# Mole or Melanoma

Sophie Parsa (sjparsa), Tom Welch (tcwelch), and Drew Wadsworth (swads)

**Introduction**

Skin cancer is becoming an ever more dire health crisis in the United States, with 1 in 5 Americans expected to develop skin cancer by age 70 and an estimated 2 Americans dying every hour [14]. While not the most common type of skin cancer, melanoma is the most lethal. In cancer treatment, one of the most important predictors of patient outcome is early detection. We aim to develop machine learning models for early detection. This can save lives because the accessibility and ease of use of a good cancer detection model may make people more likely to check themselves for melanoma. In prior research, one study estimated that when given similar melanoma identification tasks based on images, doctors varied between 69% and 91% accuracy whereas the ML models were correct 72% and 94% of the time [15]. This bodes very well for ML models to help in melanoma detection.

As input for all our models, we use a dataset of labeled melanoma and mole skin images that are processed, normalized, and reduced into a numerical matrix representation of the pixel values. We worked our way up in model complexity, beginning with training a logistic regression classifier, then a deep learning neural network, then a convolutional neural network, and finally a transfer learning model that uses the pretrained ResNet50 network followed by a second, trainable neural network. All of these models output a trained set of weights that can be used to predict the classification of melanoma or not melanoma (mole) on new data.

**Related work**

In prior papers, techniques such as CNNs and transfer learning have been used. Nasr-Esfahani, et al. produced a 7 layer convolutional neural network with accuracy 0.81 [1]. The authors of this paper used a very interesting approach to correct for/leverage the fact that images contain more than just the lesion in question. They applied a k-means classifier with k=2 to separate the lesion region of the image from the healthy skin region. Then, they applied a Gaussian blur on the healthy region to reduce any effect skin texture may have on the classifier. The resulting processed data was passed to their CNN. However, from the literature, it seems to be the case that transfer learning can produce even better results than single CNNs. Carcagnì et al. used transfer learning on top of DenseNet-121 to achieve an accuracy of 89.2% [2]. With transfer learning using ResNet50 as the base model and freezing the first roughly 90% of the resulting model's layers, Rasul, et al. achieved an accuracy of 0.9579 [3]. Rasul, et al. tried image segmentation, similarly to Nasr-Esfahani, et al. but using CNNs to predict the correct image mask instead of k-means. In Rasul, et al., the accuracy using masked images was actually slightly lower than with unmasked images. Similar approaches using convolutional networks and/or transfer learning are the state of the art for melanoma classification, which is typically done by the human eye currently. Argenziano, et al. demonstrated that mis-classification of melanoma by healthcare professionals is common, with many times more benign lesions removed from patients than actual melanoma lesions, which "increases morbidity and healthcare costs" [4].

Other papers have explored different methods, including other models and augmentation of data to produce larger training sets that can also result in a more versatile model. Pham, et al. noted that images of skin lesions can be translated, cropped, rotated, blurred, and flipped to both produce more data and to train a model that correctly classifies lesions whether or not the image is taken from a different angle or translated [5]. Codella, et al. used a support vector machine (SVM) that takes as input features extracted by a CNN [6]. This is a more novel approach than most, as most transfer learning approaches use some trainable DNN layers on top of the base model. This approach worked well for Codella, et al., however, achieving an accuracy of 0.931.

**Dataset and Features**

We use a balanced dataset consisting of images of skin lesions labeled as either "Melanoma" or "Not Melanoma" [7]. Our training data consists of 5,341 examples from each class; the validation data consists of 1,781 from each class; and the test data consists of 1,781 examples from the true class and 1,780 examples from the false class. Most images in the original dataset are 224x224 pixels, while some of the images are 600x450 pixels. For our models, we want a single input shape, so we preprocess all images to a resolution of 224x224 pixels. To do so, we downscale each 600x450 image by a factor of 2 and crop off the edges of the image (i.e. select the center of the image) to arrive at a 224x224 image. This works because the lesions are centered on the images in the original dataset, so by taking the center of the image, we keep the part that matters. Before passing it into a model, we normalize the input data. For a flat vector, we subtract the mean of the vector from each feature and divide by 225, the maximum possible value, to produce normalized values in the range $(-1, 1)$. For a 3D tensor with 3 channels (R, G, and B), we normalize in the same way per channel.

