skinCAM: An Effective Solution Towards Skin Disease Diagnosis Via Novel Deep Learning Algorithm

Harris Beg
Canyon Crest Academy

William Sun
Canyon Crest Academy

Abstract

Globally, the access to basic health care is scarce, even for identifiable and apparent diseases such as skin anomalies. Because the ability of modern AI is so strong in pattern recognition, we decided to put our knowledge of machine learning to use in the development of a modular skin disease identification machine. The concept of a portable machine maximizes reliability of the neural network system as cross checking multiple areas of the body suggests a large sample size and reduces chances of outliers. This was done by housing a microcontroller and camera within a snug box for efficiency and durability. After training (through 10 epochs of 1000 iterations) and integrating the neural network into the microcontroller, the model produced an accuracy of 90.2% from 10 test sets. Each test ran unique disease images through the trained neural network to confirm a valid prediction. Our study demonstrates that an artificial intelligence approach can be utilized for an initial diagnosis, providing users with useful information about whether or not they should consult a dermatologist for further treatment. The technology not only saves time and cuts expenses, but it is also equally as important by providing a unique tool for dermatologists around the world.

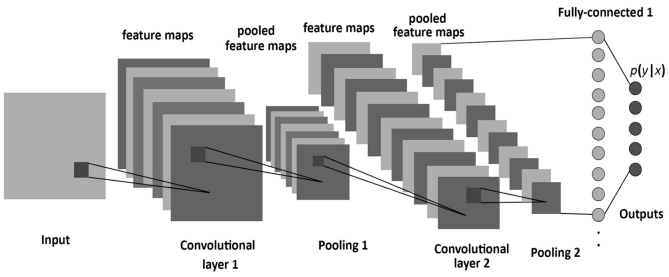
Acknowledgements

We would like to thank Mr. Mauro, Canyon Crest Academy's very own Computer Science teacher, for inspiring us to think beyond, and to pursue an idea beyond our grasp of knowledge. No matter what threw us off course throughout our development of skinCAM, Mr. Mauro allowed us to continue believing that we could achieve it, and that with this idea, skinCAM, could help people universally, regardless of where, when, or in which condition. With this passion, "skinCAM" was thus born. In addition, Mr. Mauro aided us in our process of applying for a provisional patent, a major step for the future of this product.

Introduction

Starting the 2019 Science & Engineering Fair season, we aspired to not only venture forth with an idea we loved, but one bigger than ourselves. With this, we knew that in an increasingly Artificial Intelligence - oriented world, major medical conditions affecting people across the globe could finally be reduced efficiently, easily, and tangibly by developers all over the world. Skin diseases, however, are not identifiable to most people. In fact, the core reason that skin diseases such as malignant melanoma (skin cancer) are so prevalent today is not solely because of its harmful abilities, rather because of its ability to grow exponentially over time - especially if unidentified. A personal experience has also prompted our interest in creating technology like this. Last year while going to a dermatologist to check out a unnatural looking mole, the dermatologist spent no more than a couple minutes on my case, and proceeded to charge hundreds of dollars without any treatment at all. With this novel application of machine learning, the accessibility and ease of access will be crucial to those who are trying to save money and time to check on a patch of skin. Using "skinCAM", a codename for our skin disease AI recognition program, this detection can be provided with the usage of a Convolutional Neural Network (CNN).

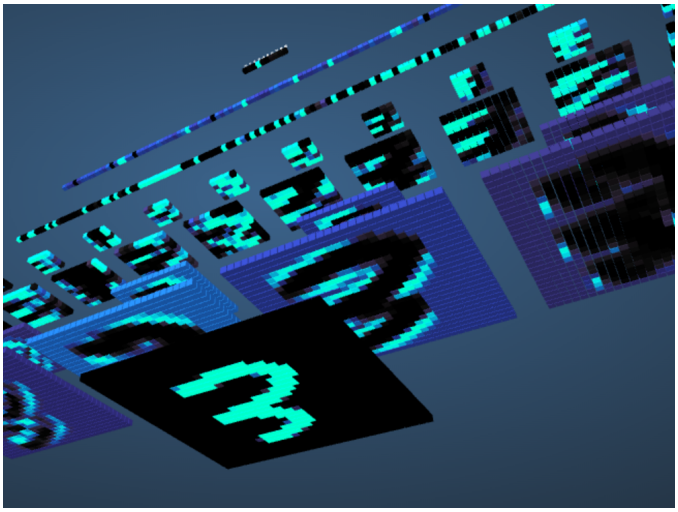
Background Information



The image above demonstrates the principle network behind the recognition algorithm used. It shows how the convolutional layer integrates with the feedforward cells, or Fully Connected Layer, at the very end of the network. Its architecture includes two convolutional layers.

- **Machine learning utilizes many functions of optimization to give computers the ability to learn without explicit code**
    - One type of neural network is the CNN, or convolutional neural network. It is much more sophisticated than a vanilla feed forward neural network
- **A feed forward network is the most basic machine learning algorithm that exploits mathematical neurons, coined perceptrons, that learn through the reduction of the error gradient with each training iteration**
    - Thus, they are able to learn and read the patterns of the data set with which it trains on
    - The computational ability of a CNN excels with large data sets, and allows for a smaller number of perceptrons, largely reducing the time it would take to train with the same accuracy
- **CNNs have revolutionized the capability of image recognition:**
    - They function by housing a convolutional layer in front of a vanilla feed forward neural network
    - The layer's purpose is to highlight particular features of an image, such as distinct color or contour patterns with mathematical filters called kernels

