Deep Convolution Learning Networks as an Early Detection System for Skin Cancer

by Daniel C. Chan, September 2018

Definition

Domain Background

Skin cancer is the uncontrolled growth of abnormal skin cells. It occurs when damaged skin cells (most often caused by ultraviolet radiation) triggers mutations, or genetic defects, that lead the skin cells to multiply rapidly and form malignant tumors. Melanoma, is the most dangerous form of skin cancer, kills an estimated 10,130 people in the US annually. Skin cancer is also the most common type of cancer and the easiest to cure, if diagnosed and treated early. Each year in the U.S. over 5.4 million cases of non-melanoma skin cancer are treated in more than 3.3 million people[1].

To spot potential Melanoma skin cancer, the American Cancer Society recommend conducting a monthly self-visual inspection using the “ABCDE rule” to look for Asymmetry, Border, Color, Diameter and Evolving, respectively, of a suspicious skin lesion[2].

To aid visual inspection one can purchase an affordable dermatoscope for less than $300 from an e-commerce site such as Amazon[3]. It is equipped with its own bright light source and a magnification lens that can illuminate deep structures in the skin not visible to naked eyes due to the glare from skin surface[4]. High resolution pictures can be taken with a smart phone or traditional camera for applying the “ABCDE rule”. I believe using these images to train a machine-learning model could form the basis of an affordable early detection system and potentially save more human lives.

The International Skin Imaging Collaboration (ISIC) is an international effort to improve Melanoma diagnosis, sponsored by the International Society for Digital Imaging of the Skin (ISDIS). The ISIC Archive[5] contains the largest publicly available collection of quality controlled dermoscopic images of skin lesions. ISIC started hosting open challenges for skin lesion analysis towards Melanoma detection in 2016. Training and testing data are readily available through a simple registration process. For the 2017 challenge, ISIC provided 2,000 images and allowed the use of external data for training purpose only. Models that have been trained with these data were then put to a blind test by classifying the disease captured in 600 different unlabelled images. There were 23 final test set submissions. All top submissions implemented various ensembles of deep learning networks. All used additional data sources to train, either from ISIC, in-house annotations or external sources[6]. I believe the key motivation by this approach is to prevent overfitting caused by small training dataset and avoid the bias induced by imbalanced disease types. Liberal use of external data resources can achieve higher score; however, it can also make the results difficult to replicate and not easy to understand key driving factors for building a robust early detection system.

For this study I chose to focus on the 2017 datasets and opted not to use data from other sources that may contain varying degrees of consistency and quality. The objective is to provide a systematic way to evaluate the capability and limitations of deep convolution neural networks through transfer learning.

Problem Statement

If basic structural differences exist that can distinguish different types of skin disease from each other, a Deep Convolution Neural Network (CNN) should be able to learn from these dermoscopic images and classify them consistently. However, publicly available images are limited, and in most cases less than a thousand for each skin disease type and will not lend themselves for training CNN from scratch. Therefore, transfer learning/feature extraction or fine tuning would be a more appropriate approach. In addition, benign cancer data are more abundant than malignant ones, thus creating a highly imbalance dataset for training, if it is not treated appropriately, the resulting model could have a bias towards benign predictions and defeat the purpose of an early detection system. Computational efficiency is also a concern, since fine tuning can take hours or days to perform for each case and can limit the number of parameters that can be explored.

The goodness of the prediction is measured by the balance between $Sensitivity$ and $Specificity$ as well as the $ROC - AUC$ (to be described) for binary classification. $Sensitivity$ measures the proportion of actual positives that are correctly classified, whereas, $Specificity$ is the proportion of actual negatives that are correctly classified. The $Specificity$ (or $1 - FP$) by average dermatologists is about 0.6 (or a FP of 0.4) at a $Sensitivity$ of 0.8 [7]. **The objective of this investigation is to devise a methodology that can produce a FP lower than 0.4 at the same $Sensitivity$.**

Metrics

Class prediction $accuracy$ measures the proportion of data with the correct class predicted. It is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP =True Positive, TN=True Negative, FP=False Positive and FN=False Negative.

Under certain circumstances, however, it could be a misleading indicator. In the case where both TP and FP are zero, the $accuracy$ can reach 80% even if the ratio between TN and FN is at 4. This is an example which the model could be making all negative predictions and still achieved a respectful $accuracy$. For the ISIC 2017 data, the ratio between benign and malignant data is 4.3, so $accuracy$ would not be an appropriate metric. Instead, $Specificity$ and $Sensitivity$ will be used as the evaluation metrics [8]. $Sensitivity$ (also known as $Recall$ or True Positive Rate) is defined as:

$$Sensitivity = \frac{TP}{TP + FN}$$

and $Specificity$ (also known as True Negative Rate or $1 - FP$) is defined as:

$$Specificity = \frac{TN}{TN + FP}$$

One can also plot True Positive Rate against False Positive Rate to measure Receiver Operating Characteristic (ROC) and the area under the ROC curve, also known as AUC, is an effective metric, even for imbalance data set, to gauge the performance of a binary classification model [9].

In this case, the Challenge organizer asked for 2 binary classifications. The first one is to distinguish Melanoma from Nevus & Seborrheic Keratosis. The second one is to distinguish Seborrheic Keratosis from Nevus & Melanoma. I will refer to them as Melanoma AUC and Seborrheic Keratosis AUC going forward.

Analysis

Data Exploration

Training images provided by ISIC are in JPEG format with various sizes. Figure 1 shows the distribution of image aspect ratio, area, length and width. Over half of the images have an aspect ratio of 1.5. Since all the deep learning models we are using can only handle square images, they must be cropped prior to training, otherwise the original shape would be distorted when resized and rotated, possibly degrading accuracy of the predicted outcome. In addition, over 90% of the images have an area of less than 10 square mega pixels, for a 64-bit accuracy, they would require less than 80 MB of memory.