The largest complication caused by our dataset is the fact that our entire dataset does not fit in memory. That is, we have a total of $2 \times 5,341 = 10,682$ images in our training set. When read into memory as a 224x224x3 tensor, python stores each element as an 8 byte float. So, our training set has a size of$10,682 \times 224 \times 224 \times 3 \times 8B = 13GB$, which is too large for many modern computers. In hindsight, we could have used the numpy dtype feature to reduce the size of each element in memory by a factor of 8, to the data-type uint8, as the RGB values read from the image files range from 0-255. This method would have been a problem when the

time came to normalize the data, however, so we could instead have represented the data as a 2 byte float. However, we opted to solve the size problem with a more scalable, yet slower, approach: by creating a generator object that inherits from keras.utils.Sequence. First, this generator produces a list of unique identifiers for each image in the dataset, and it can optionally shuffle this list or keep the list in order. Then, this generator produces batches of data of the desired size by loading batch_size images into memory and arranging the examples into a tensor of input data and a vector of labels. The generator can be called repeatedly to fetch each batch of data. Unfortunately, because the data does not all fit into memory, each example needs to be reloaded into memory from disk on every epoch. To speed up the process of fetching data, Keras provides multithreaded training on the CPU side. So, we can launch 8 worker threads on the CPU, which concurrently fetch data from disk into memory, improving data bandwidth to the model. Without using multithreading on the CPU, the training is bottlenecked by fetching data from disk, but with multithreading, data fetches catch up to the actual training algorithm running on the GPU. A consideration here is that the data generator needs to be designed such that it can be run concurrently by multiple threads without race conditions or deadlocks. Our generator was designed with this in mind — our method of ordering the data by keeping a list of unique identifiers each corresponding to a given example guarantees that no two threads will try to access the same data.

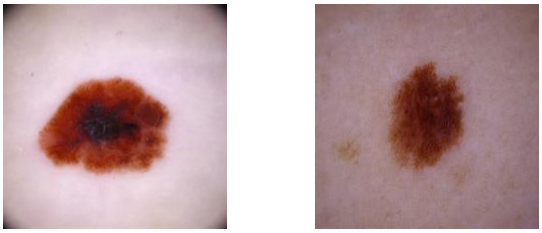


Figure 1 - (Left) Melanoma; (Right) Not Melanoma

**Methods**

Logistic Regression - We started with the standard logistic regression in a simple deep neural network and tuned hyper parameters to optimize predictive accuracy. This model uses a sigmoid function ($\sigma(z) = \frac{1}{1+e^{-z}}$) of the learned weights and the input as the hypothesis. This yields predicted probabilities between 0 and 1, with a threshold determining the actual assignment of positive or negative. Our logistic regression model optimizes the binary cross entropy loss function ($-\frac{1}{N}\sum_{i}^{N} y_i \cdot log(\widehat{y_i}) + (1 - y_i) \cdot log(1 - \widehat{y_i})$) by using stochastic mini batch gradient ascent to fit the model weights. We trained this classifier on a set of 3000 images and used the validation set accuracy to tune learning rate and batch size. For the learning rate we tested 1,0.1, 0.01, 0.001, and 0.001. The step sizes of 0.001 and 0.0001 did not appear to converge in reasonable time. On the validation set a learning rate of 1 yielded 0.515 accuracy, a learning rate of 0.1 yielded accuracy 0.5209, and a learning rate of 0.01 yielded 0.5138 accuracy on the validation sets. We therefore choses 0.1 as our final step size as it yielded the highest accuracy of the three remaining candidates. We also tuned the batch size for our mini batch by testing batch sizes of 20, 100, 200, and 500. Batch size 20 had accuracy 0.5051, batch size 100 had accuracy 0.5106, batch size 200 had accuracy 0.5147, and batch size 500 had accuracy 0.5249. Because changing the batch size and learning rate was not having a huge effect on accuracy despite the range of values being large, we did not do more extensive tuning of these parameters.