- The filter drastically increases the accuracy of the program

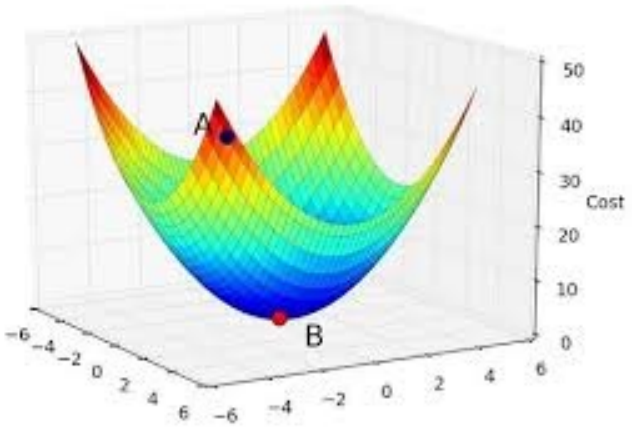


- **The CNN algorithm can be applied to the image recognition of skin diseases:**
  - Since each disease has its own unique characteristic, the filter of the convolutional layer in the code is exceptional at learning these features
  - For example, the network is able to distinguish between acne and moles with more precision because the convolutional layer allows it to differentiate between the red speckled feature of acne and the spherical dark pigment of a mole
- **A CNN works by moving a filter through the image:**
  - The pooling layer takes the matrix produced by the filter and decreases its size, thus rendering out blank areas and highlighting disease areas
- **For these purposes, the machine learning library, TensorFlow, was used**
- **Research for skin diseases:**
  - Remedies, Common Symptoms, and Identification were studied extensively
- **Database was created through sources from the internet:**
  - Hand selected images from the American Academy of Dermatology, webMD, and DermWeb
- **The following diseases were trained:**
  - Acne, Carcinoma, Chicken Pox, Eczema, Hives, Melanoma, Psoriasis, Rosacea, Warts
  - Control: Healthy skin

Softmax Regression Function used on logits as the activation function	$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp(s_j(\mathbf{x}))}$
Cost Function for Neural Network	$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}(1-y^{(i)})\log(1-h_\theta(x^{(i)})) + y^{(i)}\log h_\theta(x^{(i)})\right]$ $= -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=0}^{1} 1\{y^{(i)}=j\}\log p(y^{(i)}=j\|x^{(i)};\theta)\right]$
Adam Optimizer Algorithm we used in minimizing our Cost Function, $J(\theta)$	$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$

Softmax Regression Function used on logits as the activation function	$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp(s_j(\mathbf{x}))}$
Cost Function for Neural Network	$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}(1-y^{(i)})\log(1-h_\theta(x^{(i)})) + y^{(i)}\log h_\theta(x^{(i)})\right]$ $= -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=0}^{1} 1\{y^{(i)}=j\}\log p(y^{(i)}=j\|x^{(i)};\theta)\right]$
Adam Optimizer Algorithm we used in minimizing our Cost Function, $J(\theta)$	$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$

The above table indicates the common formulas used in a machine learning code. They are used for the training step in the program, which reduces the error that the algorithm produces. It does this by creating an optimization function that reduces the error of the cost function, which is calculated in the second formula. Then, the Adam Optimizer updates the weights of the program by taking the partial derivative of the cost function and running it through an optimization, thus lowering the cost through each iteration of training. The following image depicts the cost function decreasing over time (iterations).



Convolutional Neural Network Research:

As Stanford University's machine learning course puts it: "a typical Conv Layer: Accepts a volume of size W1×H1×D1." A standard convolutional requires four hyperparameters, which basically defines the initialized variables that the program will use. This includes the number of filters K, their spatial extent F, the stride S, and the amount of zero padding P. The common setting for these values is: F=3, S=1, and P=1.What each of these hyper-parameters defines is ultimately important for the efficiency of the neural network. The number of filters and the spatial extent that the convolutional is defined to have creates the initialized program to run through the image. What the stride does is that it moves the filter across the input image by the amount of pixels that it is defined to be. This is important for the convolutional layer to work because the filter itself is usually predefined to be 3 x 3 pixels, much smaller than the actual image, so the stride moves the filter along the image to create a full feature map of the highlights. In a training setting, the filter which is looking to highlight the particular sections of the image, will pass along each section of the image, creating a matrix-multiplication problem for the computer to work out. On the areas that are blank, with no correlation to what the filter is looking at, the output of the matrix-multiplier will be close to 0. On the other hand, when the filter passes by the area of the image that is filled with the areas of disease, the filter will multiply out a large value. What the convolutional layer does is produce a feature map that outputs the highlighted regions of the image that outputted a large value. Stanford University's CS231n class summarizes the output as: "W2×H2×D2 where: W2=(W1−F+2P)/S+1, H2=(H1−F+2P)/S+1 (i.e. width and height are computed equally by symmetry), D2=K."

Next, the CNN works by passing the feature map through an activation function to create proper weights and bias values for back-propagation. For the CNN, ReLU (Rectified Linear Unit) was used. The benefit of using ReLU in place of sigmoid is in its avoidance of vanishing gradients and non-linearity which is introduced.

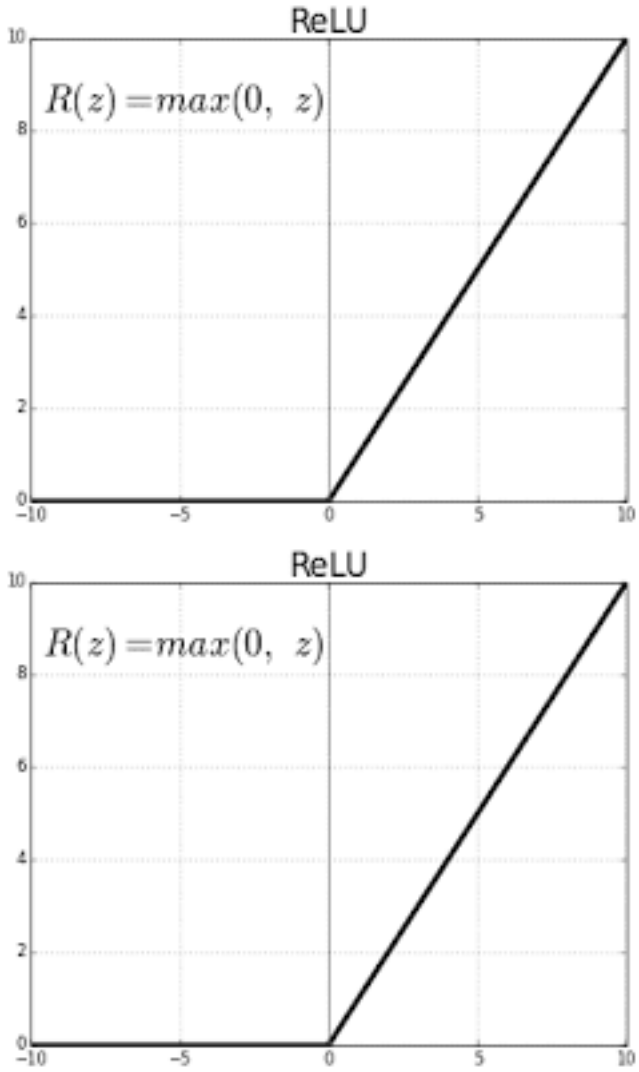
Finally, the pooling layer comes into play for the convolutional neural network. Since much of the photo image is blank space without the disease that the filter is looking for, it would be much more efficient and accurate to delete the unnecessary areas. The pooling layer does this by removing the feature maps that outputted low values and only keeping the big areas. The size is cut down directly by the amount that is specified in the code. Typically in a standard CNN architecture, the pooling layer will reduce the image size by 1/2. The pooled feature maps are converted into a matrix of values that can be read by the computer. The matrix of values is actually just individual pixel values.