Figure 2 shows the count of each disease type. There is a significant imbalance – Nevus has the highest number of training images with Seborrheic Keratosis being the least. Figure 3 shows there are 2,000, 150 and 600 images available for training, validation and testing, respectively. Nevus images are most abundant in all 3 data sets. The imbalance is most pronounced in the training set, number of Nevus images is more than 5 times of those for the other disease types. A special treatment is needed to handle this situation, otherwise the trained model will have a bias towards predicting Nevus.

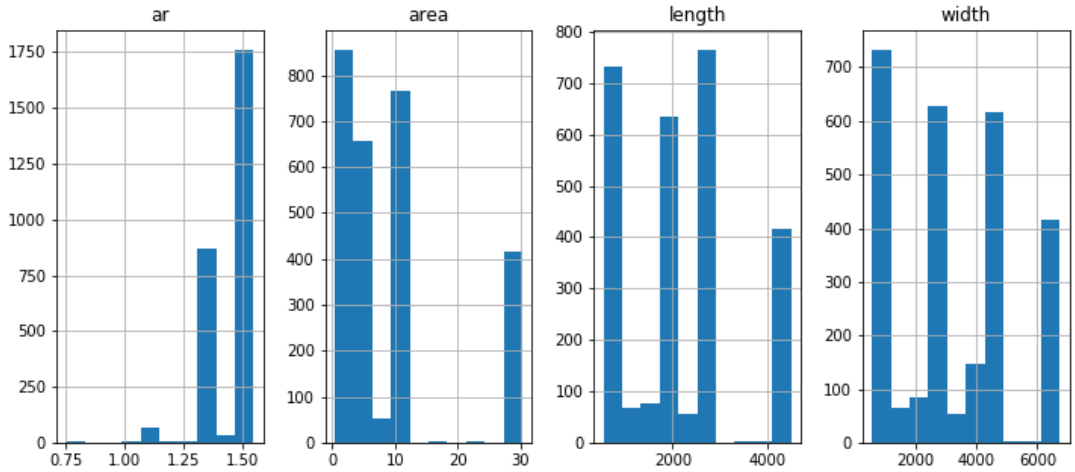


Figure 1: Characteristics of combined training, validation and testing images. From left to right, histogram for aspect ratio, area, length and width. Area is measured in square mega-pixels.

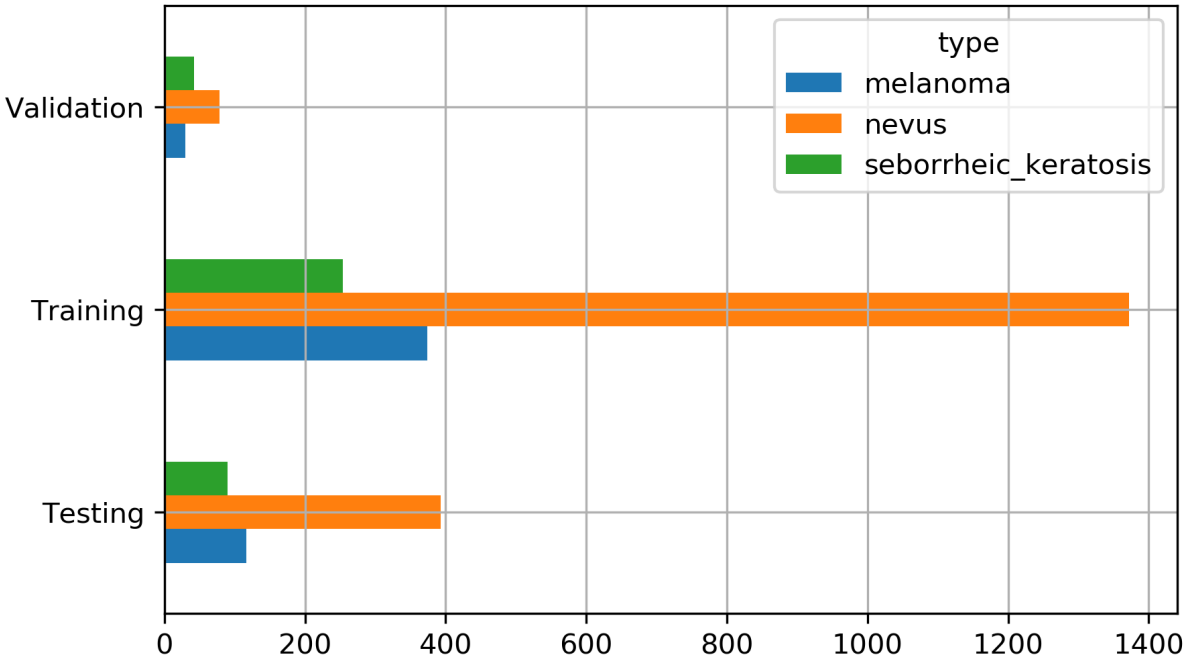


Figure 2: Image count for each disease type. For training, the ratio of Melanoma to Nevus to Seborrheic Keratosis is 0.27:1.0:0.19; for validation it is 0.39:1.0,0.54 and testing it is 0.30:1.0:0.23.

data type	Test_Data	Training_Data	Validation_Data	All
melanoma	117	374	30	521
nevus	393	1372	78	1843
seborrheic_keratosis	90	254	42	386
All	600	2000	150	2750

Figure 3: A summary of data types available for training, validation and testing. There are 2750 images all together, 2000 for training, 150 for validation and 600 for testing.

Exploration Visualization

Figures 4, 5 and 6 show a random sample of Melanoma, Nevus and Seborrheic Keratosis images in their original sizes. Images are clear and show different skin tone, lesion area is located near the center of each image. However, in some images, the view for the lesion area is interfered by skin hair and artificial markers. It is good that these interferences exist in the training dataset as the models can learn to cope with these real life situations. The structure for Melanoma tends to spread out more and does not exhibit an organized form, its color is also darker. Nevus and Seborrheic Keratosis structures are more lumpy with rounded shape and in some cases symmetric. Color is mostly brown.