We then also implemented a Keras logistic regression classifier that trains using a batch size of 500 and ten training epochs to see if the model would be an improvement over our implementation, and so that we could run logistic regression on the entire training dataset used for the more complex models. When run on the full training dataset, we did not see any significant differences in the accuracy on the validation and test sets, but the accuracy on the training set did decrease now that the whole dataset was used so the resulting model appears less overfit (discussed further in results).

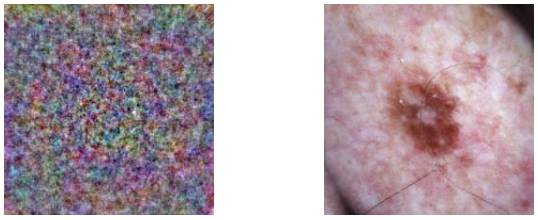


Figure 2 - (left) Logistic Regression Weight Visualization; (right) Pertinent Melanoma Example

After training the logistic regression model we saved the weights and tried to visualize them as an RGB image to see if we could visually identify key components of the image that the model was identifying. Figure 2 shows the results of this. While the visualization of the logistic weights shown in the left panel of Figure 2 does not greatly resemble a melanoma example as shown on the right, we do see circular clusters forming that are the same color. This indicates that our model is learning some sort of spatial awareness about clusters of skin that have different colors, as shown by the red and tan patches in the melanoma example

Neural Networks - The next logical jump in model complexity from logistic regression is a neural network (NN). A NN works on the same principles as logistic regression with the exception that instead of fitting just one node, one linear function and one activation, a NN consists of many connected nodes. A single layer will contain multiple nodes and each node in that layer will take the activation of all the previous layer's nodes as its input in the fully connected format. NN's are typically set up with the data fully connected to a large first layer with subsequently smaller layers, with the last layer being a single node in binary classification. With this architecture NN's train the linear parameters of each node on the binary cross entropy loss function. One key difference is that neural nets use ReLU activation functions, $ReLU(z) = max(0, z)$, for all nodes except the node before the output, where the sigmoid function is used. Intuition of NN's is supplemented by realizing that each node might learn a separator for a different attribute of an input, for instance the color or shape of a mole. The attributes learned in one layer can then be thought to help separate higher level attributes, such as malignancy, in subsequent layers. A key advantage of the NN architecture is that it can separate class data points with more than a single linear separator as we had in logistic regression.

In our NN, we followed the convention described above and trained on layers with neuron counts as follows: 1024;256;64;16;4;1. Initial training showed errors with vanishing gradients so we resolved this by adding glorot-uniform initialization for kernels and zero initialization for bias terms and normalized our input. This modification allowed our program to converge more quickly and achieve perfect accuracy on the training set indicating the need for L2 regularization, which adds losses for linear parameters' magnitudes to prevent overfitting. After adding L2 regularization the model performs well.

Convolutional Neural Networks - Convolutional Neural Networks (CNN's) are similar to NNs but are optimized for image data inputs, making them the ideal next model to test. This time, data is input as a volume: pixel width x pixel height x 3 (for red, green and blue pixel intensities). Instead of using a linear function of all of the input features, a parameter tensor shaped as a subset of the width and height of the input with equal number of channels is elementwise multiplied and summed by part of the input to give a scalar output. This process is repeated by shifting the filter across the input and storing the scalar outputs in a matrix in positional order. Multiple filters can be applied to make the output of one layer of the CNN and the output matrices are stacked as a volume to be input into the next layer. Before passing to the next layer, the elementwise ReLU is taken, dimensions are reduced by pooling together values in the volume to the max value in that region, the volume is normalized across channels and is sometimes zero padded. Typically, layers of the CNN progress such that the volume decreases in height and width and increases in number of channels. At the end of the network, the volume is flattened and fed through a small NN with the same architecture described previously. Early layers of CNNs might recognize edges while later layers recognize mole shapes and eventually higher order characteristics that a doctor might use to visually identify melanoma. The key advantage of the CNN is the ability to reuse learned parameters from one filter across all input features of a layer, allowing CNNs to use less parameters and computation power for the same task as NNs. Therefore, more parameters can be added for the same computational cost to get better performance. For reference, our CNN used 737,537 trainable parameters while our NN used 154,421,657 trainable parameters, which is about 210 times more than the CNN.