Basic Deep Network versus Convolutional Network:

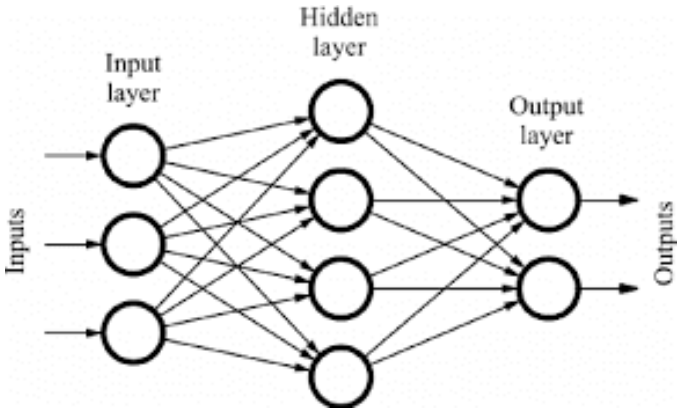
Based upon well defined research of MNIST datasets, the average accuracy for a basic neural network can reach 92 percent after 1000 iterations. The convolutional neural network however, can easily reach an accuracy of over 99.1% with the same criteria. The most advanced convolutional neural network architecture linked on GitHub uses dropout functions in addition to the already powerful technology in order to reach 99.7% accuracy. The MNIST dataset consists of 50000 training images and 10000 testing images. The MNIST is the standard for convolutional visual recognition research because of its simplicity. The images are simple 28x28 .jpg files of numbers from 0-9.

Feed Forward Neural Network Information:

The feed forward, or fully connected, layer in this network is created by a series of matrix multiplications. An ANN constitutes of weights and perceptrons. The weights of an artificial neural network are what the ML algorithm is able to alter, giving the ability to learn. And perceptrons are a mathematical unit, that introduces non-linearity though an activation function. Mathematically, this computation is represented by o = ReLU(WX').



When the fully connected is directed towards optimization, it uses back-propagation for updating weights, and gradient descent for traveling down the error gradient. The algorithm utilizes the Adam Optimizer in gradient descent. There was major discrepancy between using Adam Optimizer over RMS Prop, another optimization algorithm.



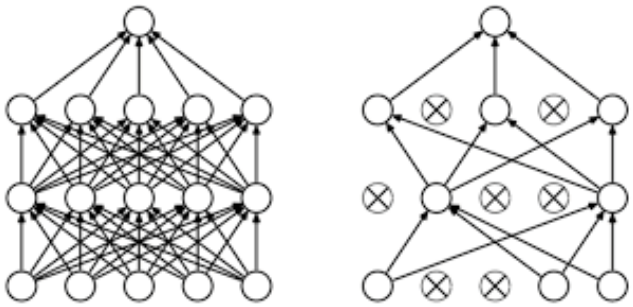
$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

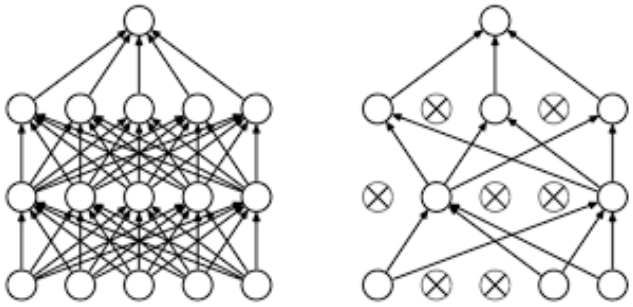
$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

Standard Procedure:

Overall, the convolutional neural network can be described with this basic procedure:
Step 1: We initialize all filters and parameters / weights with random values
Step 2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class. Let's say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]. Since weights are randomly assigned for the first training example, output probabilities are also random.
Step 3: Calculate the total error at the output layer (summation over all 4 classes)
Total Error = ∑ ½ (target probability – output probability) ²
Step 4: Use back-propagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error. The weights are adjusted in proportion to their contribution to the total error. When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0]. This means that the network has learnt to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced. Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.
Step 5: Repeat steps 2-4 with all images in the training set.

Dropout:


(a) Standard Neural Net (b) After applying dropout.


(a) Standard Neural Net (b) After applying dropout.

Our code also takes advantage of the tool dropout. This function negates chances of overfitting with the training. What overfitting does, is that during gradient descent, the code may overshoot the global or local minimum, thus failing to find the correct, accurate location. The developers of neural networks found out a way to reduce the chances of overfitting. The first would be to decrease the train_step. Train_step is the size at which gradient descent changes each time. If the value was extremely small like 1.0E-10, the code would take too long to learn as it changes too little each iteration. On the other hand, with a train step of 1.0, the optimization value could be moving too much after each iteration and completely miss the minimum. By manually testing for the best train_step could work for the code, but a better method is to use dropout: The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. Essentially, removing random weights on the feature map for the convolutional neural network will force the network to rediscover certain minimums, creating a vastly more accurate system. This however, does increase computation time.

Disease Research & Database:

Our diseases were selected from the American Academy of Dermatology (AAD): we used the top ten most common and prevalent diseases based on the information given. With each disease, we listed out the major symptoms and disease recognition factors that made each unique. Thus, a question guide was created in order to better understand each disease.

