Efficacy of a Convolutional Neural Network at Skin Cancer Detection

Ronan Konishi

Abstract—With the growing trend of big data, global smartphone subscriptions, and high processing systems, the development of a smartphone application using convolutional neural network shows potential at providing effective and accessible universal skin cancer diagnosis. Despite the recent successes at machine learning-based skin cancer detection, there is a lack of success when testing these algorithms on smartphone applications. The purpose of this study is to develop and test the efficacy of a deep learning, feedforward, supervised learning, LeNet model, convolutional neural network that is compatible with smartphones. Although the data indicate unfavorable results, this study still provides a groundwork for future studies to follow.

Index Terms—Convolutional neural networks, cancer detection, image classification, machine learning

1 Introduction

SKIN cancer is prevalent in the United States, afflicting approximately one in five Americans by the age of 70 [1]; in addition, the number of patients diagnosed with skin cancer each year is greater than that of all other forms of cancers combined [2]. At such an alarmingly high rate, it is imperative that precautions are taken before the cancer can spread and become life-threatening. One significant factor in decreasing the likelihood of death from skin cancer, especially when dealing with melanoma, is early diagnosis and treatment [3]; although the other more common forms of skin cancer, basal and squamous cell carcinomas, are less lethal, early diagnosis is still important in alleviating the excessive growth of malignant tumors and lymph nodes, which if continued to be left untreated, may also lead to death [4].

Despite the importance of regular doctor checkups for diagnosing skin cancer during its earlier stages, there are numerous factors that dissuade people from seeking aid. As stated in a Stanford news article, some patients live too far from a doctor, struggle financially, or simply lack the time for the necessary checkups [5]. However, a novel solution to such problems may be emerging in today's technologically advancing society due to the rapidly increasing global usage and availability of smartphones. As predicted in the 2013 Ericsson Mobility Report [6], there will be a total of 9.3 billion mobile subscriptions in 2019, of which 5.6 billion will be smartphone subscriptions. Therefore, a smartphone application that can reliably perform skin cancer diagnosis at a performance level on par with (or even superior to) professional doctors will make early detection for skin cancer conveniently and globally accessible; artificial intelligence demonstrates great potential at attaining this high-performance level.

2 Motivation

In this age of data proliferation, there is a substantial reliance on computers to discern patterns within large sets of data with a level of efficiency and effectiveness unattainable by humans [7]. More so in the medical field, artificial intelligence shows promise in its ability to analyze big data and perform accurate medical diagnoses [8]. In 2017, Kavya Kopparapu, a high school senior, invented a smartphone application for diagnosing diabetic retinopathy (eye cancer) by using a convolutional neural network algorithm called Res-Net-50 [9], [10]. The network was trained to detect anomalies in the eye using the data of approximately 34,000 retinal scans from the National Institutes of Health [9], [10]. She stated that "[of the] 415 million diabetics worldwide, one-third will develop retinopathy. Fifty percent will be undiagnosed. Of patients with severe forms, half will go blind in five years" [9], [10]. From a more general perspective, the idea of creating a smartphone application for diagnosing diabetic retinopathy can be applied to all types of diseases, which will save more money, time, and lives [11].

3 Related Works

Over the past two decades, there has been a multitude of methods used to classify skin lesions (an adnormal change in the skin). In 1995, a study on applying neural classifiers for classifying skin lesions applied texture statistics returned promising results [12]. Since then, there have been numerous other methods tested for classifying skin lesions including statistics, K-nearest neighborhood, classification and regession trees, ADWAT methods, and artificial neural networks [13].

In a Stanford study, the use of a deep learning convolutional neural network was used to classify skin cancer and was found to be very effective at doing so [11]. The system was able to learn how to classify skin lesions by using the pixels from the images and the labels of the diseases as inputs into their program [11]. By using a dataset of 129,450 clinical images with 2,032 different disease labels, the system tested for the most common and deadliest skin cancers: keratinocyte carcinomas vs benign seborrheic keratosis and malignant melanomas vs benign nevi, respectively [11]. It was found that the system performed as well as experts on both tasks, demonstrating the capabilities of applying artificial intelligence to skin cancer classification [11].

Although studies indicate promising results when using convolutional neural networks, there is a lack of success when applying these methods to practical situations, especially with smartphones. An analysis of 4 smartphone applications (written for the 2 most popular smartphone platforms) showed poor accuracy when tested to discern whether a given skin cancer lesion was melanoma or not [14]. The sensitivity ranged from 6.8% to 98.1% (a measurement of how often an image was identified as positive (melanoma) when the image was labeled as positive), while the specificity ranged from 30.4% to 93.7% (a measurement of how often an image was identified as negative (not melanoma) when the image was labeled as negative) [14]. The lowest accuracy was found for the application that used an automated algorithm to analyze the images [14].

Currently, there are multiple major research institutes hoping to apply their artificial intelligence techniques to smartphones. For example, the students in the Stanford study who created the deep learning convolutional neural networks system for skin cancer detection claimed that “[it would] be relatively easy to transition the algorithm to mobile devices but there still needs to be further testing in a real-world clinical setting” [5]. In addition, NVIDIA, a company that built a deep learning algorithm that classifies skin lesions as being benign or malignant as accurately as a dermatologist, stated that “the team intends to continue testing its solution in real-world clinical settings before attempting to establish the algorithm as the basis for a mobile application” [15]. Despite the recent claims for future tests on smartphones, currently, there is still a lack of successful tests with smartphone applications.

4 PROPOSED METHOD

The method for classifying skin lesions in this study was a deep learning, feedforward, convolutional neural network that used supervised learning [16] - [18]: deep learning refers to the idea that a program learns how to solve a particular problem given a set of data to learn from, feedforward refers to a artificial neural network model that does not cycle backwards, supervised learning refers to using input data with known labels, and convolutional neural network refers to an architecture specifically designed for when the inputs are images. The process was as follows: the information in the database was preprocessed by labeling, transforming, and sorting the data into a training and testing dataset; the newly formatted data from the training dataset was then fed into a deep learning classification model through which the convolutional neural network optimized its values of influence [16], [18]; and finally the testing data passed through the trained model to evaluate the effectiveness of the model.

4.1 International Skin Cancer Imaging Collaboration (ISIC) Archive

The ISIC skin cancer archive is a collection of multiple sub-databases each containing skin cancer lesion JPG files and corresponding informational JSON files. The sub-databases used were MSK1, MSK2, MSK3, MSK4, MSK5, SONIC, UDA1, UDA2, and Dermoscopedia; they were later all combined into a single directory, which totaled to 13,791 pairs of JPG and JSON files in the database used for this project [19].

4.2 Image Preprocessing

Image preprocessing refers to the transformation, normalization, labeling, and categorization of the ISIC dataset [20].

Image transformation was performed by scaling the image size to a uniform aspect ratio of 1 to 1 (length to width) and an equal pixel range of 360 pixels by 360 pixels by 3 pixels (length by width by depth) [20]. This was necessary because the convolutional layers are unable to properly convolve features over unequally dimensioned images (referred to in 4.3.1.1).

Normalization refers to the compression of pixel values from all images from a range of 0 through 255 to 0 through 1, which reduces computational complexity [20]. The purpose of using all three color channels (red, green, and blue) rather than a single grayscale channel was to ensure no skin cancer indicators were lost from removing image color.

Labelization refers to the labeling of each image as either malignant or benign by extracting the description from each image’s corresponding JSON file [20]. However, during this process, there was an issue of values in the JSON file under “benign_malignant” not being equal to either benign or malignant. To work around this, values not equal to either benign or malignant were automatically sent to a garbage disposal directory. Thus, the total number of images used in this project decreased by approximately 30 images.

Categorization refers to the random sorting of images and labels into one of two subsets labeled as either training or testing in a distribution ratio of approximately 75% and 25%, respectively, as suggested by Andrew Ng [21]. The training subset is used by the deep learning model to train the convolutional neural network to correctly identify the images. The testing dataset is used after the convolutional neural network is trained to evaluate how well the convolutional neural network classifies the images. The splitting of the dataset was achieved by assigning each image and its respective label a randomized value ranging from 0.0 to 1.0 and placing the image in the testing subset if the value was greater than 0.25, otherwise placing it inside the training subset. Although this method is not the most accurate for exact distributions, in this study, such factor is insignificant: as long as there is sufficient data in the training and testing dataset at roughly a 75 to 25 ratio, the convolutional neural network will still function properly.

4.3 Deep Learning-based Classification LeNet Model

As shown in figure 1, the deep learning convolutional neural network model chosen for this study was a simple LeNet model, chosen for this study because of its minimal data and processing power requirements, which is necessary for smartphone compatibility [Fig. 1]. The model consists of two major parts: a feature extraction model (convolutional, Leaky ReLU, and maximum pooling layers) and a classification model (fully connected and output layer) [22]. The purpose of the duplicated convolutional, Leaky ReLU, and maximum pooling layers is to enable the convolutional neural network to obtain more sophisticated feature extractions. Also, the final maximum pooling layer adds an additional reduction in the processing power and data requirements. Backpropagation and Stochastic Gradient Descent were the learning mechanics for the convolutional neural network.