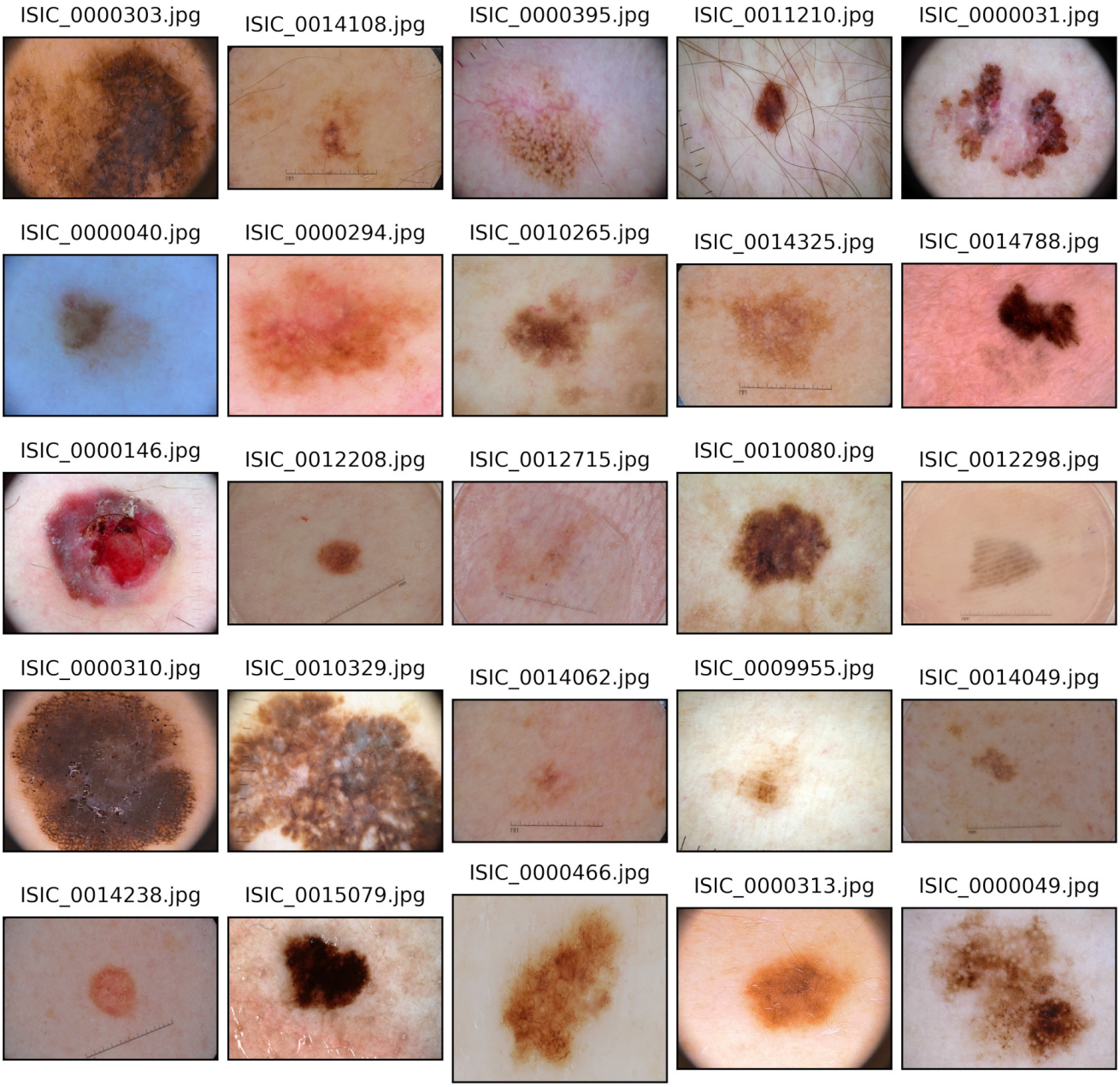


Figure 4: A random sample of Melanoma images. The file name is printed on top of each image.