Collecting the actual disease was done utilizing a web-scraper. The web-scraper accessed the HTML sources of each website from webMD, DermWeb, and the ADA. Each of these housing hundreds of thousands of images. Our code was written to download all of the .jpeg and .png files that the photo atlases stored, and storing them into a folder.

Acne	3/11/2018 6:49 PM	File folder
Carcinoma	2/27/2018 8:27 PM	File folder
Chicken Pox	2/27/2018 8:27 PM	File folder
Eczema	2/27/2018 8:27 PM	File folder
Hives	2/27/2018 8:27 PM	File folder
Melanoma	2/27/2018 8:27 PM	File folder
Psoriasis	2/27/2018 8:27 PM	File folder
Rosacea	2/27/2018 8:27 PM	File folder
Warts	2/27/2018 8:27 PM	File folder

App Research:

Image uploaded → Cropped → Sent to python script through a REST service detecting new data entries → Python script uploads image onto local storage → Machine learning model fetches dir to analyze and prints array of values → App fetches values and lists them into the "Other Results" page and picks the highest probability disease to appear right after that someone takes the picture. Firebase is more of an "intermediate" in the whole process.

The app UI was created in Adobe Illustrator and then implemented onto the Android Studio program for creating the app itself.

A user friendly and inviting setting must be made for the user interface, so we researched the best colors for this specific case. What we came up with was that a simple, futuristic, and calming color scheme was the most advantageous. Because the user is most likely frantic about the irregularity on their body, a calming mood will help them cope with whatever information is about to proceed.

Statement of Problem or Purpose

There are more than 300 skin diseases known to us today, and zero consumer-grade applications to prevent them from spreading. The purpose of this project is to provide a reliable consumer grade application utilizing machine learning and a mobile interface to relay real-time information about a consumer's skin abnormalities.

• To create an effective and accurate skin diagnostic program with machine learning algorithms in a novel application of convolutional neural network technology
• Design Criteria includes:
- Efficient transferrable .ckpt file for testing of the program
- Creation of application to run the program on mobile
• Use of materials:
- TensorFlow machine learning library
- Python
- Cloud services (Firebase)
- Database of diseases verified through viable online sources

Hypothesis

Our hypothesis was based on the following constructs and functionality features:

- A neural network trained on accurate image data of skin conditions will be able to predict the disease of a new image
- A transferrable .ckpt file that stores the trained weights and biases from the convolutional neural network will be able to be accessed in a mobile setting
    - Training data can be reused for further training
- Identification can be expressed quantitatively through the percent accuracy of the program
    - Based off of the accuracy of the code and the program
    - Use of tactile representation and user-input will increase accuracy and dependability of the program

Materials

- TensorFlow Machine Learning Library
- Web-Scraping Bot (with Python)
- Raspberry Pi
- 16x2 I2C Display
- Raspberry PiCamera
- Female to Female Raspberry GPIO wires
- Database of diseases verified through viable online sources

Training Setup: 8-core CPU AMD 4.0 GHZ, 8 GB NVIDIA GTX 1080 GPU, 20 GB of RAM; Floydhub, AWS, and Microsoft Azure

Implemented Procedure

1. Compile the database from the online and source it with viable websites such as AAD, DermWeb, and WebMD.

2. Add images of each disease into a folder as a training set, 4000 images in total.

3. Begin the build of the program with the first block of code representing the import of the data training set and any libraries that must be used like Tensorflow. The calling of the import method will be built in step 7.

4. Second block of code holds the setup of the functions and variables used throughout the program.

5. Third block of code is the convolutional neural network layer. The CNN needs a convolutional layer, a ReLU layer, and a Pooling layer for each section. In total, we coded two convolutional layers. See below:

```
def convolutional(input_image, Kernels, use_pooling=True):

    image_size = 100
    convolution = tf.reshape(input_image, [-1, image_size, image_size, 3])

    # ========== create kernels =============

    kernel_1 = Kernels["kernel_1"]
    kernel_1_bias = Kernels["kernel_1_bias"]

    kernel_2 = Kernels["kernel_2"]
    kernel_2_bias = Kernels["kernel_2_bias"]

    # ========== first convolutional layer ===========================================================

    convolution = tf.nn.conv2d(input=convolution, filter=kernel_1, strides=[1, 1, 1, 1], padding="SAME")
    convolution = convolution + kernel_1_bias
    convolution = tf.nn.relu(convolution)

    if use_pooling:
        convolution = tf.nn.max_pool(value=convolution, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")

    #tf.summary.image("convolutional 1", convolution, 3)

    # ========== second convolutional layer ==========================================================

    convolution = tf.nn.conv2d(input=convolution, filter=kernel_2, strides=[1, 1, 1, 1], padding="SAME")
    convolution = convolution + kernel_2_bias
    convolution = tf.nn.relu(convolution)

    if use_pooling:
        convolution = tf.nn.max_pool(value=convolution, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")

    return convolution
```