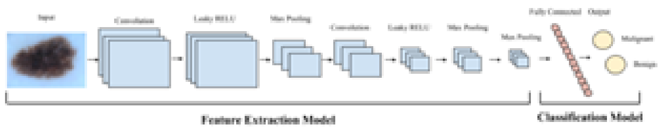

Figure 1: LeNet Model
Source: Adapted from [22]

4.3.1 *Feature Extraction Model*

The purpose of the feature extraction model is to filter out features that characterize a malignant or benign image. This model also reduces the size of the image after performing feature extractions to prepare the image for the classification model.

4.3.1.1 *Convolutional Layer*

The purpose of the convolutional layer is to extract features from the input images that are indicative of being either malignant or benign [23]-[27]. This layer consists of an equal number of filters and filtered output images. The filter convolves over each input images in a given mini-batch, in which the filter is layered over the images (the size determined by a set kernel size), and the dot product (product of each pair of overlapping pixels) is calculated [28]. The total sum of each dot product and the bias is then calculated to represent the entire cluster of pixel values from the input images as a single value on the newly filtered image. This calculation is performed over the images of the entire mini-batch starting from the top left cluster of pixels until it reaches the top right and shifting down a given number of pixels defined by the value of the vertical stride until it reaches the bottom left cluster of pixels. Then the process is repeated for the next filter until all of the filters are applied [28].

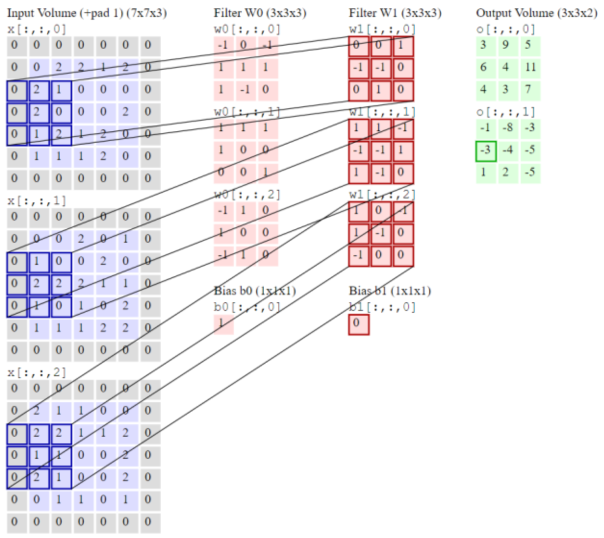

Figure 2: Convolutional Layer
Source: Adapted from [28]

4.3.1.2 *Leaky ReLU Layer*

The Leaky ReLU layer attempts to minimize the effects of negative pixel values resulting from the calculations taken in the preceding convolutional layer [25]-[27]. This layer checks each pixel values: if the value is positive, it does nothing, if the value is negative, it gets scaled to a value very close to zero [25]-[27]. The reason for maintaining a slightly negative value for the pixel (which cannot practically exist), is because a study by Cornell University shows that by doing so, the convolutional neural network performs more accurately [29].

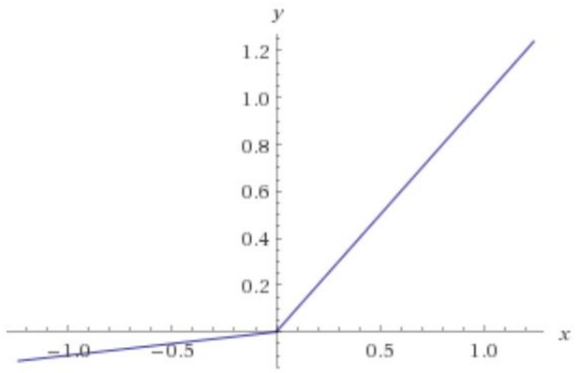

Figure 3: Leaky ReLU
Source: Adapted from [30]

4.3.1.3 *Maximum Pooling Layer*

Maximum pooling attempts to alleviate the immense processing power required by the convolutional neural network by scaling the image down [25]-[27]. Starting from the top left cluster of pixels (determined by the kernel size), until the bottom right cluster, the largest values from each bunch is calculated to represent the given cluster in the newly filtered image [25]-[27].

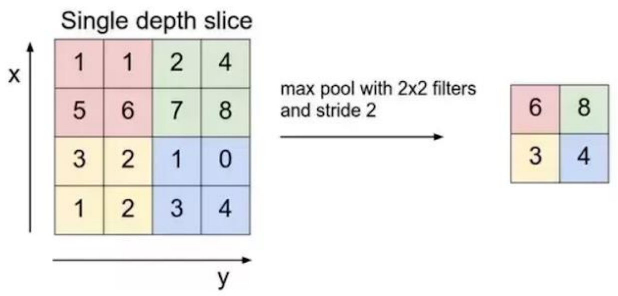

Figure 4: Maximum Pooling Layer
Source: Adapted from [31]

4.3.2 *Classification Model*

The Classification Model uses the pixel values from the filtered minibatch images by giving each value a weight, or influence, on intermediary values, which represent more sophisticated values in a fully connected layer. The values from the fully connected layer also have an influence on the output values that determine how likely given images are benign or malignant those images are. The values of the output layer will total a sum of one because of the Softmax function used.

4.3.2.1 Fully Connected Layer

The fully connected layers attach each pixel value from the previous pooling layer to each value in the fully connected layer [26], [27]. Each pixel from the input batch of images has an influence on each of the values in the fully connected layer [Fig. 5] (note that the image does not accurately represent the number of values in each layer). The influences are the tunable values, which refers to the learning of the convolutional neural network [26], [27].

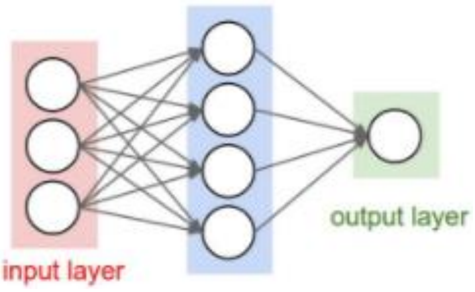

Figure 5: Fully Connected Layer
Source: Adapted from [27]

4.3.2.2 Output Layer and Softmax

Connecting the neurons from the fully connected layer to the output layer, is a function called Softmax, which forces the values in the output neurons to total a sum of one [32]. The output layer consists of a value labeled malignant and a value labeled benign.

4.3.3 Stochastic Gradient Descent and Backpropagation

Backpropagation is an algorithm used to determine how the influences of the convolutional neural network should be adjusted in the most optimal and efficient manner [33]. After a given input mini-batch of images passes through the entire convolutional neural network and ends up at the output layer, the model score for that single mini-batch can be found by taking the sum of the squared differences between the values from the output layer and the values from the labeled vectors (if benign (1,0) and if malignant (0,1)) [33]. Then by using backpropagation, the values for the influences are tuned in accordance to the given mini-batch [33].

Stochastic Gradient Descent is the process by which the convolutional neural network adjusts the values of the influence (the speed training determined by the learning rate) to attain the lowest model score [33], [34]. With stochastic gradient descent, the convolutional neural network trains in sets of images called mini-batches, so that when backpropagation is used to tune the influences, the tweaks are representative of the changes determined only by that given mini-batch [33], [34]. By performing backpropagation in this process, the convolutional neural network can be trained at a much faster rate [33], [34].

4.3.4 Momentum

Momentum is a method often used in tandem with Stochastic Gradient Descent. The problem with Stochastic Gradient Descent and the use of mini-batches to speed up the learning process is a high and rigid learning path (the path for the model score), is the excess time spent following the overly long and inflexible learning path [35]. Thus, momentum attempts to round off the learning path, allowing for a smoother path, drastically speeding the learning speed [35]. The momentum value sets how large of a curve the learning path takes, while the updater is the method by which the momentum is implemented [35].

4.3.5 Overfitting and Regularization

The problem of overfitting arises when the values of influence for a given convolutional neural network are tuned too well for the given training data. This results in the inability to generalize a prediction for any new inputs, as any new inputs will be strongly swayed by the extreme values of influences tuned to fit every single data in the training data. Two common methods used to address the issue of overfitting include the reduction in the number of features, which will force a more general solution tuned for by the convolutional neural network, or the implementation of regularization, which reduces the magnitude of the values of influences (the prefered method with convolutional neural networks and the one used in this study). The type of regularization used was called L2 regularization [36]. The level of impact of the regularization component is determined by the hyperparameter value called weight decay.

4.3.6 Weight Initialization

Weight initialization refers to the values of which the values of influence start at before any training is performed. It was suggested to use a the most common randomization algorithm called ReLU to initialize these values of influence [36].