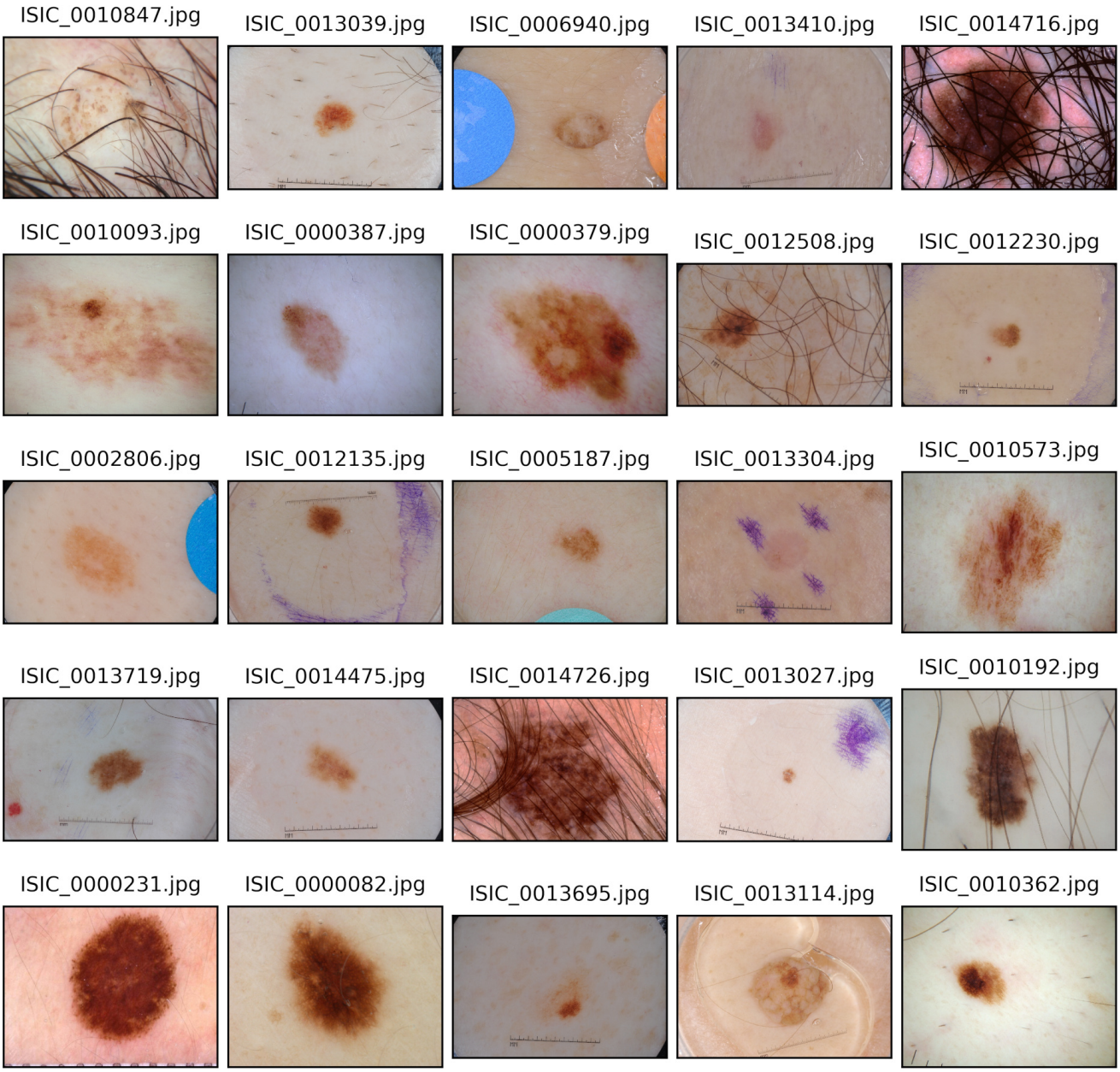


Figure 5: A random sample of Nevus images. The file name is printed on top of each image. The presence of body hair and artificial markers is evident.

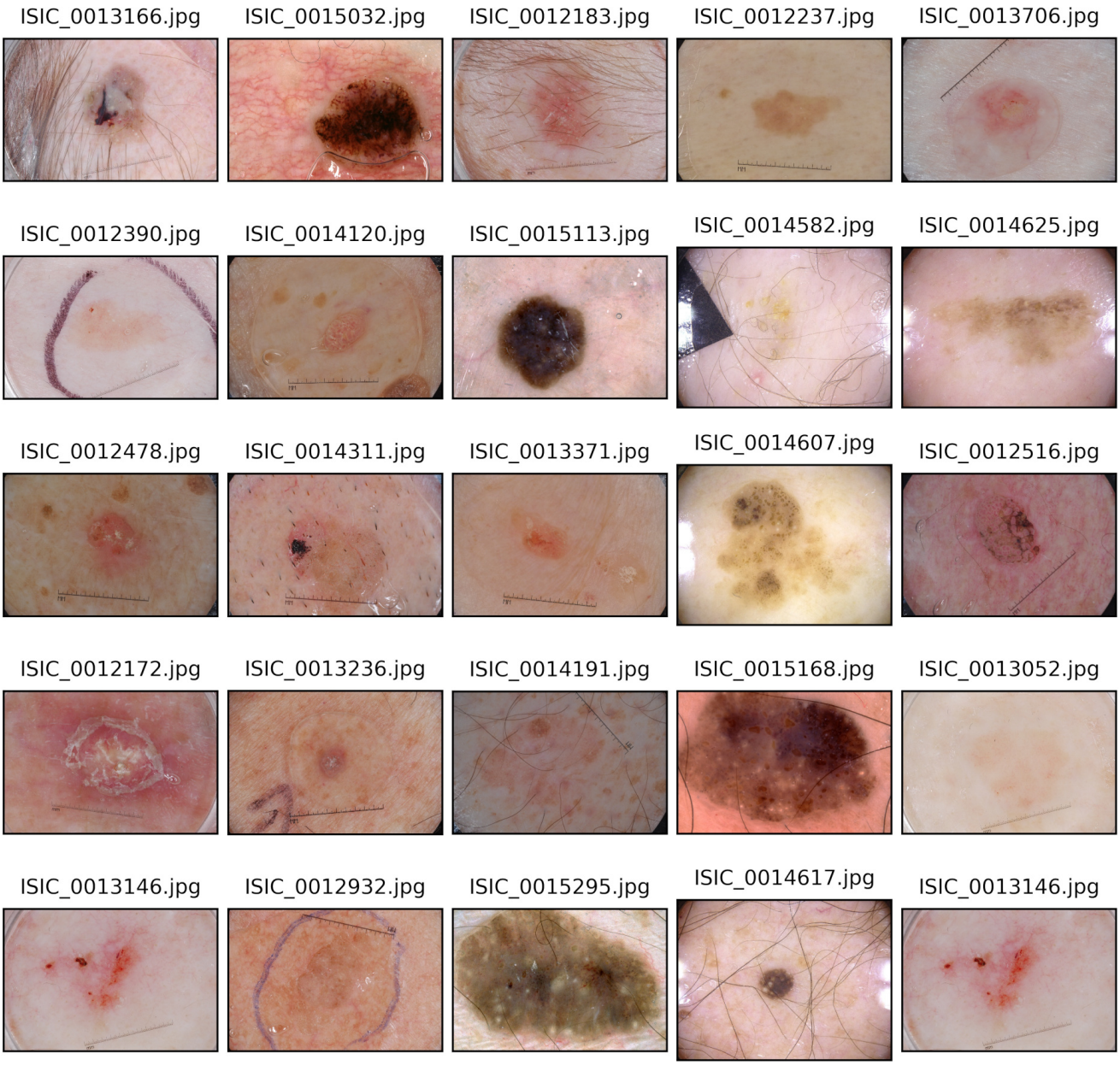


Figure 6: A random sample of Seborrheic Keratosis images. The file name is printed on top of each image. Artifical markers, body hair and light reflection are evident in some of these images.