6. The fourth block of code is the training step. tf.matmul is used frequently in our architecture at points where the input of the CNN is multiplied with training weights and biases. Then, after the matmul functions were written out for each perceptron, the back-propagation process was coded. This was done by finding cost, or the loss (tf.nn.softmax_cross_entropy_with_logits()). Training the program utilizes minimizing the cost, which was coded with the function tf.train.AdamOptimizer(learning_rate).minimize(cost). Through each training iteration, the back-propagation updates the weights and biases for the matmul operations.Our procedure calls for the code to run through 1000 iterations with a learning rate of 0.01.

```
with tf.name_scope("cost"):
    cost = tf.reduce_sum(tf.nn.softmax_cross_entropy_with_logits(labels=y_labels(index=Y, classes=num_classes), logits=fully_connected_layer))
    tf.summary.scalar("cost", cost)
    train_step = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

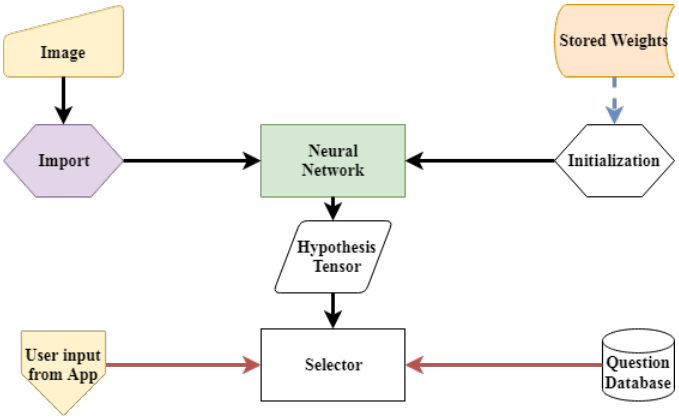
7. The final two blocks of code are the import method and the output section. Our input function was written to operate by taking in jpeg files, which holds 3 channels, RGB, the standard color model. The program locates the parent directory of the image database. Proceeding this, the image is transformed into a tensor (a matrix) Our code uses tf.image.resize_image_with_crop_or_pad(image input, height, width)

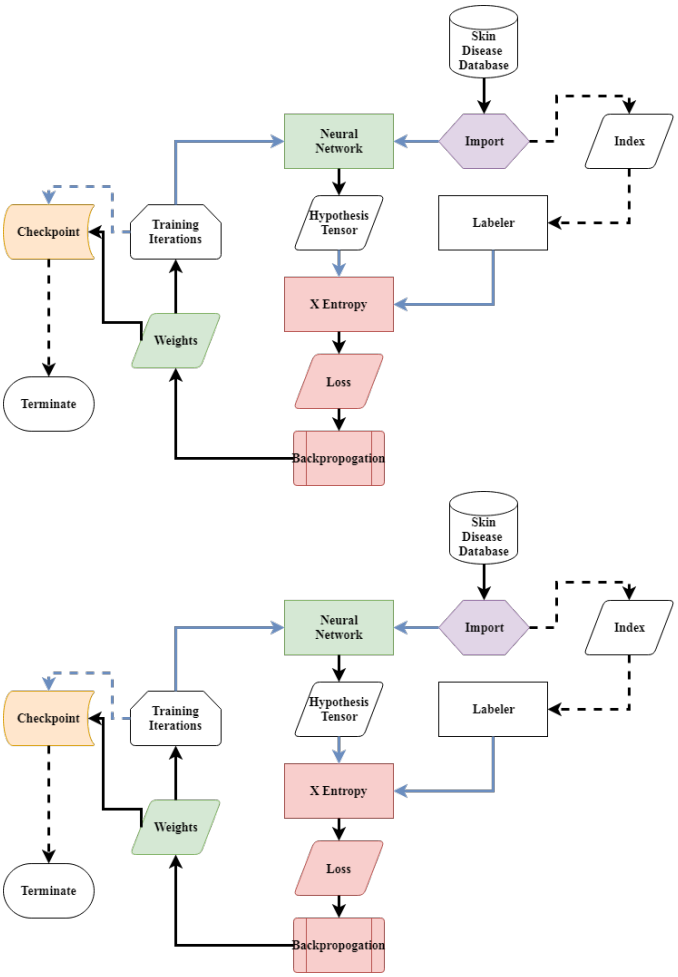
```
for iteration in tqdm(range(training_iterations), desc="COMPLETION", ncols=80):
    i = 0
    for myList in doublList:
#         print("myList:\n", myList, end="\n")
        for file in myList:
#             print("file:\n", file, end="\n")
            image_tensor = sess.run(import_image(file))
            summary, _ = sess.run([merged, train_step], feed_dict={X:image_tensor, Y:i})
        i = i + 1
    writer.add_summary(summary)
    SAVER.save(sess, SAVE_PATH)

print("\n\nTraining completed.\nRunning test prediction.\n")

while 1:
    DIRECTORY = input(r"Directory: ")
    test_input = sess.run(import_image(DIRECTORY))
    hypo = sess.run(fully_connected_layer, feed_dict={X:test_input})