5 Training Data

Dataset	Mini test dataset
Learning rate	0.001
Updater	Nesterov's
Momentum	0.9
Weight decay	0.005
Mini-batch size	20
Epochs	3
Number of Channels	3
Image dimensions	360x360
Optimization Algorithm	Stochastic Gradient Descent
Weight init	RELU
1st Conv kernel size	5x5
1st Conv stride	1x1
1st Conv # of Filters	20
1st Max Pooling kernel size	2x2
1st Max Pooling stride	2x2
2nd Conv kernel size	5x5
2nd Conv stride	1x1
2nd Conv # of Filters	50
2nd Max Pooling kernel size	2x2
2nd Max Pooling stride	2x2
Activation function	Softmax

Table 1: Hyperparameters from Test 1, LeNet CNN, where hyperparameters are based on banana classifier [22], mini dataset

Dataset	Full dataset
Learning rate	0.001
Updater	Nesterov's
Momentum	0.9
Weight decay	0.005
Mini-batch size	20
Epochs	1
Number of Channels	3
Image dimensions	360x360
Optimization algorithm	Stochastic Gradient Descent
Weight init	RELU
1st Conv kernel size	5x5
1st Conv stride	1x1
1st Conv # of Filters	20
1st Max Pooling kernel size	2x2
1st Max Pooling stride	2x2
2nd Conv kernel size	5x5
2nd Conv stride	1x1
2nd Conv # of Filters	50
2nd Max Pooling kernel size	2x2
2nd Max Pooling stride	2x2
Activation function	Softmax

Table 2: Hyperparameters from Test 2, single epoch (number of times run through entire dataset), full dataset

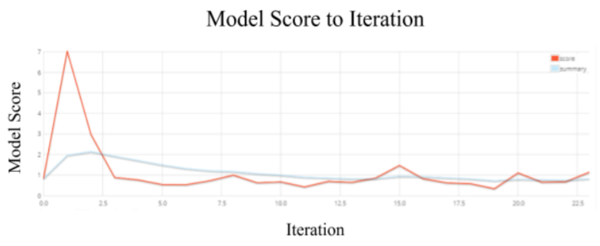

Figure 6: Model Score to Iteration Graph from Test 1

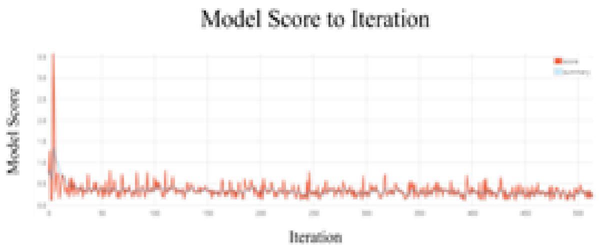

Figure 8: Model Score to Iteration Graph from Test 2

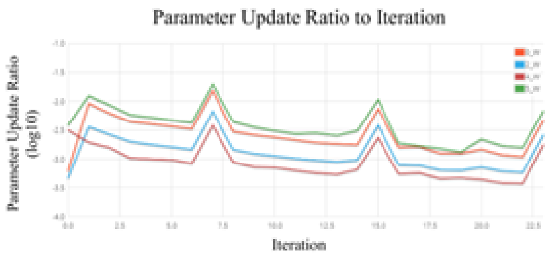

Figure 7: Parameter Update Ratio to Iteration Graph from Test 1

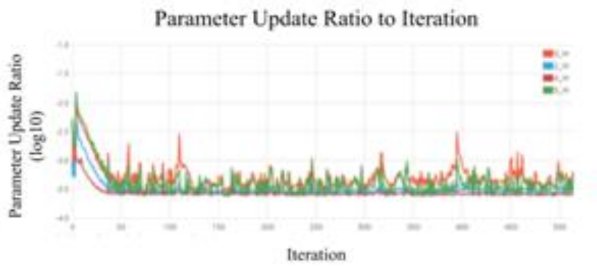

Figure 9: Parameter Update Ratio to Iteration Graph from Test 2

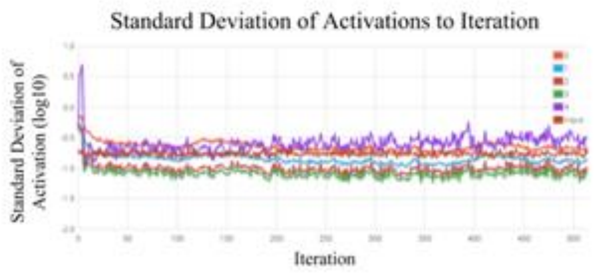

Figure 10: Standard Deviation of Activations to Iteration Graph from Test 2

Dataset	Full Dataset
Learning rate	0.001
Updater	Adam
Momentum	N/A
Weight decay	0.005
Mini-batch size	20
Epochs	1
Number of Channels	3
Image dimensions	360x360
Optimization algorithm	Stochastic Gradient Descent
Weight init	RELU
1st Conv kernel size	5x5
1st Conv stride	1x1
1st Conv # of Filters	20
1st Max Pooling kernel size	2x2
1st Max Pooling stride	2x2
2nd Conv kernel size	5x5
2nd Conv stride	1x1
2nd Conv # of Filters	50
2nd Max Pooling kernel size	2x2
2nd Max Pooling stride	2x2
Activation function	Softmax

Table 3: Hyperparameters from Test 3, Adam Updater, with evaluation output, full dataset

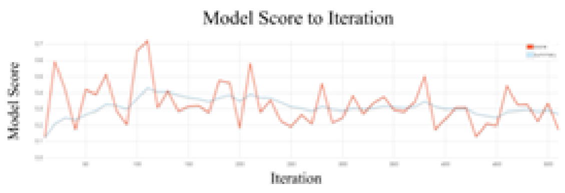

Figure 11: Model Score to Iteration Graph from Test 3

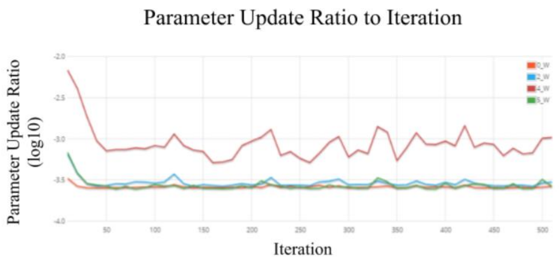

Figure 12: Parameter Update Ratio to Iteration Graph from Test 3

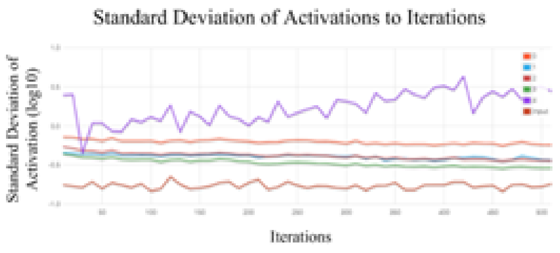

Figure 13: Standard Deviation of Activations to Iteration Graph from Test 3

Examples labeled as 0 classified by model as 0:	2611 times
Examples labeled as 0 classified by model as 0:	566 times
Examples labeled as 0 classified by model as 0:	211 times
Examples labeled as 0 classified by model as 0:	76 times

# of Classes	Accuracy	Precision	Recall	F1 Score
2	0.7735	0.5202	0.5389	0.1619

Table 4: Evaluation Table from Test 3

6 Evaluation Results

Test 1: LeNet model, where hyperparameters were based on banana classifier

The purpose of the first test was to analyze the efficacy of the skin cancer classification convolutional neural network when using a method called transfer learning. Transfer learning [37] is the application of a set of hyperparameters already tuned for another similar machine learning problem. In this case, a banana disease classifier which had the hyperparameters below, acted as a template for the skin cancer detection model.

Parameter	Optimization algorithm	Learning Rate	Momentum	Weight Decay	Batch Size	Activation Function	Iterations
Value	Stochastic Gradient Descent	0.001	0.9	0.005	10	Sigmoid	30

Table 5: Banana Classifier Hyperparameters Adapted from [22]

To save time, the amount of data trained on the convolutional neural network was a little under 2% that of the entire dataset. From analyzing the model score to iteration graph, the results appear to indicate accurate readings [38], and the learning rate seems to be at a satisfactory good level since the graph closely resembles that of figure 14 [28]. In addition, because the model score steadily decreased as the number of iterations increased, the network should have been properly learning to correctly classify images [38]. From analyzing the parameter ratio to iterations graph, another evaluation could be made about the learning rate, which is that because the ratio is approximately 10^-3, the learning rate is at a good value [38]. One concern from this graph was the sudden spikes at around iteration 7, 15 and 23, but they may have simply been caused by the repeating of data (given that the epoch was set to 3). Also, this graph did not diverge past 10^-2 and 10^-4 (when not considering the spikes), thus the weights could be assumed to have been set to a good enough value to learn useful features [38]. Although the spikes were not concerning enough to apply a different weight initialization to try and help alleviate the spikes, one may attempt to do so in a future test.

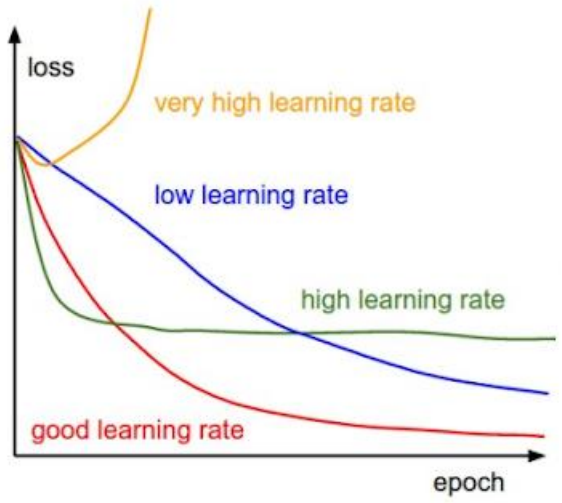

Figure 14: Model Score to Iteration Graph with Various Learning Rates
Source: Adapted from [28]

Test 2: Single epoch

After comparing the results from the training of the skin cancer detecting convolutional neural network with that of an ideally trained convolutional neural network [38], the next step was to train with the entire dataset. After training, the dataset showed a model score to iteration function with a good learning rate, averaging out to about 0.3, which is something commonly seen from an optimally set training rate, normalization, optimization method, and mini batch quantity [38]. Also, the parameter ratio to iteration graph evened out to a value of about 10^-3, and never peaked past 10^-2 or 10^-4, indicating a good learning rate and weight initialization [38]. However, there were a few unusual spikes in the graph at iteration 50, 100, and 400 that prompted some questioning. After consulting Alex Black, a computer science graduate from Monash University, with over 5 years of experience in the artificial intelligence industry, who currently works at SkyMind, he stated that the spikes were not too concerning, but gradient chipping was an optional method that could reduce such spikes if concerned (it was not implement in this study). He also advised that the updater be changed from Nesterov's to Adam and that the iterations only display in intervals of 10 to prevent stark oscillations in the graph, which makes analysis of the graphs difficult. Because the standard deviation of activations to iterations graph stabilizes after a few hundred iterations to a value between 0.5 and 2.0, the weight initializations, regulation, data normalization, and learning rates could be said to be at ideal values [38].