Prior to conducting the training, I examined if these images provide distinguishable features that will facilitate classification. The *PCA* and *TSNE* functions in *sklearn* have been used (detailed implementation is not provided here since it is not a focus of the current investigation). Suffice to say, the first 50 PCA components account for about 45% of variations. Figure 7 is the 2-component t-SNE visualization which reveals 4 visible clusters. Malenoma images have a higher concentration in the upper left hand region but they are co-mingled with the other 2 skin disease types. Classifying them, correctly would require sophisticated methods. Seborrheic Keratosis images concentrate more toward the lower part of the figure and form a tight cluster indicating that it may not be as challenging to classify them correctly.

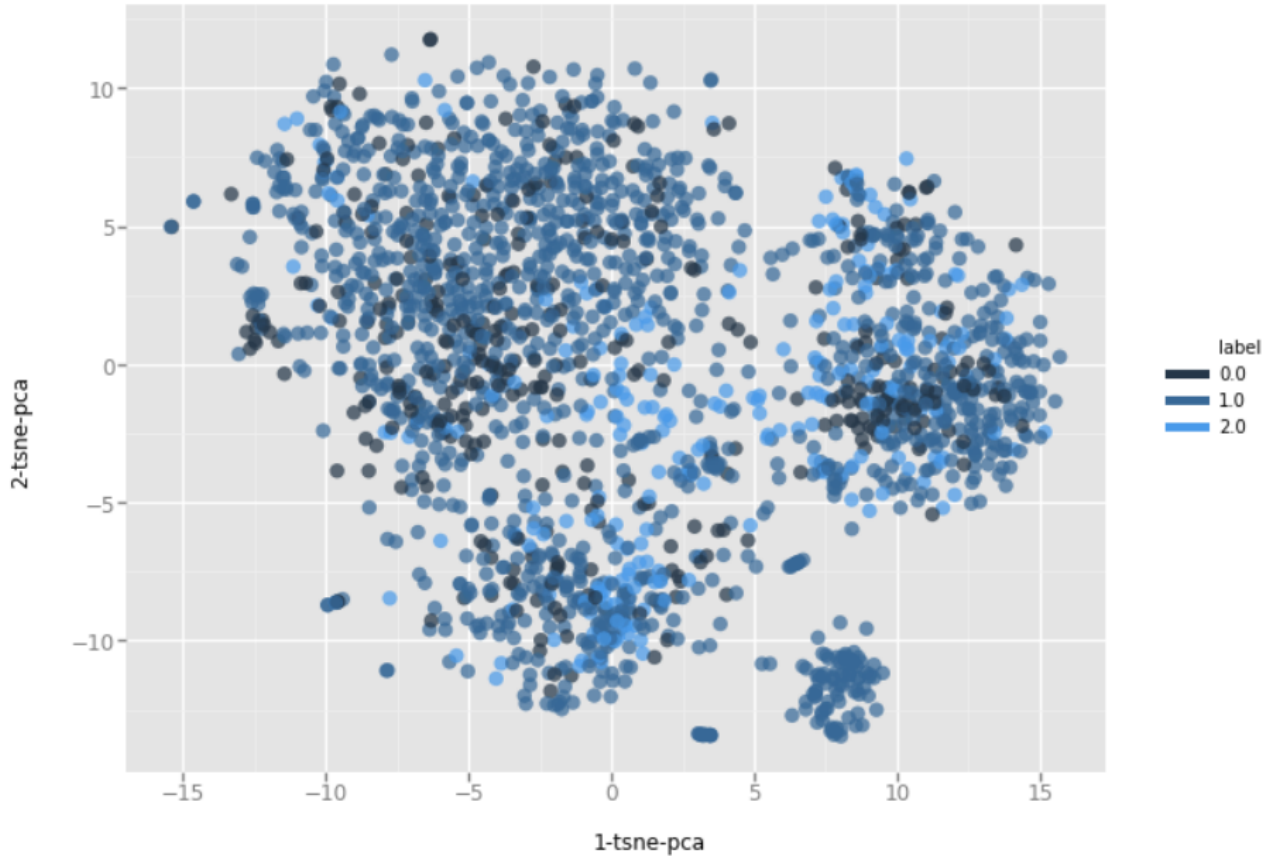
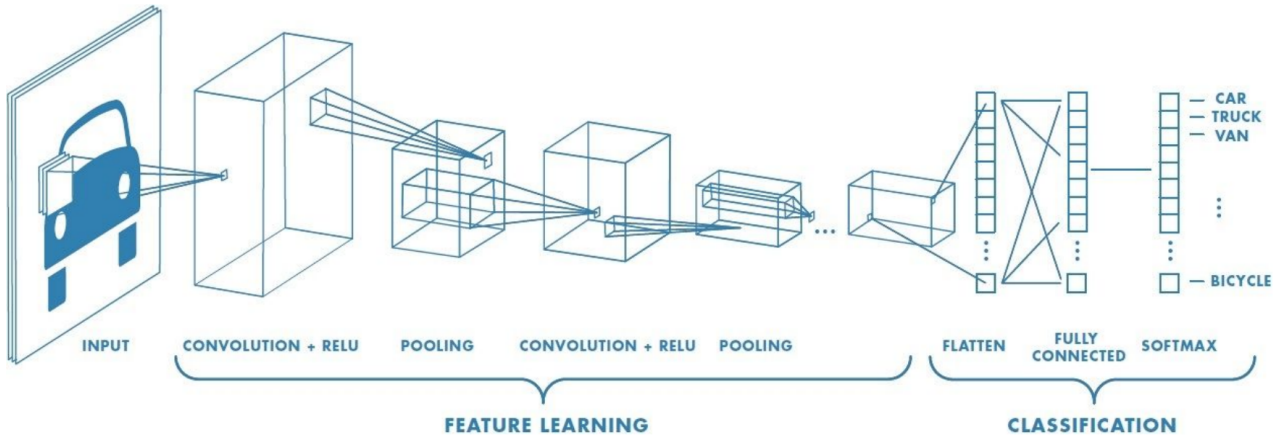


