not correlate with test accuracy in the expected manner (i.e. lower test loss corresponding to higher test accuracy and vice versa). One potential reason this can occur in general (and why I suspect it occurred in the course of this project) is the existence of class imbalances (i.e. when the number of examples/elements in the different classes is highly varied and some classes are not comparable/close enough in size). Similarly, the model for each experiment identified as "best" (based on lowest epoch validation loss during training) universally had lower test loss but *lower* test accuracy than their corresponding final models from the end of training. In fact, every single experiment's "best" model had lower test accuracy than the worst performing (by way of lowest test accuracy) of the experiments' final models. This observation challenges my earlier (heuristic) assumptions that validation/test loss increasing would strongly predict worsening test accuracy of the models in question (and vice versa).

Another interesting observation was that decreasing weight decay had mixed results (improving FTA for 1PA models but decreasing FTA for the original architecture) despite consistently increasing loss (with other factors held constant) across the board. I suspect it's because the original architecture simply had so many more parameters than the 1PA(-based) models (nearly four times as many parameters), so the presence of weight decay was helpful when there were more parameters (thus needing their values to be somewhat reined in) while it was detrimental (at least, less helpful) when there were fewer parameters (it may have overly reined in/restricted the ability of these fewer parameters to learn appropriate values). With that said, I was still left with the question of the disconnect/inconsistency between the FTL and FTA trends when varying weight decay as well. For lack of a more evident conclusion at the moment, I suspect this returns to the first observation I made and the probable effects of imbalanced classes in the Dermnet dataset I used for this study.

Lastly, some other promising observations I wanted to note are that the significantly lighter weight 1PA-based models were able to achieve comparable performance to the *much* larger models based on the original architecture and that dropout with the default dropout rate ($p = 0.5$) detracted very little or slightly increased the performance (classification accuracy) of the corresponding models without dropout while reliably lowering the test loss and validation losses. The former observation indicates that there is hope yet that, in future work, lighter weight models (e.g. by a factor of four as here) can keep up fairly well with much heavier models in performance while saving training and inference cost and overall being more deployant, and the latter observation signals that dropout is one of the most consistent, reliable regularizers we experimented with in the course of this project. While loss (training loss history, validation loss history, or final test loss) was not particularly correlated with or predictive of accuracy performance in this project, the ability and efficacy of dropout to regular-

ize this (or related) model architectures could prove useful when the class imbalance issue is better understood and/or mitigated (i.e. when there is a more typical, expected relationship between the various loss metrics and predictive accuracy/performance).

6.2. Further Work

In further work based on and expanding upon the work done and described in this project, I have several goals in mind and factors I'd like to investigate further. One fairly straightforward goal (with a far less straightforward path to achieving it) is that I'd like to work on improving classification prediction accuracy on this task. While it's certainly better than random ($\frac{1}{23} \approx 4.35\%$), the ballpark of $\approx 25\%$ for comprehensive test accuracy still leaves much to be desired with regards to performance. I'd probably experiment with a few other architectural modifications as well

Two other, possibly related goals for future work would be to investigate the (seemingly variable) tendency of the 1PA-based models trained and evaluated in this study to skew their class prediction distributions relative to the original architecture and to investigate the effects of the class imbalances in the (Dermnet) dataset on various aspects of the model's training and performance (particularly to better understand its effects on the relationship between training loss, validation loss, test loss, and accuracy/performance). I suspect that some of the effects of the class-imbalanced dataset may be in part responsible for the former issue of the 1PA-based models skewing their prediction distributions, so that would be an interesting hypothesis to investigate, potentially by using more granular evaluation metrics such as precision, recall, F1 score, and more. In addition to better understanding the effects of class imbalances in this specific case, I'd also like to attempt to mitigate these effects by exploring methods such as data augmentation, adjusting class weights during training and/or evaluation, and others.

While I still stand by my choice to attempt to tackle this 23-class classification task, I also think it might be worth trying to develop a model based on a subset of this Dermnet dataset containing fewer classes but still more than other projects have tackled (e.g. 10-12 clases) as an intermediary task which is hopefully more feasible while being a productive, meaningful stepping stone toward my larger, more ambitious goal introduced in this report.

References

[1] P. Gupta, J. Nirmal, and N. Mehendale. A survey on computer vision approaches for automated classification of skin diseases. *Multimedia Tools and Applications*, pages 1–33, 2024. 2

[2] S. E. Sorour, A. A. Hany, M. S. Elredeny, A. Sedik, and R. M. Hussien. An automatic dermatology detection system based on deep learning and computer vision. *IEEE Access*, 11:137769–137778, 2023. 2