Test 3: Adam Updater, with evaluation output

When training the convolutional neural network with the updated parameters, the model score to iteration graph remained relatively low throughout the training process (approximately 0.3), although not following the typical path of a well-trained convolutional neural network [Fig. 14]. Looking at the light blue summary path, it could be described as slightly increasing (which may indicate an excessively large learning rate or an incorrect data normalization) or flat/slowly decreasing score (which may indicate an excessively small learning rate or an unideal optimization method) [38]. Also, from looking at the parameter ratio to iterations chart, the values seem to diverge to 10^-4, with only a single weight floating around 10^-3. In addition, the standard deviation of activation to iterations chart indicates values between 10^-1.5 to 10^0.5. After consulting Alex Black, he recommended running an evaluation test with the convolutional neural network to analyze how well the convolutional neural network is really training. After running that test, the results indicated a high rate of incorrectly classifying the image. The evaluation table showed that the convolutional neural network correctly identified the image as either malignant or benign 77.35 percent of the time [16]. The percent of images labeled malignant out of all the skin lesions predicted as either correctly or incorrectly malignant was 52.02 [16]. The percent of correctly detected malignant images out of all the skin lesions labeled malignant was 53.89 [16]. A combination of both precision and recall can be described by the F1 score which was 16.15 [16].

7 Discussion

Despite finishing with undesirable test results, the tests still give rise to a few concerns and considerations for future testing. After consulting Adam Black about the results, he stated that the class imbalance (unequal distribution of malignant and benign images) had been a major issue. When dealing with class imbalance, depending on the ratio (in this case 1:13, malignant: benign), there are several steps that should be taken. As with this case, the imbalance is not too drastic, and can likely be compensated for by changing the threshold value for the evaluation class. By plotting a Receiver Operating Characteristics (ROC) curve and an area under the ROC curve [39], [40], the desired thresholds for the evaluation tests can be obtained.

Another problem with the convolutional neural network could be the images in the datasets themselves. The dataset used to train the convolutional neural network was a compilation of smaller datasets obtained from the ISIC-archive, with one of them labeled as SONIC. Although most of the images were clean, within the sub dataset labeled SONIC were images with additional objects on the side of the skin lesion. Thus, the convolutional neural network may have tried to find patterns within those insignificant objects, which would have negatively impacted the training and evaluation results. Recently, the addition of a new dataset from a study called "Human-Against-Machine" contained 10,000 images, all of which contained much cleaner and desirable images [19], [41]. A future test with only these images should be considered.

As with smartphone compatibility, although no tests were directly run on a smartphone, the program used from the preceding tests are compatible with smartphones. The deeplearning4j library that was used to develop the software in this project has a well-tested built-in library for Android capabilities, which allows for the easy integration with smartphones. In addition, the software was developed in a way so that the training could be performed on the computer, and after, the trained network with the tuned hyperparameters, weights, and biases can be transferred to a smartphone via a .zip file. The evaluation and testing on the testing dataset can also be performed on the computer before exporting the .zip file on the convolutional neural network, all of which significantly reduces the memory and processing load required by the smartphone. In addition, the .zip file only requires a total of about 0.686 gigabytes of memory, making it feasible to create an application under 1 gigabytes.

Due to the lack of time, many components of this project were incompletable and left untested. Future works for this project could include developing the smartphone application to ensure that the processor can properly handle the program and the amount of memory on a typical smartphone is sufficient. In addition, the smartphone application should have picture taking capabilities to enable testing on live subjects, rather than on only pre-gathered images.

8 Conclusion

Although, the results of this study do not indicate an effective convolutional neural network, the potential in applying convolutional neural networks for skin cancer classification is hardly diminished. On the contrary, this study provides a steady groundwork for future studies regarding machine learning-based classification problems. By avoiding major issues encountered during this study, including the use of an undesirable set of database images and the neglect of optimizing the threshold value when evaluating the convolutional neural network, one can use this study as a guide to start his or her own machine-learning classification project. One may also choose to continue from where this study concluded, by downloading this open source software, finish training the convolutional neural network, making it compatible with smartphones, and testing it in the real world.

References

[1] H. Rogers, M. Weinstock, S. Feldman and B. Coldiron, "Incidence Estimate of Nonmelanoma Skin Cancer (Keratinocyte Carcinomas) in the US Population, 2012", *JAMA Dermatology*, vol. 151, no. 10, p. 1081, 2015.

[2] *Cancer Facts & Figures*. Atlanta: American Cancer Society, 2018 [Online]. Available: https://www.cancer.org/content/dam/cancer-org/research/cancer-facts-and-statistics/annual-cancer-facts-and-figures/2018/cancer-facts-and-figures-2018.pdf. [Accessed: 07- Jan- 2018]

[3] "Survival Rates for Melanoma Skin Cancer, by Stage", *Cancer.org*, 2016. [Online]. Available: https://www.cancer.org/cancer/melanoma-skin-cancer/detection-diagnosis-staging/survival-rates-for-melanoma-skin-cancer-by-stage.html. [Accessed: 15- Jan- 2018]

[4] "Basal and Squamous Cell Skin Cancer Stages", *Cancer.org*, 2017. [Online]. Available: https://www.cancer.org/cancer/basal-and-squamous-cell-skin-cancer/detection-diagnosis-staging/staging.html. [Accessed: 15- Jan- 2018]

[5] T. Kubota, "Deep learning algorithm does as well as dermatologists in identifying skin cancer", *Stanford News*, 2017 [Online]. Available: https://news.stanford.edu/2017/01/25/artificial-intelligence-used-identify-skin-cancer/. [Accessed: 16- Jan- 2018]

[6] Ericsson, "Ericsson Mobility Report: Global smartphone subscriptions to reach 5.6 billion by 2019", Ericsson, Stockholm, 2013 [Online]. Available: http://mb.cision.com/Main/15448/2245678/661730.pdf. [Accessed: 23- Jan- 2018]

[7] U. Fayyad, G. Piatetsky-Shapiro and P. Smyth, "From Data Mining to Knowledge Discovery in Databases", *AI Magazine*, vol. 17, no. 3, 2018 [Online]. Available: https://www.aaai.org/ojs/index.php/aimagazine/article/viewFile/1230/1131. [Accessed: 09- Jan- 2018]

[8] S. Dilsizian and E. Siegel, "Artificial Intelligence in Medicine and Cardiac Imaging: Harnessing Big Data and Advanced Computing to Provide Personalized Medical Diagnosis and Treatment", *Current Cardiology Reports*, vol. 16, no. 1, 2014.

[9] A. Bleicher, "Teenage Whiz Kid Invents an AI System to Diagnose Her Grandfather's Eye Disease", *IEEE Spectrum*, 2017. [Online]. Available: https://spectrum.ieee.org/the-human-os/biomedical/diagnostics/teenage-whiz-kid-invents-an-ai-system-to-diagnose-her-grandfathers-eye-disease. [Accessed: 15- Dec- 2018]

[10] S. Dean, "Science Alert", *This Teenage Girl Invented an AI-Based App That Can Quickly Diagnose Eye Disease*, 2017. [Online]. Available: https://www.sciencealert.com/this-teenage-girl-invented-a-brilliant-ai-based-app-that-can-quickly-diagnose-eye-disease. [Accessed: 15- Apr- 2018]

[11] A. Esteva, B. Kuprel, R. Novoa, J. Ko, S. Swetter, H. Blau and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks", *Nature International Journal of science*, vol. 542, no. 7639, pp. 115-118, 2017.

[12] M. Hintz-Madsen, L. Hansen, J. Larsen, E. Olesen and K. Drzewiecki, "Design and evaluation of neural classifiers application to skin lesion classification", *Neural Networks for Signal Processing [1995] V. Proceedings of the 1995 IEEE Workshop*, 2002.

[13] A. Hoshyar, A. Al-Jumaily and R. Sulaiman, "Review on automatic early skin cancer detection", *2011 International Conference on Computer Science and Service System (CSSS)*, 2011.

[14] J. Wolf, J. Moreau, O. Akilov, T. Patton, J. English, J. Ho and L. Ferris, "Diagnostic Inaccuracy of Smartphone Applications for Melanoma Detection", *JAMA Dermatology*, vol. 149, no. 4, p. 422, 2013.

