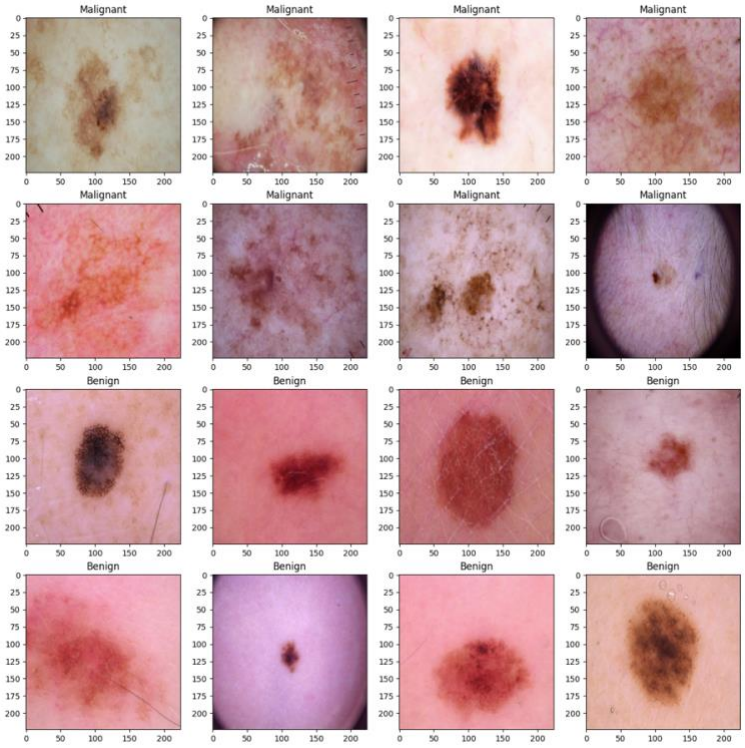
Skin Cancer Detection: Malignant vs. Benign

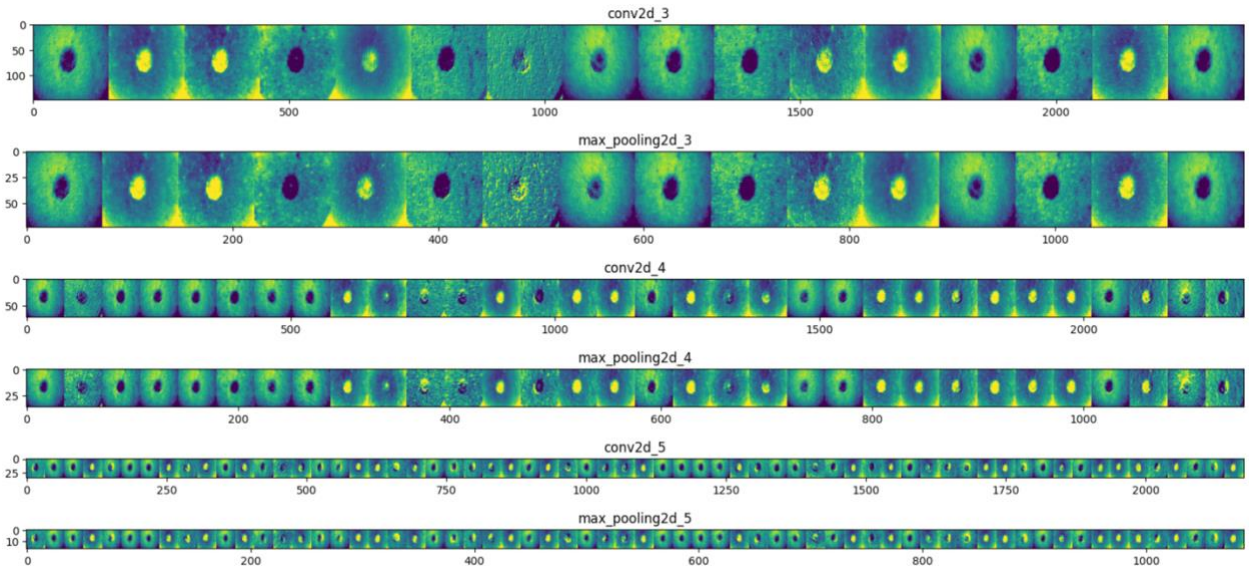
Introduction

Skin cancer is one of the most common forms of cancer to exist, with millions of diagnoses of basal and squamous skin cancers each year. As diagnoses continue to increase, early detection becomes crucial for successful treatment. The dataset we are using to build and train a convolutional neural network (ConvNet) is from The International Skin Imaging Collaboration (ISIC). The overall objective of this project is to not only successfully classify which images that are fed through our ConvNet are malignant or benign, but also build a model that can help increase skin cancer detection in the future. To accomplish this, we are using 3,297 images where our training dataset will include 1,197 malignant images and 1,440 benign images, and our validation (test) dataset will include 300 malignant images and 360 benign images. From this, our train-test split is about 75% training and 25% validation. The image below will give the reader a small visual representation of what the dataset that we are working with looks like.