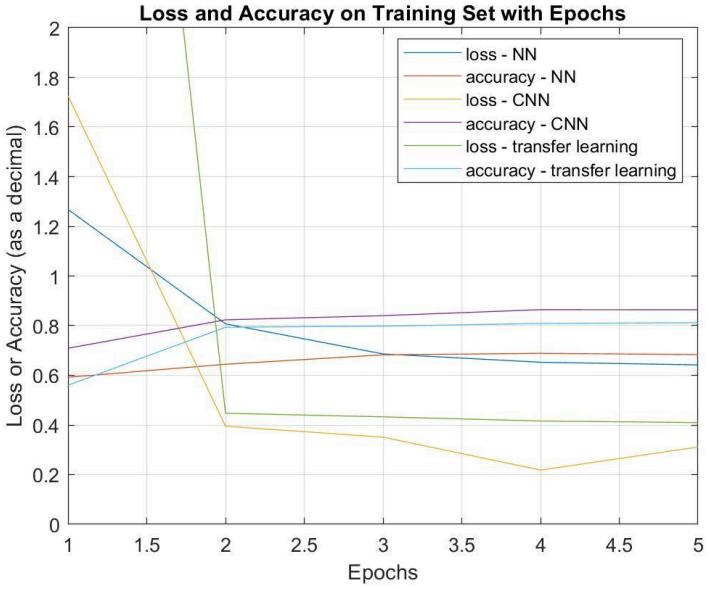
Our CNN follows this formatt with the following structure: zp(3,3), CONV2D(f=7,s=1,c=32), BN, ReLU, maxpool(f=2, s=2), zp(2,2), CONV2D(f=5,s=1, c=64), BN, ReLU, maxpool(f=2, s=2), zp(1,1), CONV2D(f=3,s=1, c=128), BN, ReLU, maxpool(f=2, s=2), CONV2D(f=3,s=3, c=128),BN,ReLU, maxpool(f=3, s=3), flatten, FC(n=512), FC(n=32), FC(n=1). (Where zp is zero padding along (height,width), CONV2D is a set of c filters of (height, width) = f with strides s, maxpool is across a fxf slice of a volume taking stride length s, and FC is a fully connected layer with n neurons.) We trained with an Adam optimizer to give our algorithm "momentum" on the binary cross entropy loss.

Transfer Learning - The final model we implemented was a transfer learning model that builds on top of the pre-trained ResNet50 model, which has been trained on the ImageNet database and is accessible as a Keras Application. We remove the top layer from the ResNet model, normalize and flatten the output from ResNet, and add 5 trainable fully-connected layers onto the end of the model. The neuron counts of the trainable layers are as follows: 4096;1024;256;32;1. The activation function used for each hidden layer is ReLU, and the activation function for the output layer is sigmoid. Overall, this creates a model where the first set of layers is the pre-trained convolutional neural network — each layer of which we freeze during training — followed by additional trainable layers to the end of the model. The resulting model uses the pre-trained model to identify features in the input data and the new layers to classify the image as melanoma/not melanoma, based on the features extracted from the input images by the pre-trained model. This

aids in efficiency and allows us to make use of previous work in the broader field of image classification. Further work applying transfer learning could potentially improve the resulting model by using fine-tuning. That is, after sufficiently training only the layers we have added, we could unfreeze the layers of the base ResNet50 model and train the entire model for a small number of epochs, in an effort to further improve accuracy and specialize the model for the purpose of melanoma classification. However, when fine tuning, we would want to be careful to not overfit the training data, so we could use a smaller learning rate.