[15] T. Kontzer, "How an AI App Can Translate a Photo into a Skin Cancer Diagnosis", *NVIDIA*. 2017 [Online]. Available: https://blogs.nvidia.com/blog/2017/05/23/ai-app-skin-cancer-diagnosis/. [Accessed: 08- Jan- 2018]

[16] A. Ng, *Data Science, Machine Learning by Andrew Ng*. Stanford: Stanford, 2014.

[17] Y. Abu-Mostafa, *Learning From Data, Machine Learning course*. Pasadena: California Institute of Technology, 2012.

[18] Eclipse Deeplearning4j Development Team, "Deeplearning4j: Open-source distributed deep learning for the JVM", 2018. [Online]. Available: https://deeplearning4j.org/. [Accessed: 15- Apr- 2018]

[19] International Skin Imaging Collaboration archive. [Online} Available: https://isic-archive.com/. Accessed Apr. 10, 2018.

[20] Eclipse Deeplearning4j Development Team, "Customized Data Pipelines for Images etc.", *Deeplearning4j*, 2018. [Online]. Available: https://deeplearning4j.org/simple-image-load-transform. [Accessed: 27- Jan- 2018]

[21] A. Ng, *Train / Dev / Test sets*. 2018 [Online]. Available: https://www.coursera.org/learn/deep-neural-network/lecture/cxG1s/train-dev-test-sets. [Accessed: 06- Apr- 2018]

[22] J. Amara, B. Bouaziz and A. Algergawy, "A Deep Learning-based Approach for Banana Leaf Diseases Classification", *PubMed*, 2017 [Online]. Available: https://www.btw2017.informatik.uni-stuttgart.de/slidesandpapers/E1-10/paper_web.pdf. [Accessed: 14- Mar- 2018]

[23] A. Ng, *Strided Convolutions*. 2018 [Online]. Available: https://www.coursera.org/learn/convolutional-neural-networks/lecture/wfUhx/strided-convolutions. [Accessed: 05- Apr- 2018]

[24] *Convolutional Neural Networks (CNNs) explained*. 2017 [Online]. Available: https://www.youtube.com/watch?v=YRhxdVk_sIs. [Accessed: 04- Mar- 2018]

[25] A. Deshpande, "A Beginner's Guide To Understanding Convolutional Neural Networks Part 2", 2016. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/. [Accessed: 15- Apr- 2018]

[26] B. Rohrer, *How Convolutional Neural Networks work*. 2016 [Online]. Available: https://www.youtube.com/watch?v=FmpDIaiMIeA. [Accessed: 03- Apr- 2018]

[27] L. Serrano, *A friendly introduction to Convolutional Neural Networks and Image Recognition*. Mountain View: Udacity, 2018.

[28] J. Johnson and A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition", 2018. [Online]. Available: http://cs231n.github.io/neural-networks-3/. [Accessed: 19- Mar- 2018]

[29] B. Xu, N. Wang, T. Chen and M. Li, "Empirical Evaluation of Rectified Activations in Convolutional Network", *Cornell University Library*, 2015 [Online]. Available: https://arxiv.org/abs/1505.00853. [Accessed: 23- Jan- 2018]

[30] "What is the “dying ReLU” problem in neural networks?", *Stack Exchange*, 2016. [Online]. Available: https://datascience.stackexchange.com/questions/5706/what-is-the-dying-relu-problem-in-neural-networks. [Accessed: 17- Apr- 2018]

[31] A. Karpathy and J. Johnson, "CS231n Convolutional Neural Networks for Visual Recognition", 2018. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 17- Apr- 2018]

[32] A. Ng, *Training a Softmax Classifier*. 2018 [Online]. Available: https://www.coursera.org/learn/deep-neural-network/lecture/LCsCH/training-a-softmax-classifier. [Accessed: 29- Mar- 2018]

[33] G. Sanderson, *Deep Learning*. 2017 [Online]. Available: http://3b1b.co/neural-networks. [Accessed: 10- Mar- 2018]

[34] Udacity, *Stochastic Gradient Descent*. 2016 [Online]. Available: https://www.youtube.com/watch?v=hMLUgM6kTp8. [Accessed: 05- Apr- 2018]

[35] University of Toronto, *3. The Momentum Method*. 2013 [Online]. Available: https://www.youtube.com/watch?v=8yg2mRJx-z4. [Accessed: 28- Mar- 2018]

[36] J. Johnson and A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition", 2018. [Online]. Available: http://cs231n.github.io/neural-networks-2/. [Accessed: 19- Mar- 2018]

[37] A. Karpathy and J. Johnson, "CS231n Convolutional Neural Networks for Visual Recognition", 2018. [Online]. Available: http://cs231n.github.io/transfer-learning/. [Accessed: 17- Apr- 2018]

[38] Eclipse Deeplearning4j Development Team, "Visualize, Monitor and Debug Network Learning", *Deeplearning4j*, 2018. [Online]. Available: https://deeplearning4j.org/visualization. [Accessed: 27- Jan- 2018]

[39] R. Patwari, *ROC Curves*. 2013 [Online]. Available: https://www.youtube.com/watch?v=21Igj5Pr6u4. [Accessed: 17- Apr- 2018]

[40] K. Markham, *ROC Curves and Area Under the Curve (AUC) Explained*. 2014 [Online]. Available: https://www.youtube.com/watch?v=OAl6eAyP-yo. [Accessed: 17- Apr- 2018]

[41] P. Tschandl, C. Rosendahl and H. Kittler, "The HAM10000 Dataset: A Large Collection of Multi-Source Dermatoscopic Images of Common Pigmented Skin Lesions", *Cornell University Library*, 2018 [Online]. Available: https://arxiv.org/abs/1803.10417. [Accessed: 17- Apr- 2018]

APPENDIX

Main.java

```java
package com.research.skindetector;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.IOException;
import java.util.Random;

 /**
 * The heart of the program that calls everything needed to run.
 *
 *
 *
 *
 * @author
 * @version 1.0
 */
 public class Main {

    static Random ranNumGen;
    static JsonImageRecordReader recordReader;

    //hyperparameters
    static double learningRate = 0.001;
    static double momentum = 0.9;
    static double weightDecay = 0.005;

    static int rngseed = 123;

    static int height = 360; //of image
    static int width = 360; //of image
    static int nChannels = 3; // Number of input channels
    static int outputNum = 2; // The number of possible outcomes
    static int batchSize = 20; //batch size for Stochastic Gradient Descent

    private static Logger log = LoggerFactory.getLogger(Main.class);

    public static void main(String[] args) throws IOException {
 //        String trainedpath = "C:\\Users \\Documents\\AP Research\\skin-detector\\trained_model.zip"; //uncomment if
using an already trained model
        File mixedData = new File("C:\\Users\\Desktop\\ISIC-images\\mixedData\\");
        File trainData = new File("C:\\Users\\Desktop\\ISIC-images\\trainData\\");
        File testData = new File("C:\\Users\\Desktop\\ISIC-images\\testData\\");
        NeuralNetwork network = new NeuralNetwork(mixedData, trainData, testData, rngseed, height, width, nChannels,
batchSize, outputNum);

        network.buildNet(learningRate, momentum, weightDecay);

        log.info("*****TRAIN MODEL********");
        network.train();

        log.info("*****ENABLE UI********");
        network.UIenable();

//        log.info("*****SAVE TRAINED MODEL******");
//        network.saveBuild("trained_model.zip");

        log.info("*****EVALUATE MODEL*******");
        log.info(network.evaluate().stats());
    }
}
```