Building a Convolutional Neural Network

The images being fed into the convolutional neural network are color images of size 150x150. These images will have three color channels, Red, Green, and Blue (R, G, B). The architecture of the convolutional neural network we aim to implement will have three modules (convolution + ReLU + max-pooling). The convolutions will operate on 3x3 windows while the max-pooling layers will operate on 2x2 windows. From this, our first convolution will extract 16 filters, the second convolution will extract 32 filters, and the third (final) convolution will extract 64 filters. To give the reader a visual representation of what is happening behind the scenes in our model, we can visualize the steps below, which take an image as input and output intermediate representations of all layers.



After completing this step, we will flatten our feature map to a one-dimensional tensor so we can add two fully connected layers on top as we are dealing with a binary (two-class) classification problem. Since this is a binary classification problem, the end of our network will include the sigmoid activation function, which allows the output for our network to be scalar (0 or 1).

The next step is configuring our convolutional neural network model for training. Since our model deals with binary classification (malignant or benign), we will be utilizing the binary cross-entropy loss function. The binary cross-entropy loss function is as follows:

$$\ell(y, \hat{y}) = -ylog(\hat{y}) - (1 - y)log(1 - \hat{y})$$

Throughout the model compilation, we will use the optimizer, "RMSprop" as the RMSprop optimization algorithm is preferable to stochastic gradient descent. The RMSprop optimizer is an extension of stochastic gradient descent with momentum as it restricts oscillations in the vertical direction and has a constant learning rate. With an increasing learning rate, the algorithm can reach convergence faster as it can take larger steps in the horizontal direction. We then define the metric we look to measure which is model accuracy.

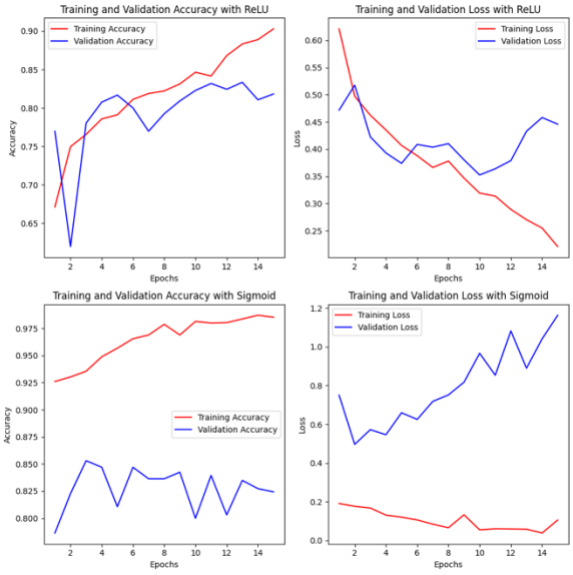
As we start preprocessing the data, the data generators will read our images and convert them into float32 tensors and feed them into our network. The generator will grab batches of 20 images of size 150x150 and their labels. The data is then normalized by pixel value in the [0, 1] range as the pixel values are typically between [0, 255].

After this, we can finally fit (train) our model. The training will take all 2,637 images of our training data set and all 660 images of our validation data set. We will then use the standard mini-batch size of 32 and continue training for 15 epochs. The RMSprop optimizer uses an extension of stochastic gradient descent and backpropagation to calculate the gradients of the model parameters with respect to the loss function. From this, we can see that our highest training accuracy achieved was 90.29% (in epoch 15). Furthermore, it is important to note that we can see our accuracy and validation accuracy improving throughout many of the epochs that the model passes through.

The model above utilizes the Rectified Linear Unit (ReLU) activation function. ReLU is one of the most popular activation functions as it can help mitigate the vanishing gradient problem that can occur when training deep neural networks. ReLU is also known to be computationally more efficient to compute when compared to other activation functions such as sigmoid. However, due to the constant desire of wanting to improve our model, we will also utilize the sigmoid activation function in the next trial and see which has better accuracy.