**Results**

After implementing the methods described above, we found some interesting results. Logistic regression did not perform very well, because of its simplicity and inability to properly separate the classes, and it was only slightly better than random guessing with an accuracy of 52.3% and area under the ROC curve (AUROC) of 0.521 on the test set (see Fig. 5). As expected, the NN, having more parameters than logistic regression to separate data, performed slightly better with an accuracy of 63.2% and AUROC of 0.641 on the test set. With the NN we notice that we only get a higher true positive rate (TPR) with a high false positive rate (FPR) (see Fig. 4). In practice we want a high TPR even if we get some false positives, because it is better to diagnose someone with melanoma who doesn't have it than to give false reassurance that someone with melanoma doesn't have it, but ideally we'd want a high TPR with a low FPR. Intuitively, this means that if we set the NN decision threshold high we'd get a high TPR and FPR and perform notably better than random guessing. Continuing the trend we see that our CNN has the best performance with an accuracy of 86.1% and AUROC of 0.943 on the test set. We also see from the ROC curve that it fills the area and is farthest away from random guessing. The increased performance of the CNN can likely be attributed to translation invariance as the location of the melanoma in the image does not change its classification, whereas for logistic regression and NN's it does. What's surprising is that the transfer learning with our deep neural network attached performs worse than the CNN coming in with an accuracy of 80.9% and AUROC of 0.896 on the test set. We do see that it performs much better than the NN on its own as it is a larger network, but we expected it to perform better than the CNN since ResNet50 was trained on a much larger dataset for many more epochs. The difference in performance might be attributed to the fact that ResNet50 wasn't trained on melanoma and non-melanoma images alone. This could lead to filters in deeper layers within ResNet50, which may separate higher order attributes that are pertinent to other applications but not melanoma classification. These filters would then assign low weights to the unrelated melanoma images and render that branch of filters useless to our application. This suggests that the useful, pre-trained filters in ResNet50, are less in number than our CNN's trained filters.



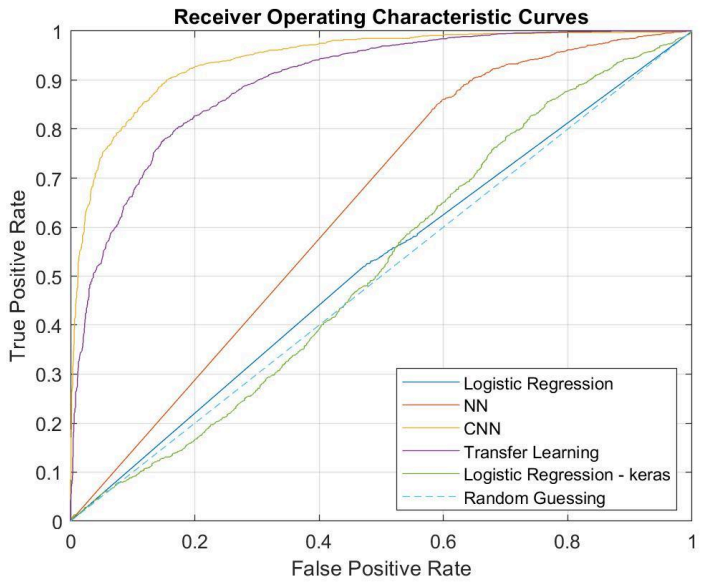




Figure 3 (left) - Loss and Training Accuracy with Epochs of Training; Figure 4 (right) - ROC Curves for each Model