NeuralNetwork.java

```java
package com.research.skindetector;

import org.datavec.api.split.FileSplit;
import org.datavec.image.loader.NativeImageLoader;
import org.deeplearning4j.api.storage.StatsStorage;
import org.deeplearning4j.datasets.datavec.RecordReaderDataSetIterator;
import org.deeplearning4j.datasets.iterator.AsyncDataSetIterator;
import org.deeplearning4j.eval.Evaluation;
import org.deeplearning4j.eval.ROC;
import org.deeplearning4j.evaluation.EvaluationTools;
import org.deeplearning4j.nn.api.OptimizationAlgorithm;
import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.WorkspaceMode;
import org.deeplearning4j.nn.conf.inputs.InputType;
import org.deeplearning4j.nn.conf.layers.ConvolutionLayer;
import org.deeplearning4j.nn.conf.layers.DenseLayer;
import org.deeplearning4j.nn.conf.layers.OutputLayer;
import org.deeplearning4j.nn.conf.layers.SubsamplingLayer;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.nn.weights.WeightInit;
import org.deeplearning4j.parallelism.ParallelWrapper;
import org.deeplearning4j.ui.api.UIServer;
import org.deeplearning4j.ui.stats.StatsListener;
import org.deeplearning4j.ui.storage.InMemoryStatsStorage;
import org.deeplearning4j.util.ModelSerializer;
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.DataSet;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.dataset.api.preprocessor.DataNormalization;
import org.nd4j.linalg.dataset.api.preprocessor.ImagePreProcessingScaler;
import org.nd4j.linalg.factory.Nd4j;
import org.nd4j.linalg.learning.config.Adam;
import org.nd4j.linalg.learning.config.Nesterovs;
import org.nd4j.linalg.lossfunctions.LossFunctions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.Random;

/**
* Convolutional Neural Network class that applies Supervised Learning for Skin Cancer Detection.
*
* This is the main class that builds, trains, and evaluates the neural network.
*
* Based on deeplearning4j open source library and tutorials.
*
* @author
* @version 1.0
*
*/
public class NeuralNetwork {
   private static Logger log = LoggerFactory.getLogger(Main.class);
   File trainData, testData;
```

```java
    int rngseed, height, width, channels, batchSize, outputNum;
    String netPath;
    Random ranNumGen;
    MultiLayerNetwork model;
    JsonImageRecordReader recordReader;
    DataNormalization scaler;
    DataSetIterator test_iter, train_iter;
    int numEpochs = 1; //number of times trained through dataset

    /**
     * Constructor for non-distinguished training and testing data (have all data in single directory, mixed data).
     * Also used if needing to build a neural network.
     * Note that you still must call the .build() function in the main class
     *
     * @param mixedData Path to file with mixed data
     * @param rngseed Integer that allows for constant random generated value
     * @param height The height of image in pixels
     * @param width The width of image in pixels
     * @param channels The number of channels (e.g. 1 for gray scaled and 3 for RGB)
     * @param batchSize The number of images in a given minibatch
     * @param outputNum The number of nodes in the output layer
     * @throws IOException
     */
    public NeuralNetwork(File mixedData, File trainData, File testData, int rngseed, int height, int width, int channels, int batchSize, int outputNum) throws IOException {
        this.trainData = trainData;
        this.testData = testData;
        this.rngseed = rngseed;
        this.height = height;
        this.width = width;
        this.channels = channels;
        this.batchSize = batchSize;
        this.outputNum = outputNum;
        this.netPath = netPath;
        ranNumGen = new Random(rngseed);
        vectorization();
        dataSplitter(mixedData, trainData, testData);
        log.info("Building Neural Network from scratch...");
    }

    /**
     * Constructor for non-distinguished training and testing data (have all data in single directory, mixed data).
     * Also used if importing an already built neural network.
     *
     * @param mixedData Path to file with mixed data
     * @param rngseed Integer that allows for constant random generated value
     * @param height The height of image in pixels
     * @param width The width of image in pixels
     * @param channels The number of channels (e.g. 1 for gray scaled and 3 for RGB)
     * @param batchSize The number of images in a given minibatch
     * @param outputNum The number of nodes in the output layer
     * @param netPath The path from which the neural network is being imported
     * @throws IOException
     */
    public NeuralNetwork(File mixedData, File trainData, File testData, int rngseed, int height, int width, int channels, int batchSize, int outputNum, String netPath) throws IOException {
        this.trainData = trainData;
        this.testData = testData;
        this.rngseed = rngseed;
        this.height = height;
        this.width = width;
        this.channels = channels;
        this.batchSize = batchSize;
        this.outputNum = outputNum;
        this.netPath = netPath;
        ranNumGen = new Random(rngseed);
        vectorization();
        dataSplitter(mixedData, trainData, testData);
        log.info("Building Neural Network from import...");
        loadNet(netPath);
    }

    /**
     * Constructor for preemptively defined training and testing data.
     * Also for if needing to create a neural network.
     * Note that you still must call the .build() function in the main class
     *
     * @param trainData Path to file with training data
     * @param testData Path to file with
     * @param rngseed Integer that allows for constant random generated value
     * @param height The height of image in pixels
     * @param width The width of image in pixels
     * @param channels The number of channels (e.g. 1 for gray scaled and 3 for RGB)
     * @param batchSize The number of images in a given minibatch
     * @param outputNum The number of nodes in the output layer
     * @throws IOException
     */
    public NeuralNetwork(File trainData, File testData, int rngseed, int height, int width, int channels, int batchSize, int
outputNum) throws IOException {
        this.trainData = trainData;
        this.testData = testData;
        this.rngseed = rngseed;
        this.height = height;
        this.width = width;
        this.channels = channels;
        this.batchSize = batchSize;
        this.outputNum = outputNum;
        this.netPath = netPath;
        ranNumGen = new Random(rngseed);
        vectorization();
        log.info("Building Neural Network from scratch...");
//        buildNet();
    }

    /**
     * Constructor for preemptively defined training and testing data.
     * Also for if importing an already built neural network.
     *
     * @param trainData Path to file with training data
     * @param testData Path to file with
     * @param rngseed Integer that allows for constant random generated value
     * @param height The height of image in pixels
     * @param width The width of image in pixels
     * @param channels The number of channels (e.g. 1 for gray scaled and 3 for RGB)
     * @param batchSize The number of images in a given minibatch
     * @param outputNum The number of nodes in the output layer
     * @param netPath The path from which the neural network is being imported
     * @throws IOException
     */
    public NeuralNetwork(File trainData, File testData, int rngseed, int height, int width, int channels, int batchSize, int
```

```java
outputNum, String netPath) throws IOException {
        this.trainData = trainData;
        this.testData = testData;
        this.rngseed = rngseed;
        this.height = height;
        this.width = width;
        this.channels = channels;
        this.batchSize = batchSize;
        this.outputNum = outputNum;
        this.netPath = netPath;
        ranNumGen = new Random(rngseed);
        vectorization();
        log.info("Building Neural Network from import...");
        loadNet(netPath);
    }

    /**
     * Performs Image Preprocessing by labeling all of the data, scaling the images to a uniform,
     * and vectorizing the pixels values.
     */
    private void vectorization(){
        JsonPathLabelGenerator label = new JsonPathLabelGenerator();
        recordReader = new JsonImageRecordReader(height, width, channels, label);
//        recordReader.setListeners(new LogRecordListener()); //uncomment to check the label for each input data
    }

    /**
     * Builds a neural network using gradient descent with regularization algorithm.
     *
     * @param learningRate The rate at which the neural network learns
     * @param momentum The rate for increasing training rate (note that it was felt unused with Adam Updater)
     * @param weightDecay
     */
    public MultiLayerNetwork buildNet(double learningRate, double momentum, double weightDecay) {

        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
                .seed(rngseed) //saves this neural network with the given optimizations
                .iterations(1) // Training iterations as above
                .regularization(true).l2(weightDecay) //prevents overfitting
                .learningRate(learningRate) // alpha from gradient descent (how fast it goes down the gradient)
                .weightInit(WeightInit.RELU) //method of randomizing weights in a gaussian distribution with equal variance throughout each layer
                .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)// (ideal for big data and big models) uses randomized minibatches to compute cost and gradient descent, which iterates through many minibatches
                .updater(new Adam()) //helps remove oscillation, by rounding off. (helps to start with minimal momentum 0.5 and after following gradient path well, increase momentum)
                .list()
                .layer(0, new ConvolutionLayer.Builder(5, 5) //5x5 pixel feature, stride moves
                        //nIn and nOut specify depth. nIn here is the nChannels and nOut is the number of filters to be applied
                        .nIn(channels) //3
                        .stride(1, 1) //1px to right, then 1 px down
                        .nOut(20)
                        .activation(Activation.LEAKYRELU) // shrinks values (0-256 for R in RGB) from a value between x and 1 (where 0 to 1 is linear with slope of 1 and x to 0 is linear with minimal slope)
                        .build())
                .layer(1, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX) // takes max in 2x2 pixel and represents a new image where that 2x2 space equals a single pixel
                        .kernelSize(2,2)
                        .stride(2,2)
                        .build())
                .layer(2, new ConvolutionLayer.Builder(5, 5)
                        //Note that nIn need not be specified in later layers
                        .stride(1, 1)
                        .nOut(50)
                        .activation(Activation.LEAKYRELU)
                        .build())
                .layer(3, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
                        .kernelSize(2,2)
                        .stride(2,2)
                        .build())
                .layer(4, new DenseLayer.Builder().activation(Activation.LEAKYRELU).nOut(500).build())
                .layer(5, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                        .nOut(outputNum)
                        .activation(Activation.SOFTMAX)
                        .build())
                .setInputType(InputType.convolutionalFlat(height,width,channels))
                .backprop(true).pretrain(false).build();

        model = new MultiLayerNetwork(conf);
        model.init();

        return model;
    }

    /**
     * Gets the neural network model
     *
     * @return model
     */
    public MultiLayerNetwork getNet(){
        return model;
    }

    /**
     * Enables the User Interface to allow for analysis of the neural network training.
     */
    public void UIenable(){
        UIServer uiServer = UIServer.getInstance();

        StatsStorage statsStorage = new InMemoryStatsStorage(); //Alternative: new FileStatsStorage(File) - see UIStorageExample
        int listenerFrequency = 10;
        this.getNet().setListeners(new StatsListener(statsStorage, listenerFrequency));

        uiServer.attach(statsStorage);
        for(int i = 0; i < numEpochs; i++) {
            this.getNet().fit(this.getTrainIter());
        }
    }

    /**
     * Trains the neural network with the training data.
     *
     * @throws IOException
     */
    public void train() throws IOException {
        FileSplit train = new FileSplit(trainData, NativeImageLoader.ALLOWED_FORMATS, this.ranNumGen);

        recordReader.initialize(train);
        DataSetIterator temp_iter = new RecordReaderDataSetIterator(recordReader,batchSize,1,outputNum);
        scaler = new ImagePreProcessingScaler(0,1); //normalization
        scaler.fit(temp_iter);
        temp_iter.setPreProcessor(scaler);
        train_iter = new AsyncDataSetIterator(temp_iter);

//        model.setListeners(new ScoreIterationListener(10)); //Uncomment to display how well the neural network is training

        //disables the java garbage collector
        Nd4j.getMemoryManager().setAutoGcWindow(5000);
        Nd4j.getMemoryManager().togglePeriodicGc(false);

        //Accommodates for java memory issues by optimizing usage of computing powers
        ParallelWrapper wrapper = new ParallelWrapper.Builder(model)
                .prefetchBuffer(8)
                .workers(2)
                .averagingFrequency(5)
                .reportScoreAfterAveraging(false)
                .workspaceMode(WorkspaceMode.SEPARATE)
                .build();
    }

    /**
     * Evaluates the neural network by running the network through the testing data set.
     *
     * @return eval The output of the evaluation
     * @throws IOException
     */
    public Evaluation evaluate() throws IOException {
        FileSplit test = new FileSplit(testData, NativeImageLoader.ALLOWED_FORMATS, ranNumGen);

        recordReader.initialize(test);
        test_iter = new RecordReaderDataSetIterator(recordReader,batchSize,1,outputNum);
        scaler = new ImagePreProcessingScaler(0,1);
        scaler.fit(test_iter);
        test_iter.setPreProcessor(scaler);

        Evaluation eval = new Evaluation(outputNum);

        ROC roceval = new ROC(outputNum);
        roceval.calculateAUC();
        roceval.calculateAUCPR();
        roceval.getRocCurve();
        roceval.getPrecisionRecallCurve();
        EvaluationTools.exportRocChartsToHtmlFile(roceval, new File("roc_chart_HAM.html"));
//        model.doEvaluation(iteratorTest, eval, roceval);

        while(test_iter.hasNext()) {
            DataSet next = test_iter.next();
            INDArray output = model.output(next.getFeatureMatrix());
            eval.eval(next.getLabels(), output);
        }
        return eval;
    }

    /**
     * Saves the trained neural network
     *
     * @param filepath The path and file to which the neural network should be save to
     * @throws IOException
     */
    public void saveBuild(String filepath) throws IOException {
        File saveLocation = new File(filepath);
        boolean saveUpdater = false; //want to enable retraining of data
        ModelSerializer.writeModel(model,saveLocation,saveUpdater);
    }

    /**
     * For testing purposes. Displays images with labels from a given database.
     *
     * @param numImages The number of images to display
     */
    public void imageToLabelDisplay(int numImages, DataSetIterator iter){
        for (int i = 0; i < numImages; i++) {
            DataSet ds = iter.next();
            System.out.println(ds);
            System.out.println(iter.getLabels());
        }
    }

    /**
     *
     * @param mixedDataset The path to the dataset with all of the data
     * @param trainData The path to which the training dataset should be placed
     * @param testData The path to which the testing dataset should be placed
     * @throws IOException
     */
    private void dataSplitter(File mixedDataset, File trainData, File testData) throws IOException {
        this.trainData = trainData;
        this.testData = testData;

        File[] mixedData = mixedDataset.listFiles();
        String temp1, temp2;
        for(int i = 0; i < mixedData.length/2; i++){
            double random = Math.random();
            temp1 = mixedData[i*2].toString();
            temp2 = mixedData[i*2+1].toString();

            if (random > 0.25) {
                Files.move(mixedData[i*2].toPath(), new File(trainData + "\\" + temp1.sub-
string(temp1.lastIndexOf('\\')+1)).toPath());
                Files.move(mixedData[i*2+1].toPath(), new File(trainData + "\\" + temp2.sub-
string(temp2.lastIndexOf('\\')+1)).toPath());
            } else {
                Files.move(mixedData[i*2].toPath(), new File(testData + "\\" + temp1.sub-
string(temp1.lastIndexOf('\\')+1)).toPath());
                Files.move(mixedData[i*2+1].toPath(), new File(testData + "\\" + temp2.sub-
string(temp2.lastIndexOf('\\')+1)).toPath());
            }
        }
    }

    /**
     * Loads the neural network
     *
     * @param NetPath The path to import the neural network from (should be .zip format)
     * @throws IOException
     */
    private void loadNet(String NetPath) throws IOException {
        File locationToSave = new File(NetPath);
        model = ModelSerializer.restoreMultiLayerNetwork(locationToSave);
    }

    /**
     * Gets the trained iterator
     *
     * @return train_iter
     */
    public DataSetIterator getTrainIter(){
        return train_iter;
    }
}
```