Following ceteris paribus, and only changing the activation function from ReLU to sigmoid, we can quickly see that the accuracy has improved significantly. The highest training accuracy achieved was 98.71% (in epoch 14), and the highest validation accuracy achieved was 85.30% (in epoch 3). From this, we can conclude that the sigmoid activation function performs better if we only look at model accuracy and validation accuracy. Below

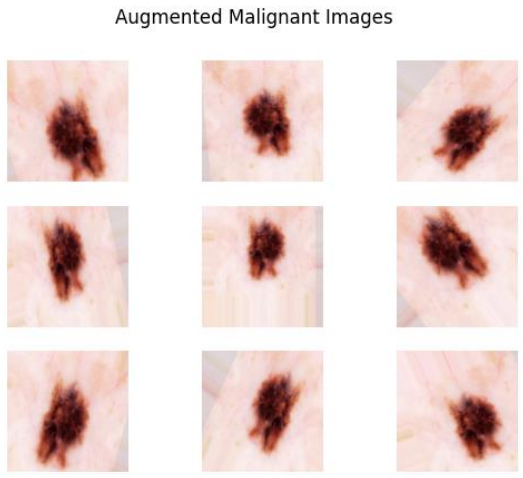
will include 4 subplots, each subplot consisting of which activation function was used and its metric measured (accuracy, validation accuracy, loss, validation loss).



Reduce Overfitting

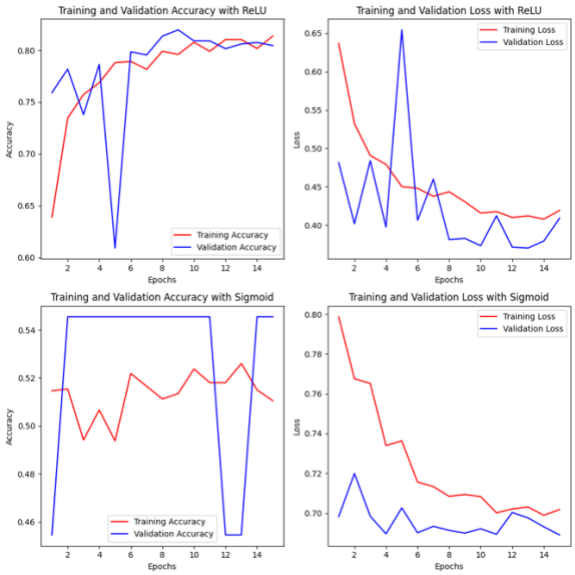
As you can see from the images above, we were able to attain high accuracy and moderately high validation accuracy. Due to this, now we want to generalize the model a bit so that we do not develop any overfitting. In practice with deep neural networks, there are numerous ways to reduce overfitting such as adding dropout, L1 penalty, L2 penalty, using a pre-trained ConvNet, and employing data augmentation. For this project, we will only be using data augmentation and dropout. We will look to add data augmentation to our data preprocessing step and then add dropout to our convolutional neural network.

With our small training dataset size, we look to augment the images with several random transformations so that when we train our model, it will never see the same image twice. Some techniques used with data augmentation include modifying the image rotation, width, height, horizontal flip, shear range, and zoom range. Below includes a subplot of multiple augmented images.



Even after data augmentation, the inputs that the model sees are still heavily intercorrelated so we also must introduce dropout into our model. Dropout is where you randomly drop units (both hidden and visible, along with their connections) from the neural network in training. For our purposes, we are adding a dropout layer to the fully

connected layer with the ReLU activation function and 512 hidden neurons. We then will repeat this step only with the sigmoid activation function. The dropout rate we are using is 0.5, which is the maximum amount (50%). After training the model, we can see the model accuracy and validation accuracy have both decreased.