Figure 7. t-SNE visualization of the training data, there are $2,000$ images with 3 different types, Label 0 is Melanoma, 1 is Nevus and 2 is Seborrhaic Keratosis

Algorithms and Techniques

Unlike physical objects such as cars and buildings, skin disease images do not have defined edges, corners and orientation. Therefore, methods that depend on manual feature engineering will not work well as they are difficult to generalize. Traditional fully connected neural network could be considered, but the demand for computational resources would make it impractical. For instance, a $224 \times 224 \times 3$ image will generate $150,528$ weights for each neuron. If merely 100 of them are present in the hidden layer, the total number of weights can reach 15 billion for just one image! The 2017 ISIC Challenge has $2,000$ training images and the number of weights can quickly balloon to an astronomical number of 30 billion. Even with a lower single precision arithmetic of 4 bytes per floating point number, the among of required memory can be as high as 120 GB. This demonstrates connecting neurons to each image pixel will not be a viable approach, instead we will connect each neuron to only a local region of the 3-dimension image input. This is the foundational idea of Convolution Neural Network (CNN) and is shown below [19]. It consists of 2 stages - Feature Learning and Classification. During the Feature Learning stage, the image is convoluted with a set of different filters to create a receptive field with the extend specified by the number of filters.



Size of a filter, F, number of filters, K, stride, S, and padding, P are input parameters. For the same size of image we discussed above, with $F = 3$, $K = 64$, $P = 0$ each one of the $224 \times 224 \times 64$ neurons will be connected to the same region of the image with a size $3 \times 3 \times 3$. As they are connected to the same region, we can impose the condition that they share the same weights and reduce a size of $3 \times 3 \times 3 \times 64$ or $1,728$ which is a much more manageable than that of a fully connected layer. The figure below provides operational details for an example where $3 \times 3 \times 3$ filters convolute with an $7 \times 7 \times 3$ image [19].

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	2	0	0	1	0
0	2	2	0	1	0	0
0	2	2	1	2	1	0
0	0	1	1	0	2	0
0	2	1	0	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	1	1	0	2	2	0
0	0	1	1	0	2	0
0	1	2	0	2	0	0
0	0	2	0	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	1	0	0	0
0	2	1	0	0	1	0
0	0	2	2	2	1	0
0	1	2	1	0	2	0
0	2	1	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	-1	-1
-1	0	0
-1	-1	0

w0[:,:,1]

-1	0	1
-1	0	0
-1	-1	-1

w0[:,:,2]

-1	-1	1
0	0	0
1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	1
1	1	0
0	0	0

w1[:,:,1]

1	0	0
1	-1	1
-1	0	0

w1[:,:,2]