    prediction = sess.run(tf.nn.softmax(hypo))
    argmax = sess.run(tf.argmax(prediction))
    print(argmax.tolist().index(max(argmax)))
    print("\n", prediction)
```

8. Run the neural network on a Python IDE.

9. Debug any errors, using Stack Overflow as a resource to solve the red compiling issues that the IDE outputs.

skinCAM Functional Flow





Training Program Specification

1. Import function of database sends the images through 12 layers. Each layer can include a convolution, a ReLU, a pool, and a dropout
2. Fully connected layer takes the summation of tensors (prediction) and compares it to the index (correct value) to produce a loss value
3. Learning progresses with backpropagation by minimizing the loss value through gradient descent
4. System repeats after each iteration and adds trained weights to a checkpoint file

Distributable Program Specification

1. A separate code takes the trained weights from the convolutional neural network and runs a user image through the program
2. A softmax function takes the matrix multiplication output and generates a value between 0-1 (prediction)
3. As a method of validating results and increasing accuracy, the user is given a set of quick tactile-based questions
4. The program implements the users' answers to maximize validity

Mobile Application Specification

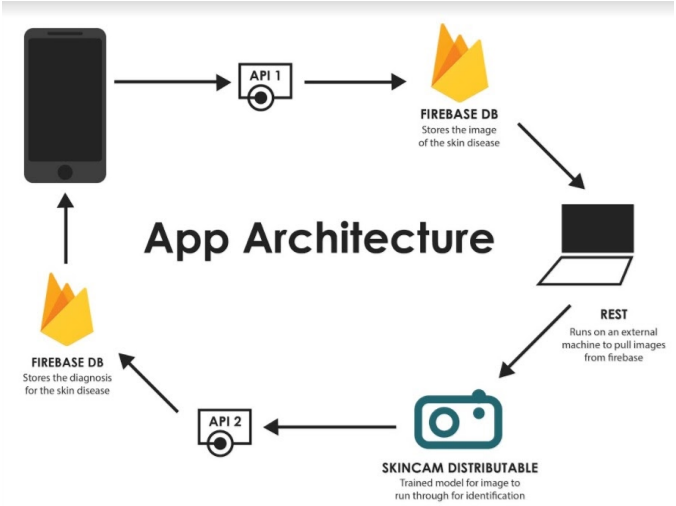


Image uploaded → Cropped → Sent to python script through a REST service detecting new data entries → Python script uploads image onto local storage → Machine learning model fetches dir to analyze and prints array of values → App fetches values and lists them into the "Other Results" page and picks the highest probability disease to appear right after that someone takes the picture. Firebase is more of an "intermediate" in the whole process.

Hardware Specification

The Raspberry Pi takes in a pre-trained model that is compiled into a .ckpt file. Upon interpreting this pre-trained model, the Pi (upon boot) will continuously take pictures at 10 second intervals given input from the camera. There was no need for a database/backend in this case, due to the localized experimentation on the Pi.

Development Progress

Initial Feed Forward Network

We began the coding process by testing image recognition capabilities with a standard feed forward neural network. The feed forward network utilized five hidden layers to train the images. Each layer used tf.nn.matmul in order to combine and summate all weights with the input data for a perceptron. The tf.nn.relu function was also processed in each layer for reducing errors.

The resulting accuracy of the feed forward neural network was extremely unbalanced. The undisputed fact that predictions were different during each testing period proved that training did not render as successful.

Several issues were deduced based upon the progress of the scientific process. Firstly, the computational power of the basic network was not enough for an acceptable product. This could be fixed by implementing technology from convolutional networks: based upon research from the AI community, convolutional neural networks drastically improve visual learning. Another issue with the program was that there was a distinct possibility for the database to be flawed. The cropping tool for the program was not initialized to the preferred area, which could have created errors in the images. For example, the code was programmed to crop all database images into a 100 x 100 .jpg file. The 100 x 100 section would most likely be taken out from the top-left corner based on standards of coding. In our case, the disease might not have been located around the top-left corner, thus decreasing training accuracy. If the neural network was training on the flawed data, accuracy

would have been reduced. Our solution to fix this issue was to crop each of the images in the center, an area where all images included the disease.

Second Iteration of Feed Forward Network

The second version of our neural network utilized 10 hidden layers to train the program. We did this to test the validity of the structure in neural networks. Our hypothesis was that if more layers were added onto a program, the accuracy would increase. Sure enough, the accuracy of the program was higher after training in the same number of iterations, but the training time was significantly greater. For the initial program, it could be trained in less than a day for 1000 images, but then the new network was exponentially greater in training time: it took almost a week for 1000 images.

First Iteration of Convolutional Neural Network

At this point, we decided to test the convolutional neural network theory. After we had created two convolutional layers with pooling and ReLU, we tested to see if the network was more accurate. Indeed, after 100 iterations, the initial network only had 20% accuracy, while the new convolutional neural network reached 40% accuracy. On the other hand, we still stuck with our original code with the 5 feed forward layers. Later we realized this was entirely unnecessary and a large waste of time. The program was more efficient than the 10 layer feed forward network and took less time to train, so it was proven that for future endeavors the convolutional neural network was the way to go. This backs up the research that we did because the convolutional neural network is clearly superior in image recognition than the standard deep net.

Second Iteration of Convolutional Neural Network

Next, we decided to recreate the convolutional neural network to increase efficiency in the code. We increased the number of convolutional layers and decreased the number of feed forward layers to see if that would benefit the cause. The production turned out to be quite surprisingly good. Initially, the global minimum was reached within 10 iterations of training. What we used was five convolutional layers and 1 feed forward layer, closer to the Tensorflow model for Convnets. This seemed too good to be true, because it is nearly impossible to find a global minimum within 10 iterations. After testing the program, we realized that the code was training on black images, which proved that the error was in the database. Most likely, the cropping software developed was incorrect, so the images were terrible to train on. To fix this issue, we changed the cropping software to crop the images separately and input those into the neural network. After fixing these issues and training the program for about a week, we used this program for the Greater San Diego Science and Engineering Fair.