From our trained models, we see that our minibatch size, learning rate, and general parameter tuning, described in the methods section, worked well as we observe convergence within the first five epochs (see Fig. 3). Another thing that is interesting to observe is that our initial models, which were trained and tested on a dataset with 1,000 examples, performed better than our current results, which were trained on about 10,600 examples. This suggests that there is quite a bit of variance within the dataset. When we observe bias and variance across the models trained on the full dataset, we find that some have a larger variance problem and others have a

larger bias problem. Logistic regression for instance has 99.1% accuracy on the training dataset, but only 52.1% accuracy on the test dataset. This suggests a high variance which is indicative of overfitting, indicating that training on more examples or adding regularization would improve the model (note that logistic regression was only run on 3,000 examples). On the other hand, from Fig. 5 we see that the NN and transfer network had about the same train and test performance but have a low overall accuracy, suggesting high bias and underfitting. This could be aided by a larger network, training for longer or with better algorithms, or by doing more hyper parameter tuning. Finally, the CNN has a train accuracy of 91.3% and test accuracy of 86.1% suggesting that there seems to be a balance in bias and variance issues, and that the CNN could be improved by a combination of the described techniques.

| Model | Set | Accuracy | Precision | Sensitivity | Specificity | F1 score | AUCROC |
|---|---|---|---|---|---|---|---|
| Logistic Regression | Train | 0.991 | 0.991 | 0.991 | 0.991 | 0.991 | 0.996 |
| | Validation | 0.525 | 0.509 | 0.535 | 0.516 | 0.522 | 0.527 |
| | Test | 0.523 | 0.522 | 0.515 | 0.515 | 0.527 | 0.521 |
| Deep NN | Train | 0.633 | 0.889 | 0.590 | 0.768 | 0.709 | 0.643 |
| | Validation | 0.628 | 0.882 | 0.590 | 0.749 | 0.707 | 0.631 |
| | Test | 0.632 | 0.882 | 0.593 | 0.753 | 0.709 | 0.641 |
| CNN | Train | 0.913 | 0.886 | 0.939 | 0.890 | 0.912 | 0.976 |
| | Validation | 0.866 | 0.834 | 0.896 | 0.839 | 0.864 | 0.939 |
| | Test | 0.861 | 0.816 | 0.902 | 0.826 | 0.857 | 0.943 |
| Transfer Learning | Train | 0.820 | 0.860 | 0.797 | 0.846 | 0.828 | 0.906 |
| | Validation | 0.822 | 0.862 | 0.903 | 0.846 | 0.832 | 0.904 |
| | Test | 0.809 | 0.854 | 0.788 | 0.834 | 0.820 | 0.896 |

Figure 5 - Table of Metrics for each Model on Train, Validation and Test Sets (main metrics highlighted)

**Conclusion**

After designing, tuning and running experiments on a logistic regression, a NN, a CNN, and a transfer learning model, we found the CNN to perform best with an accuracy of 86.1% and AUROC of 0.943 on the test set. We see that at 86.1% accuracy, our CNN performs comparably with the average doctor (69% to 91% accuracy) and the average ML model (72% to 94% accuracy) [15]. The logistic regression and NN on their own performed insufficiently, and the transfer network performed only slightly worse than the CNN. We attribute the high performance of the CNN to its translation invariance and its ability to train more quickly on fewer, shared parameters in the same number of epochs as the other algorithms. In the future we'd like to focus more attention on our CNN. We noticed that there was room for improvement in terms of bias and variance for CNN. We plan to address bias by making a larger network, training for longer, and by doing more hyper parameter tuning. To address variance we'd hope to implement data augmentation to increase the size of our dataset and add regularization. With some more work, our CNN architecture might one day be good enough to implement as an app so that anyone could check for melanoma as an early warning system.