-1	1	0
0	-1	1
1	0	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

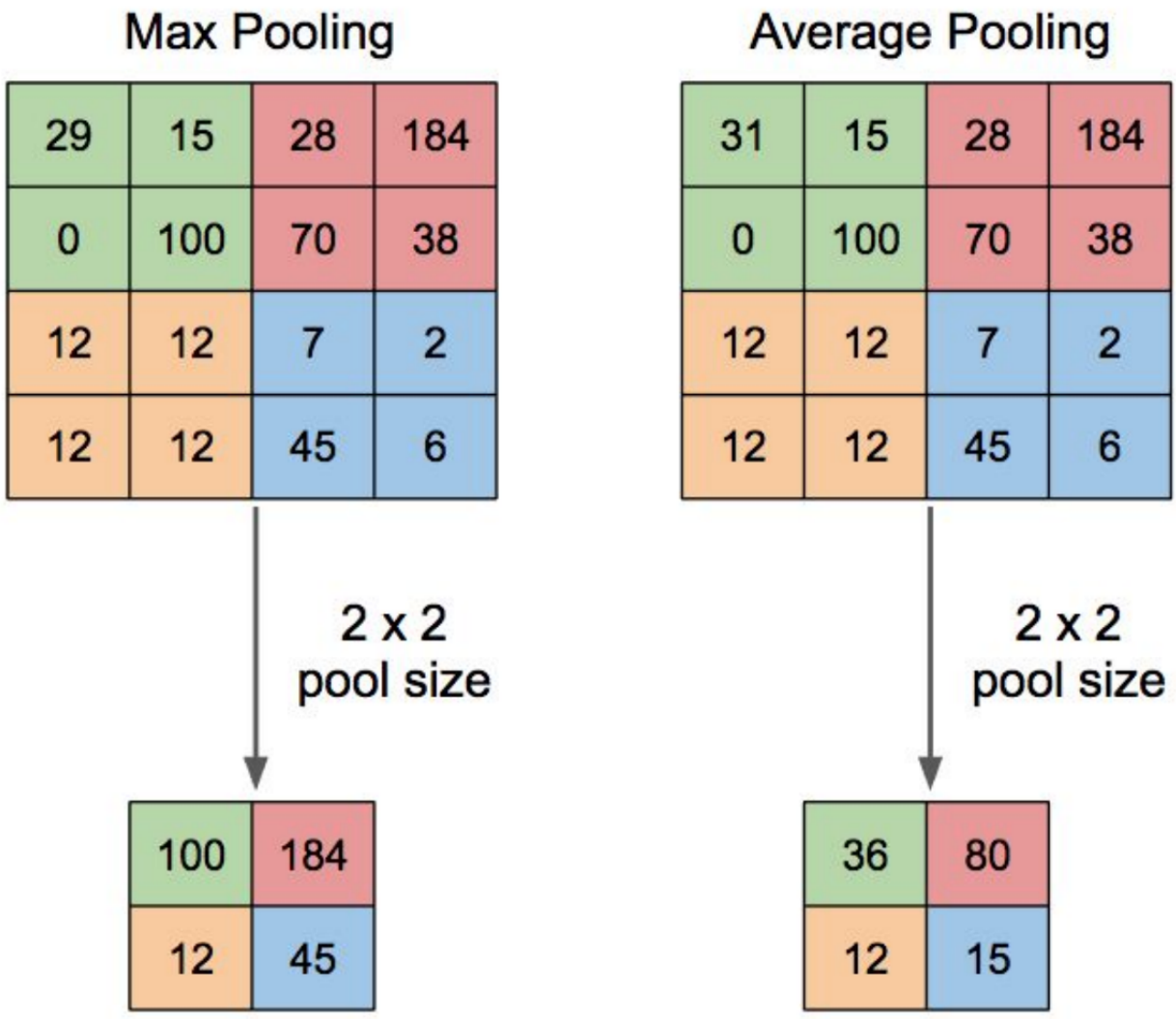
-6	-7	-5
-9	-6	-9
3	-5	-8

o[:,:,1]

2	3	-2
7	4	1
5	5	7

A Pooling layer is oftenly inserted in-between successive Convolution layers. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The most common forms of Pooling are Max and Average, as shown below they can reduce the size by a factor of 2.

It is a repeat use of Convolution and Pooling layers that significantly reduce CNN's demand for computing resources. For instance, the whole VGGNet is composed of Convolution layers that perform 3×3 convolutions with $S = 1$ and $P = 1$, and Pooling layers that perform 2×2 Max pooling with $S = 4$ and $P = 0$. The memory requirement is 93 MB per image and the model has 138 million parameters [20].



To accelerate numerical convergence, I employed the cyclic learning rate method advanced by Leslie[11]. Learning rates are varied periodically between a lower and an upper bound in a triangular pattern as shown in Figure 8. One cycle consists of 2 step sizes. The learning rate starts at the lower bound value and increases linearly, reaching the upper bound at the end of one epoch. The next epoch will start with the upper bound value and decreases linearly towards the lower bound. The purpose of this approach to find a flat local minimum on the loss function surface, so that the trained model can generalize well.

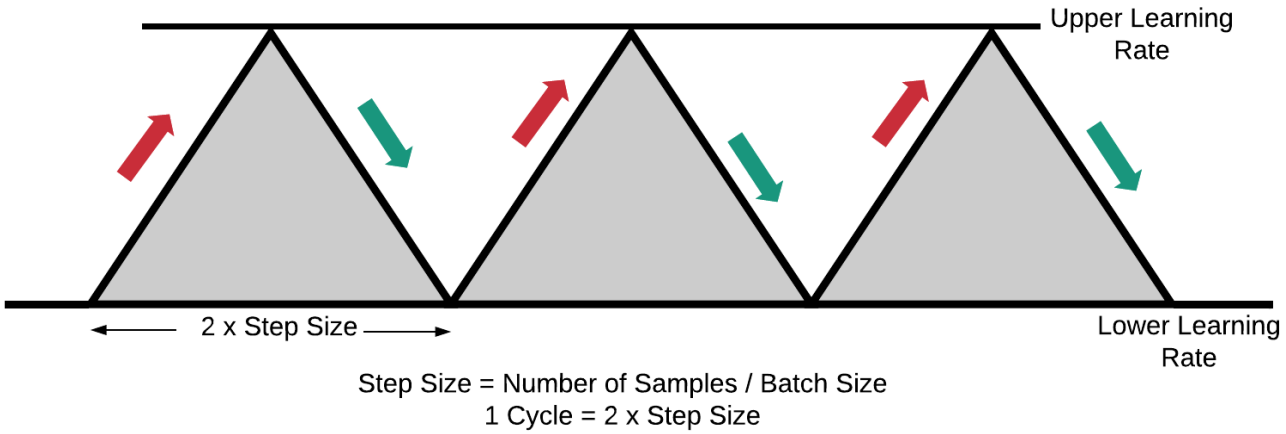


Figure 8: The triangular pattern of cyclic learning rate. The difference between low and upper learning rates can be as high as one order of magnitude.

To assess prediction repeatability, I used a 5-fold cross-validation method and divided the training data using the *sklearn.model_selection.StratifiedKFold* function below to preserve the original class distribution of training dataset.

```
from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
```

Benchmark

I looked at 2 sets of benchmark, first one is to verify the current solution platform is stable and computationally efficient to produce competitive results. Second is to test the current solution methodology on 2017 ISIC dataset using MobileNet [18] which is designed specifically for mobile and embedded devices and is less demanding on computing resources with the sacrifice of lower accuracy.