Third Iteration of the Convolutional Neural Network

We decided to go even further and increase accuracy with a more powerful tool. By utilizing an architecture of 12 convolutional neural networks, our accuracy increased significantly. The number of iterations we had to train on decreased immensely by implementing more layers. An issue however, was that by increasing the number of layers, pooling occurred too many time and decreased the size of the images by too much, so we had to reduce pooling layers. After testing the new program, conclusively the program overshot at a high rate, so we needed to retrain the program. Fixing this error was simple: our train_step was set at 0.01 which was too high, so we changed it to 0.00000001. We determined that that was the most optimal train_step.

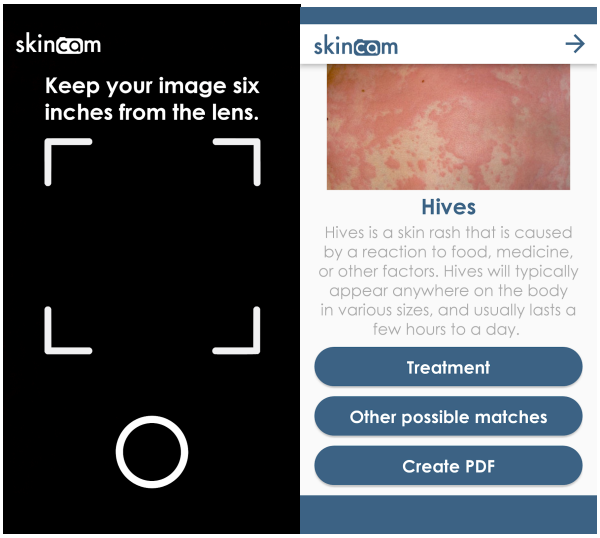
Current Neural Network Utilized

Since the previous model performed to about where we wanted (95.2%), we wanted to modify the code to make it more efficient. First, we added the dropout function to reduce overfitting errors, which allowed us to increase train_step to 0.000001. This decreased the number of iterations it took to train the program. Next, we redefined calculation functions so that the code could flow better and over many iterations save time. Finally, we decided to add a weight storage system. We did this to help the computer in its computations. During testing, a clear trend that after each iteration, more time was taken to complete calculations, so by stopping the program and restarting it after awhile, we could save a lot of time. To do this, saving the weights and rebooting them after 10 iterations allowed the computer to rest and calculate faster.

App Development

When starting initial app development for skinCAM, it was concluded that we should create an Android app, as it was most easily integrable with our TensorFlow machine learning script. With this in mind, we built a functional, but basic and unappealing layout initially.

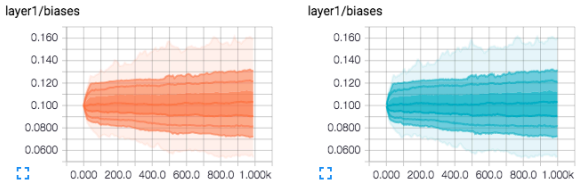
Following our success at the Greater San Diego Science and Engineering Fair, though, a particular interest in our prototype design was noted, which we have now fully adapted to. This new design contains a live camera preview as well as proper cropper that can yield the relay of real information obtained from the patent-pending system that reads in the image from the phone and outputs the machine-learning based disease probabilities. Our new design is depicted below:



Hardware Development

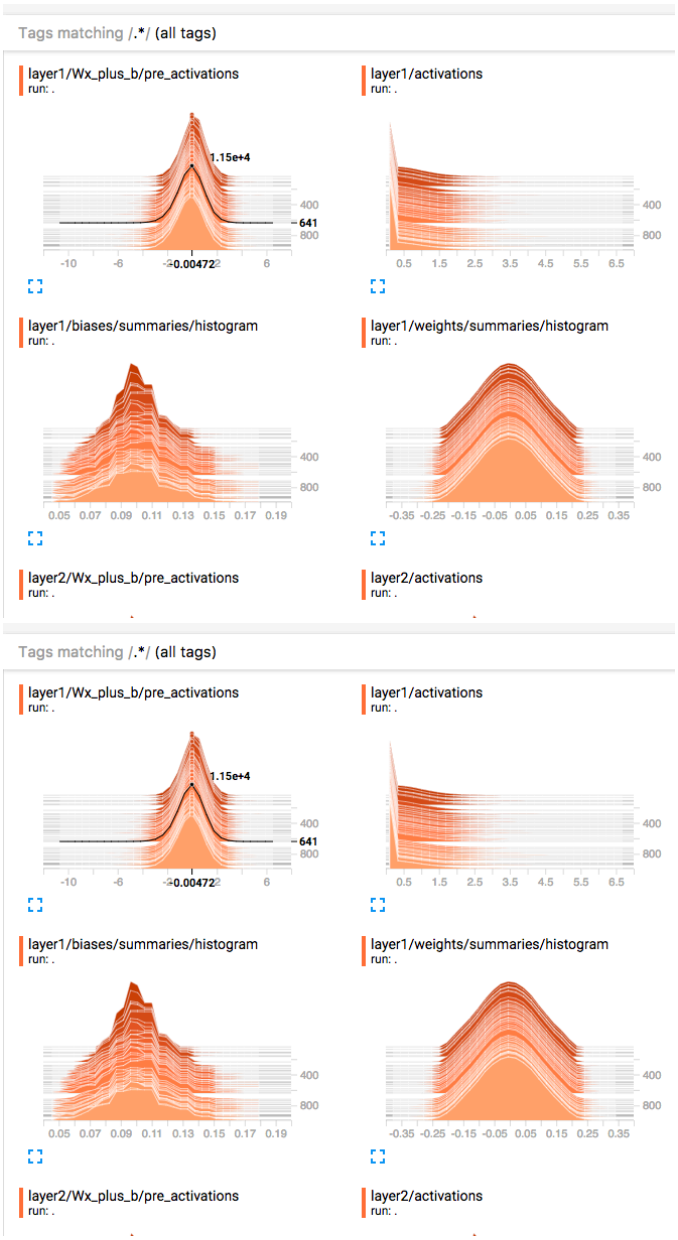
The Raspberry Pi we used for this project was a Raspberry Pi 3, due to the ease of connection between the Pi and Python scripts. This script interacts with the backend model of TensorFlow and translates the inputs of the Pi into the outputs that are viewable with the 16x2 display and, if required, a monitor.

Data and Data Analysis

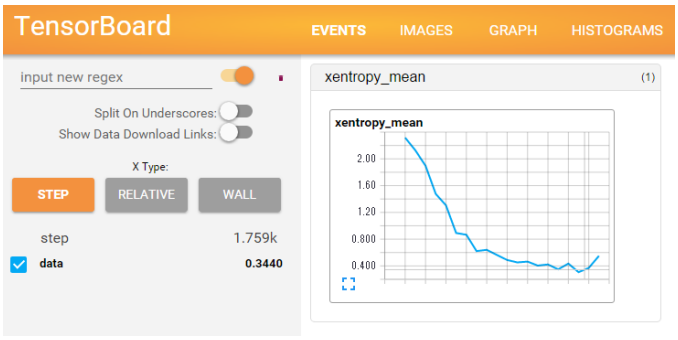


The distribution graph illustrates the gradual change of the weights within the neural network. The slow growth and attenuation of these weights represent the gradual process of learning. In this case the learning is represented by numerical figures to thirty-two point values of a decimal (float value). It's

through these algorithms that the average user can hope to save usually expensive procedures and offer a rather less costly means of diagnosing potentially dangerous diseases. These evolutionary standards represent not only the growth of the networks correlative ability, but also the possible growth of an entirely new industry.



The histograms above are essentially another representation of the weights as the number of iterations progress, a different representation of the Distribution graph. At the very back, we have the oldest weights. By looking at the entire graph, you are able to observe the gradual evolution of the weights within the networks. The evolution proves that we are able to properly train the network. The trend demonstrates that the research we have conducted. We have conducted an acceptable and successful basepoint as an image recognizer. We can also see how slow the weights evolve during the iterative process, thus detailing the need for a more comprehensive system with the greater capabilities and computational power. A drawback is that the training time is more inefficient compared to leading machine learning algorithms under tech giants such as Google and Facebook. This shows the need for vastly improving our systems needs in the future.



The x-entropy graph is a graphical representation of our loss with respect to each progressive iteration of training. The general trend appears to model a negative exponential trend. This visual represents a proper trend as the error should be decreasing with the progression of time. The process of this decreasing cross entropy proves that there is a rise in the network's ability to correlate the images with their appropriate labels. In addition it proves that the functions used such as tf.nn.matmul() and tf.nn.relu(), are, among with backpropagation working together to create a cohesive system that reduces error.

Findings & Results

Training Iterations	Accuracy
100	0.32
200	0.45
300	0.49
400	0.57
500	0.63
600	0.70
700	0.78
800	0.85
900	0.89
1000	0.90

The learning rate was set to 0.0001, with 1000 iterations, and the program was trained on approximately 5000 images. The criteria with which the accuracy was calculated for the novel application of CNNs was the number of times the program would output the correct disease. It was tested 1000 times, with 10 distinct testing images for each disease. As shown, the accuracy after the first 100 iterations was 40.1%, while at 1000 iterations it was able to reach 95.2%. We gathered the accuracy at each of these checkpoints by testing the saved weights from the training at each landmark iteration (i.e. 100, 200…). Then taking the output of the network and running it through a series of questions to verify the findings solidified our research even further.