Residual Block

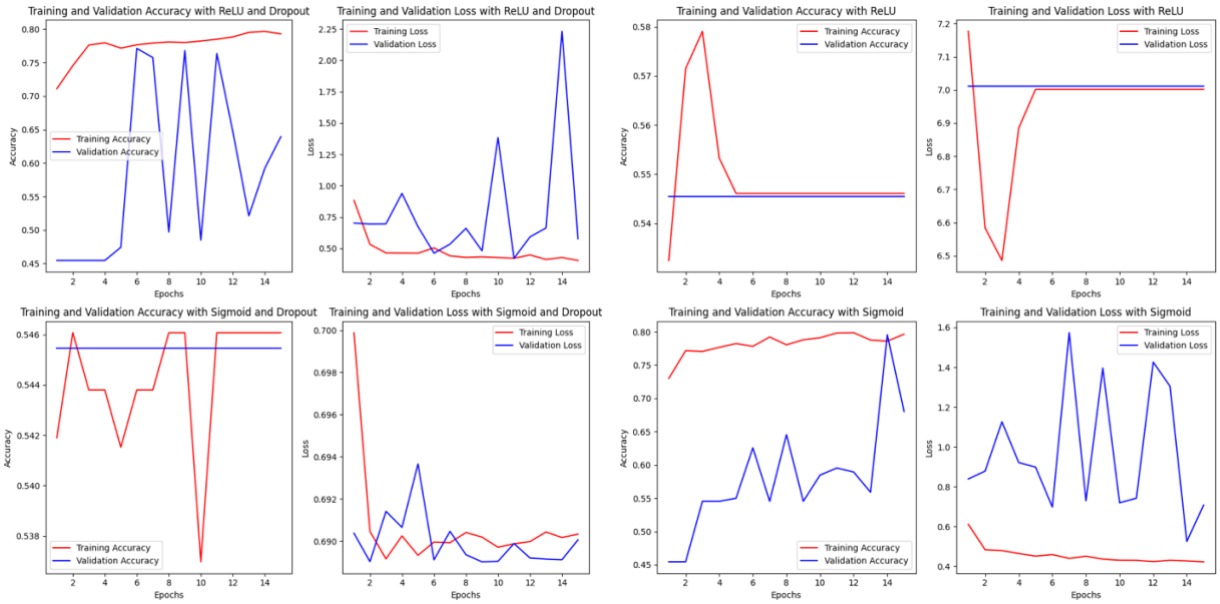
To help mitigate the vanishing gradient problem even further and improve our model's performance, we look to add a residual (skip) connection. The residual connection is an information shortcut around destructive or noisy blocks (such as blocks that contain ReLU or sigmoid activation functions, or dropout layers), which can enable error gradient information from early layers to propagate noiselessly through the deep neural network. In other words, the inputs propagate more efficiently through the residual connections and allow the gradient to go directly from the input layer to the output layer.

Batch Normalization

The model below also includes batch normalization, the most common form of data normalization. In short, normalization is a method that seeks to make different samples seen by a machine learning model more similar to each other. Batch normalization is where the data is centered on zero by subtracting the mean, and then giving the data a unit standard deviation by dividing the data by its own standard deviation. Batch normalization reduces the effect of internal covariate shift which helps the deep neural network learn more efficiently. Introducing batch normalization will help the model generalize well to new unseen data, which in our project are the augmented images that our model will train on.

Residual Block and Batch Normalization Results

Again, we performed four separate trials, where the first two trials include a dropout rate of 0.2 (20%), and the following two trials don't include any dropout. The dropout is added after three of the four residual blocks. The three residual blocks that include dropout all have a filter of 64 and include pooling. We also will be switching between ReLU and sigmoid activation functions so that we can conclude which activation function performed better. After completing trials with both residual block and batch normalization in our convolutional neural network, we concluded that ReLU achieved a higher training accuracy when dropout is added, and sigmoid achieved a higher training accuracy when there was no dropout added.



Dropout of 0.2 No Dropout

Conclusion

After exhausting our search for which convolutional neural network model performs the best, we can conclude that if we only desire training accuracy, we would use the first model introduced with the sigmoid activation function as it achieved a training accuracy of 98.71%. However, since we want to introduce some regularization techniques so that our model not only performs well, but generalizes well and does not overfit the data, we finally conclude that the desired model will include ReLU, dropout, and data augmentation. As stated before, we are choosing the ReLU activation function as ReLU avoids the vanishing gradient. With some fine-tuning, our model scored a high training accuracy of 86.65% (in epoch 97) and a high validation accuracy of 86.52% (in epoch 59). With data augmentation on the training data, we rescaled the data and set the rotation range to 40, width shift range to 0.2, height shift range to 0.2, shear range to 0.2, zoom range to 0.2, and horizontal flip to "True." Some of the fine-tuned model parameters we implemented were adding 0.05 dropout after the first max-pooling layer, 0.1 dropout after the second and third max-pooling layers, and another 0.1 dropout to the fully connected dense layer. The first convolution extracted 16 filters of size 3x3, followed by a max-pooling layer of window size 2x2. The second convolution extracted 32 filters of size 3x3, followed by a max-pooling layer of window size 2x2. The third convolution extracted 64 filters of size 3x3, with another max-pooling layer of window size 2x2. Then, we flatten our feature map to a one-dimensional tensor and create a fully connected layer with 512 hidden neurons. The output layer includes a single neuron with the sigmoid activation function. The loss function was again binary cross-entropy, the optimizer was RMSprop, the learning rate was set to 0.001, and the scoring metric was accuracy. In our last attempt, we also increased the epoch size to 100 epochs which allowed us to better witness the convergence of our model. In training, the total amount of trainable parameters was 9,494,561. The result of our final model is below.