JsonPathLabelGenerator.java

```java
package com.research.skindetector;

import org.datavec.api.io.labels.PathLabelGenerator;
import org.datavec.api.writable.Writable;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import javax.json.Json;
import javax.json.JsonObject;
import javax.json.JsonReader;
import javax.json.JsonValue;
import java.io.File;
import java.net.URI;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.datavec.api.writable.Text;

/**
 * JSON Path Label Generator
 *
 * Label generator for a given JPG image, extracted from a JSON file
 *
 * @author
 * @version 1.0
 */
public class JsonPathLabelGenerator implements PathLabelGenerator {

    String benign = "benign";
    String malignant = "malignant";

    /** Constructor */
    public JsonPathLabelGenerator() {}

    /**
     * Gets the label from JSON file for a given JPG image.
     * Note that the comments make testing and debugging the method much simpler.
     *
     * @param JpgPath The path of the JPG image
     * @return benign or malignant The status of the given input JPG image written in the JSON file
     * @return null Returns null if the method fails to create a JSON Reader object
     */
    @Override
    public Writable getLabelForPath(String JpgPath) {
//        System.out.println("jpg" + JpgPath);
        String JsonPath = fileExtensionRename(JpgPath, "json");
//        System.out.println("json" + JsonPath);
        try {
            JsonReader jsonReader = Json.createReader(new FileReader(JsonPath)); //Json path is absolute file path
            JsonObject json = jsonReader.readObject();
            if (!json.getJsonObject("meta").getJsonObject("clinical").isNull("benign_malignant")) {
                return new Text(json.getJsonObject("meta").getJsonObject("clinical").getString("benign_malignant"));
            } else {
//                System.out.println(json.getJsonObject("meta").getJsonObject("clinical").isNull("benign_malignant"));
                return null;
            }
        } catch (FileNotFoundException e){ e.printStackTrace();}
        catch (IOException e){ e.printStackTrace();}
        catch (Exception e){ e.printStackTrace();}
        return null;
    }

    /**
     * Gets the JSON path for a given file JPG image.
     *
     * @param uri The path of the JPG image
     * @return benign or malignant The status of the given input JPG image written in the JSON file
     * @return null Returns null if the method fails to create a JSON Reader object
     */
    @Override
    public Writable getLabelForPath(URI uri){
        return getLabelForPath(new File(uri).toString()); //remove getName() for absolute path
    }

    /**
     * Renames the file extension
     *
     * @param input The entire file name that needs extension change
     * @param newExtension The new extension
     * @return File with new extension
     */
    private static String fileExtensionRename(String input, String newExtension) {
        String oldExtension = getFileExtension(input);

        if (oldExtension.equals("")) {
            return input + "." + newExtension;
        } else {
            return input.replaceFirst(Pattern.quote("." + oldExtension) + "$", Matcher.quoteReplacement("." + newExten-
sion));
        }
    }

    /**
     * Gets the file extension.
     *
     * @param input File to get extension from
     */
    private static String getFileExtension(String input) {
        int i = input.lastIndexOf('.');

        if (i > 0 &&  i < input.length() - 1) {
             return input.substring(i + 1);
         } else {
             return "";
         }
     }

     /**
      * Only compatible with newer versions of deeplearning4j.
      */
//    @Override
//    public boolean inferLabelClasses(){
//        return true;
//    }
}