The data is statistically acceptable because an accuracy of 95.2% after training is within the design criteria error (10%). After 1000 iterations of training, only 48 out of 1000 predictions of the program were wrong. Three of the errors where in Rosacea, which indicates a high visual similarity between the diseases Rosacea and Acne. The other two errors were in chicken pox, which

is also similar to Acne and Rosacea, thus the database must be expanded in order to create a larger distinction between the different diseases. The results also show that the accuracy could still be increased drastically. With just 1000 iterations and a relatively small database, 95.2% accuracy could be easily improved, along with the efficiency of the code.

The system used for training can be improved as well. An 8-core AMD CPU at 4.0 ghz was used to train the program, along with a NVIDIA GTX 1080. With a multiple GPU system or a cloud-based computing system, the train time can be reduced. Overall, a successful model of the novel neural network was created as our findings show that it is useful and achievable to produce a skin disease recognition program with machine learning applications.

Conclusion

The purpose of this project was to create an accurate and efficient convolutional neural network to recognize various skin diseases. By using the powerful technology of convolutional networks, a trustworthy image recognition program was developed. The design criteria was met because the software generated an accuracy of 95.2%, well within the accepted range of 10% error, thus deeming this project's hypothesis as correct. The data collected proved that a machine learning algorithm is capable of distinguishing between different skin diseases. As the program continued to train on thousands of skin images (the independent variable), the error of the convolutional network decreased until it was 95% accurate (the dependent variable). This shows a correlation between the size of the data set and the accuracy of the software, which means that with a large enough set of training data, the model should reach industry levels of accuracy. Sources of error in this project were because of the inaccuracy of the image database. The data included areas of hair and other distractions, which could have altered the training correlations. For example, since color was a built filter in the convolutional neural network, the darker patches of hair could have changed the accuracy of the program. New understanding of the success at which neural networks learn to recognize images, can be applied to widespread fields that will revolutionize the world for the next generation. In addition, the ever-growing power of computing will come to aid the mathematical models to reach great depths of discovery.

Recommendations

- XSEDE:
    - More powerful training technology utilizing multi-faceted Tesla P100 GPU's
    - Ability to train on more costly neural network architectures such as: Inception V3, ResNet
- Reach out to local hospitals and dermatologists:
    - Extensive growth of the database, providing for more accurate diagnosis and training
    - Allows for continuous testing of the program
- Mass production of Raspberry Pi-based device
- Cost reduction to effectively mass produce the device

References

AAAI Digital Library, aaai.org/Library/library.php

"Acne." Acne | American Academy of Dermatology, www.aad.org/public/diseases/acne-and-rosacea/acne

"Android - Camera API" https://developer.android.com/guide/topics/media/camera.html

Artificial Intelligence: A Modern Approach. Pearson Education Limited, 2013

"Build Software Better, Together." GitHub, github.com/

"Chickenpox | Varicella | MedlinePlus." MedlinePlus Trusted Health Information for You, medlineplus.gov/chickenpox.html

"Common Moles, Dysplastic Nevi, and Risk of Melanoma." National Cancer Institute, www.cancer.gov/types/skin/moles-fact-sheet

CS231n Convolutional Neural Networks for Visual Recognition, cs231n.github.io/convolutional-networks/#conv

DermWeb, www.dermweb.com/photo_atlas/.

Michalski, Ryszard S., et al. Machine Learning: An Artificial Intelligence Approach. Tioga Pub. Co., 1983

"Moles." Mayo Clinic, Mayo Foundation for Medical Education and Research, 9 Jan. 2018, www.mayoclinic.org/diseases-conditions/moles/symptoms-causes/syc-20375200

"National Institutes of Health." National Institutes of Health, U.S. Department of Health and Human Services, www.nih.gov/

"Psoriasis." About Psoriasis, www.psoriasis.org/about-psoriasis

"Skin Cancer Foundation." Melanoma - SkinCancer.org, www.skincancer.org/skin-cancer-information/melanoma

"Skin Health and Skin Diseases | NIH MedlinePlus the Magazine." Skin Diseases, medlineplus.gov/magazine/issues/fall08/articles/fall08pg22-25.html

"Tutorials | TensorFlow." TensorFlow, www.tensorflow.org/tutorials/

"WRITING A SCIENTIFIC RESEARCH ARTICLE", www.columbia.edu/cu/biology/ug/research/paper.html

Jain, Sarthak. "How to Easily Detect Objects with Deep Learning on Raspberry Pi." Medium.com, Medium, 20 Mar. 2018, medium.com/nanonets/how-to-easily-detect-objects-with-deep-learning-on-raspberrypi-225f29635c74.

"TensorFlow Image Recognition on a Raspberry Pi." Silicon Valley Data Science, 19 Sept. 2017, www.svds.com/tensorflow-image-recognition-raspberry-pi/.

Rotate Display 90º? - Raspberry Pi Forums, www.raspberrypi.org/documentation/.

Artificial Intelligence: A Modern Approach. Pearson Education Limited, 2013

"Build Software Better, Together." GitHub, github.com/

"Chickenpox | Varicella | MedlinePlus." MedlinePlus Trusted Health Information for You, medlineplus.gov/chickenpox.html

"Common Moles, Dysplastic Nevi, and Risk of Melanoma." National Cancer Institute, www.cancer.gov/types/skin/moles-fact-sheet

CS231n Convolutional Neural Networks for Visual Recognition, cs231n.github.io/convolutional-networks/#conv

DermWeb, www.dermweb.com/photo_atlas/.

Michalski, Ryszard S., et al. Machine Learning: An Artificial Intelligence Approach. Tioga Pub. Co., 1983

"Moles." Mayo Clinic, Mayo Foundation for Medical Education and Research, 9 Jan. 2018, www.mayoclinic.org/diseases-conditions/moles/symptoms-causes/syc-20375200