**Contributions**
We all worked together at a high level to decide what models we wanted to use and figure out basic methods. We also all helped each other with debugging the many issues encountered when coding the models and setting up training environments. For specific code implementation, Tom designed, coded, debugged, and trained the NN, the CNN, and the NN applied after transfer learning. He also wrote the functions that calculate, plot, and save metrics for predictions, roc curves and other metrics such as accuracy and F1 score. Tom worked together with Sophie and spent a decent amount of time trying to get saliency maps working, but then we focused our attention on printing the weights from logistic regression. Tom and Drew also worked together to get GCP set up. Tom wrote the sections for the results, conclusion and methods for the CNN and NN. Sophie wrote code to implement our own logistic regression model first using vectorized batch gradient descent and then using stochastic mini batch gradient descent. She did work to tune hyperparameters for the model and then also implemented a Keras logistic regression model. She also wrote feature analysis code to extract weights and visualize them and helped with some modifications to image processing to accommodate transfer learning. She did preliminary work running PCA on the training data to see if any component could explain a lot of the variance in the data but we did not end up including this in our final models. Sophie and Drew worked together on implementing a transfer learning model. Sophie wrote the introduction and parts of the results and methods on logistic regression, weight visualization, and transfer learning. Drew wrote the data preprocessing code and the data pipeline that pulls batches of examples from disk and organizes them into tensors for ingestion by each model. He did a first run of the models on a GCP instance to debug and evaluate training speed and modified both the models and the data pipeline to allow CPU multiprocessing. Drew also wrote code to downsize the dataset with random selection for development purposes. Drew did the literature review, and based on the literature, chose a base model and methodology for the transfer learning model. Drew edited each of the models to allow pulling single batches from disk at a time, which was easy for the DNNs (due to Keras) but required a separate implementation of training and inference on our hand-written logistic regression. Drew outlined the logistic regression parameter analysis and helped Sophie with the implementation of weight visualization. For this report, Drew wrote the related work section and the data section, and he co-wrote the transfer learning section with Sophie.

**Github URL:** https://github.com/drewads/cs229_project

**References**

[1] E. Nasr-Esfahani et al. Melanoma detection by analysis of clinical images using convolutional neural network. 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2016, pp. 1373-1376, doi: 10.1109/EMBC.2016.7590963.

[2] Pierluigi Carcagnì , Andrea Cuna , and Cosimo Distante. A Dense CNN Approach for Skin Lesion Classification. 26 July 2018. https://arxiv.org/pdf/1807.06416.pdf.

[3] M. F. Rasul, N. K. Dey and M. M. A. Hashem. A Comparative Study of Neural Network Architectures for Lesion Segmentation and Melanoma Detection. 2020 IEEE Region 10 Symposium (TENSYMP), 2020, pp. 1572-1575, doi: 10.1109/TENSYMP50017.2020.9230969.

[4] G. Argenziano, et al. Accuracy in melanoma detection: A 10-year multicenter survey. Journal of the American Academy of Dermatology, Volume 67, Issue 1, 2012. https://doi.org/10.1016/j.jaad.2011.07.019.

[5] Pham TC., Luong CM., Visani M., Hoang VD. Deep CNN and Data Augmentation for Skin Lesion Classification. Lecture Notes in Computer Science, vol 10752, 2018. Springer, Cham. https://doi.org/10.1007/978-3-319-75420-8_54.

[6] Codella N., Cai J., Abedini M., Garnavi R., Halpern A., Smith J.R. Deep Learning, Sparse Coding, and SVM for Melanoma Recognition in Dermoscopy Images. Lecture Notes in Computer Science, vol 9352, 2015. Springer, Cham. https://doi.org/10.1007/978-3-319-24888-2_15.

[7] https://www.kaggle.com/drscarlat/melanoma (our dataset)

[8] C.R. Harris, K.J. Millman, S.J. van der Walt et al. Array programming with NumPy. Nature 585, 357–362. 2020. DOI: 0.1038/s41586-020-2649-2. (Publisher link).

[9] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing

Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[10] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

[11] P Umesh. “Image Processing in Python”. CSI Communications, 23. 2012.
This is the citation for the PIL code library used in our implementation.

[12] Van Rossum, G. (2020). The Python Library Reference, release 3.8.2. Python Software Foundation.

[13] Chollet, F., & others. (2015). Keras. https://keras.io.

[14]“Skin Cancer Facts & Statistics.” *The Skin Cancer Foundation*, 1 Apr. 2021, www.skincancer.org/skin-cancer-information/skin-cancer-facts/.

[15]Randal White MD. “Machine Learning Primer for Clinicians–Part 12.” *HIStalk*, histalk2.com/2019/01/16/machine-learning-primer-for-clinicians-part-12-2/.