```

JsonImageRecordReader.java

```java
package com.research.skindetector;

import org.datavec.image.recordreader.BaseImageRecordReader;
import org.datavec.api.io.labels.PathLabelGenerator;
import org.datavec.api.split.FileSplit;
import org.datavec.api.split.InputSplit;
import org.datavec.api.util.files.FileFromPathIterator;
import org.datavec.image.loader.NativeImageLoader;

import java.io.*;
import java.net.URI;
import java.util.*;
import java.nio.file.Files;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * JSON Image Record Reader
 *
 * Takes in a dataset of JPG images with corresponding JSON files and labels JPG images from a value in the JSON file.
 *
 * @author
 * @version 1.0
 */
public class JsonImageRecordReader extends BaseImageRecordReader {

    /**
     * Constructor
     *
     * @param height The height of image in pixels
     * @param width The width of image in pixels
     * @param channels The number of channels (e.g. 1 for grayscaled and 3 for RGB)
     * @param labelGenerator Label (of either malignant or benign) for a given JPG image
     */
    public JsonImageRecordReader(int height, int width, int channels, PathLabelGenerator labelGenerator){
        super(height, width, channels, labelGenerator);
    }

    /**
     * Initializes the image record reader by transforming input images and labeling them with their corresponding
     * status (either malignant or benign) in a format compatible by the deeplearning4j library.
     *
     * NOTE: IF LABELS FROM JSON ARE NOT EITHER "MALIGNANT OR BENIGN", ALTHOUGH THE JPG WILL BE MOVED TO THE GARBAGECOLLECT
     * FOLDER YOU STILL MUST MANUALLY SEARCH FOR THE JSON IN EITHER THE TRAINING OR TESTING FOLDER
     *
     * @param split The file path for the dataset that should be initialized
     * @throws IOException
     */
    @Override
    public void initialize(InputSplit split) throws IOException {
        //transforms image to the given height and width for the given channel
        if (imageLoader == null) {
            imageLoader = new NativeImageLoader(height, width, channels, imageTransform);
        }

        inputSplit = split;
        URI[] locations = split.locations();
        if (locations != null && locations.length >= 1) {
            if (appendLabel && labelGenerator != null) {
//                if (appendLabel && labelGenerator != null && labelGenerator.inferLabelClasses()) {
                Set<String> labelsSet = new HashSet<>();
                for (URI location : locations) {
                    File imgFile = new File(location);

                    if(labelGenerator.getLabelForPath(location) != null){
                        if(labelGenerator.getLabelForPath(location).toString().equals("benign") || labelGenerator.getLabelForPath(location).toString().equals("malignant")) {
                            String name = labelGenerator.getLabelForPath(location).toString();
                            labelsSet.add(name);
                        }
                    } else {
                        File garbageCollect = new File("C:\\Users\\Desktop\\test\\garbageCollect\\");
                        if (!garbageCollect.exists()) {
                            garbageCollect.mkdir();
                        }
                        File tempjson = new File((fileExtensionRename(imgFile.toString(),"json")));
//                        System.out.println(imgFile.toPath());
//                        System.out.println(garbageCollect.toPath()  + "\\" + imgFile.toString().substring(imgFile.toString().lastIndexOf('\\')+1));
                        Files.move(imgFile.toPath(), new File(garbageCollect.toPath()  + "\\" + imgFile.toString().substring(imgFile.toString().lastIndexOf('\\')+1)).toPath());
//                        Files.move(tempjson.toPath(), new File(garbageCollect.toPath()  + "\\" + tempjson.toString().substring(tempjson.toString().lastIndexOf('\\')+1)).toPath());
                    }
                }
                labels.clear();
//                System.out.println("clear");
                labels.addAll(labelsSet);
//                System.out.println("addAll labelsSet");
            }
            iter = new FileFromPathIterator(inputSplit.locationsPathIterator()); //This handles randomization internally if necessary
//            System.out.println("Randomization");
        } else
            throw new IllegalArgumentException("No path locations found in the split.");

        if (split instanceof FileSplit) {
            //remove the root directory
//            System.out.println("Remove root directory?");
            FileSplit split1 = (FileSplit) split;
            labels.remove(split1.getRootDir());
        }

        //To ensure consistent order for label assignment (irrespective of file iteration order), sort the list of labels
        Collections.sort(labels);
//        System.out.println("Collection sorted");
    }

    /**
     * Renames the file extension
     *
     * @param input The entire file name that needs extension change
     * @param newExtension The new extension
     * @return File with new extension
     */
    private static String fileExtensionRename(String input, String newExtension) {
        String oldExtension = getFileExtension(input);

        if (oldExtension.equals("")) {
            return input + "." + newExtension;
        } else {
            return input.replaceFirst(Pattern.quote("." + oldExtension) + "$", Matcher.quoteReplacement("." + newEx-
tension));
        }
    }

    /**
     * Gets the file extension.
     *
     * @param input File to get extension from
     */
    private static String getFileExtension(String input) {
        int i = input.lastIndexOf('.');

        if (i > 0 &&  i < input.length() - 1) {
            return input.substring(i + 1);
        } else {
            return "";
        }
    }
}
```

DataDownloader.java

```
package com.research.skindetector;

import org.apache.commons.io.FilenameUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.research.skindetector.utilities.DataUtilities;

import java.io.File;
import java.io.IOException;

 /**
 * Inspired by a developer from deeplearning4j
 *
 * Downloads data from a given database
 *
 * Note that this class is not properly working because the entire ISIC_Dataset used in this study could not be saved
 * into a github directory due to the excessively large data space requirement
 */
 public class DataDownloader {
    private static Logger log = LoggerFactory.getLogger(Main.class);
    public static String DATA_URL = "http://github.com/ISIC_Dataset/raw/master/ISIC_Dataset.tar.gz";
    public static String DATA_PATH = FilenameUtils.concat(System.getProperty("java.io.tmpdir"), "ISIC_Da-
taset/");

    /**
     * Constructor
     */
    public DataDownloader(){}

    /**
     * Constructor
     *
     * @param DATA_URL
     * @param DATA_PATH
     */
    public DataDownloader(String DATA_URL, String DATA_PATH){
        this.DATA_URL = DATA_URL;
        this.DATA_PATH = DATA_PATH;
    }

    /**
     * Downloads the data
     *
     * @throws IOException
     */
    public void download() throws IOException {
        File directory = new File(DATA_PATH);

        if(!directory.exists()){
            directory.mkdir();
        }

        String filePath = "/ISIC_Dataset.tar.gz";
        String archivePath = DATA_PATH + filePath;
        File archiveFile = new File(archivePath);
        File extractedFile = new File(DATA_PATH + "ISIC_Dataset");

        if (!archiveFile.exists()){
            log.info("Downloading Database. Please wait...");

            String tmpDirStr = System.getProperty("java.io.tmpdir");
            String archizePath = DATA_PATH + filePath;

            if (tmpDirStr == null) {
                throw new IOException("System property 'java.io.tmpdir' does specify a tmp dir");
            }

            File f = new File(archizePath);
            if (!f.exists()) {
                DataUtilities.downloadFile(DATA_URL, archizePath);
                log.info("Data downloaded to ", archizePath);
            } else {
                log.info("Using existing directory at ", f.getAbsolutePath());
            }

            //Extract tar.gz file to output directory
            DataUtilities.extractTarGz(archivePath, DATA_PATH);
        } else {
            log.info("Data (.tar.gz file) already exists at {}", archiveFile.getAbsolutePath());
            if (!extractedFile.exists()) {
                //Extract tar.gz file to output directory
                DataUtilities.extractTarGz(archivePath, DATA_PATH);
            } else {
                log.info("Data (extracted) already exists at {}", extractedFile.getAbsolutePath());
            }
        }
    }

    public String getDataURL(){
        return DATA_URL;
    }

    public void setDataURL(String DATA_URL){
        this.DATA_URL = DATA_URL;
    }

    public String getDataPath(){
        return DATA_PATH;
    }

     public void setDataPath(String DATA_PATH){
         this.DATA_PATH = DATA_PATH;
     }
}
```

DataUtilities.java

```java
package com.research.skindetector.utilities;

import org.apache.commons.compress.archivers.tar.TarArchiveEntry;
import org.apache.commons.compress.archivers.tar.TarArchiveInputStream;
import org.apache.commons.compress.compressors.gzip.GzipCompressorInputStream;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
 import org.apache.http.impl.client.HttpClientBuilder;

 import java.io.*;

 /**
 * Common data utility functions.
 *
 * @author fvaleri (deeplearning4j developer)
 */
 public class DataUtilities {

    /**
     * Download a remote file if it doesn't exist.
     * @param remoteUrl URL of the remote file.
     * @param localPath Where to download the file.
     * @return True if and only if the file has been downloaded.
     * @throws Exception IO error.
     */
    public static boolean downloadFile(String remoteUrl, String localPath) throws IOException {
        boolean downloaded = false;
        if (remoteUrl == null || localPath == null)
            return downloaded;
        File file = new File(localPath);
        if (!file.exists()) {
            file.getParentFile().mkdirs();
            HttpClientBuilder builder = HttpClientBuilder.create();
            CloseableHttpClient client = builder.build();
            try (CloseableHttpResponse response = client.execute(new HttpGet(remoteUrl))) {
                HttpEntity entity = response.getEntity();
                if (entity != null) {
                    try (FileOutputStream outstream = new FileOutputStream(file)) {
                        entity.writeTo(outstream);
                        outstream.flush();
                        outstream.close();
                    }
                }
            }
            downloaded = true;
        }
        if (!file.exists())
            throw new IOException("File doesn't exist: " + localPath);
        return downloaded;
    }

    /**
     * Extract a "tar.gz" file into a local folder.
     * @param inputPath Input file path.
     * @param outputPath Output directory path.
     * @throws IOException IO error.
     */
    public static void extractTarGz(String inputPath, String outputPath) throws IOException {
        if (inputPath == null || outputPath == null)
            return;
        final int bufferSize = 4096;
        if (!outputPath.endsWith("" + File.separatorChar))
            outputPath = outputPath + File.separatorChar;
        try (TarArchiveInputStream tais = new TarArchiveInputStream(
                new GzipCompressorInputStream(new BufferedInputStream(new FileInputStream(inputPath))))) {
            TarArchiveEntry entry;
            while ((entry = (TarArchiveEntry) tais.getNextEntry()) != null) {
                if (entry.isDirectory()) {
                    new File(outputPath + entry.getName()).mkdirs();
                } else {
                    int count;
                    byte data[] = new byte[bufferSize];
                    FileOutputStream fos = new FileOutputStream(outputPath + entry.getName());
                    BufferedOutputStream dest = new BufferedOutputStream(fos, bufferSize);
                    while ((count = tais.read(data, 0, bufferSize)) != -1) {
                        dest.write(data, 0, count);
                    }
                    dest.close();
                }
            }
        }
    }
}
```