In essence, the transfer-learning solution I am assembling together is for multi-class classification. To quantify its capability, I have applied it to the well-known CIFAR-10 dataset [12] which contains 60,000 32x32 pixels color images in 10 different classes which represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images for each class or 60,000 images in total. I used 50,000 of them as training data and 10,000 for validation to train ResNet50 and VGG16 models. All the images were scaled up to 224×224 pixels. Weights were initialized to those obtained from ImageNet. The quality of the prediction is measured by its $accuracy$ in classifying the test data correctly. Figures below show the convergence rate of Stochastic Gradient Decent Method (SGD), with momentum=0.8, batch size=32, cyclic learning rates that vary between 10^{-4} and 10^{-2} for 50 epochs. It took 6 hours of computational time on a Nvidia 1080Ti GPU or 7.2 minutes/epoch. Validation $accuracy$ of the current approach is 93.7% and 95.51% for VGG16 and ResNet50, respectively, which are very competitive to the top result of 96.53% reported by Rodrigob [13]. For the ResNet50 model, it reaches a testing $accuracy$ of 95.51% after 2 epochs, the computation could have stopped there using an early stopping strategy and resulted in a computational time of 14.4 minutes which would make it a top-5 training time in the DAWNBench competition [14]. The unique convergence characteristic for cyclic learning rate is evident. Not only was I able to train all 176 layers in the ResNet50 model all at once without having to go through a layer-by-layer progression, I also managed to use a learning rate of 10^{-2} without suffering from a numerical divergence. One could also make a point this rapid convergence rate could attribute to the fact that both CIFAR-10 and ImageNet have some common images.

With these results established, I am confident the current solution platform is stable and has a good chance of predicting high quality results for skin cancer detection. My next step is to establish a benchmark comparison with the shallower MobileNet CNN (96-layer deep) applied on the 2017 ISIC test data. The predicted AUCs are 0.77 and 0.81 as shown in Figure 11. They took 478 seconds and 30 epochs to obtain. They are not as good as the top submissions to the Challenge [7] so there is room for improvement which is the objective of this study.

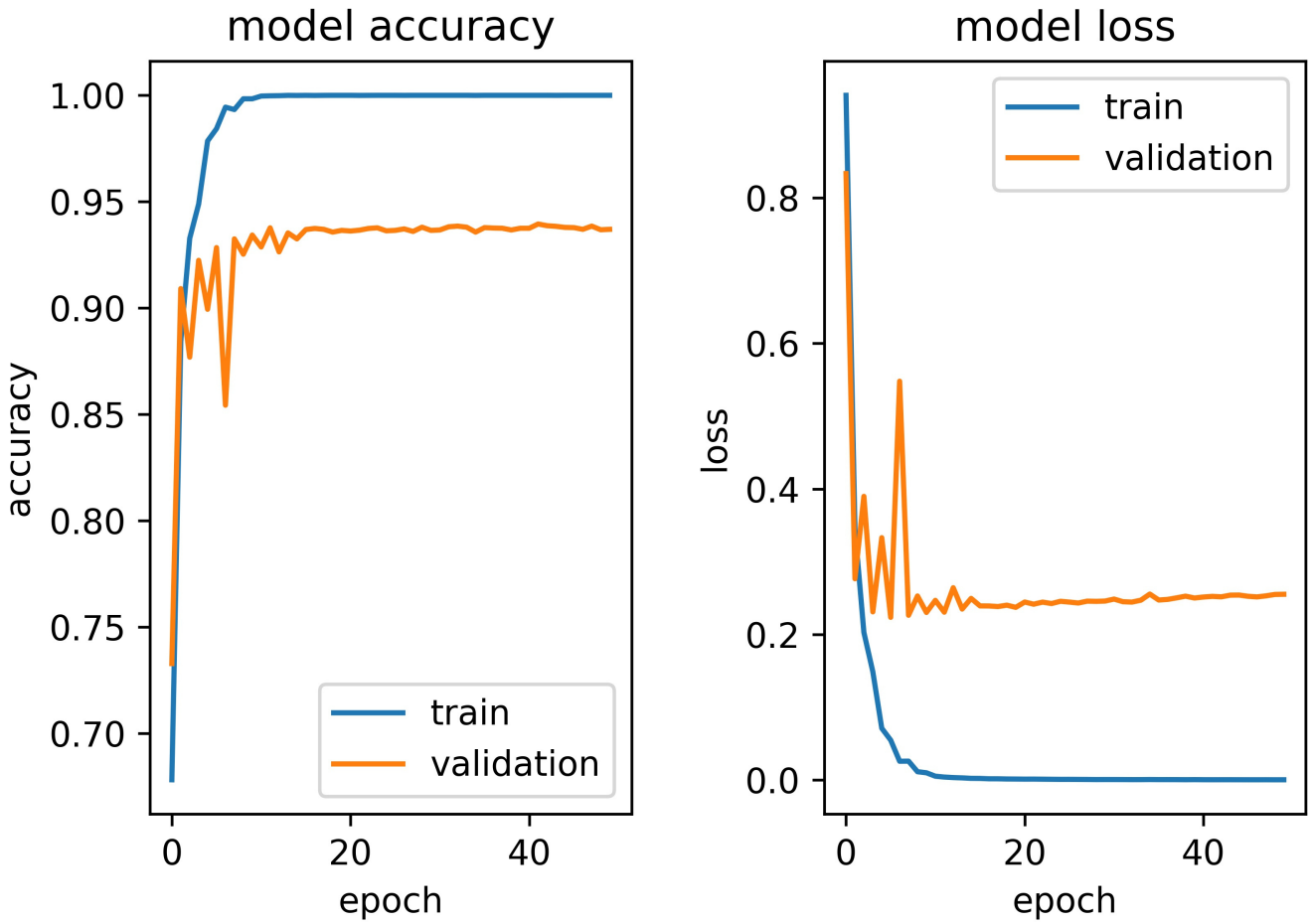


Figure 9. Convergence characteristics of the VGG16 model. Training accuracy reaches 100% and training loss is nearly zero. No overfitting is detected. Cyclic learning rate upper bound set to 10^{-3} to prevent numerical divergence. Batch size is 32.

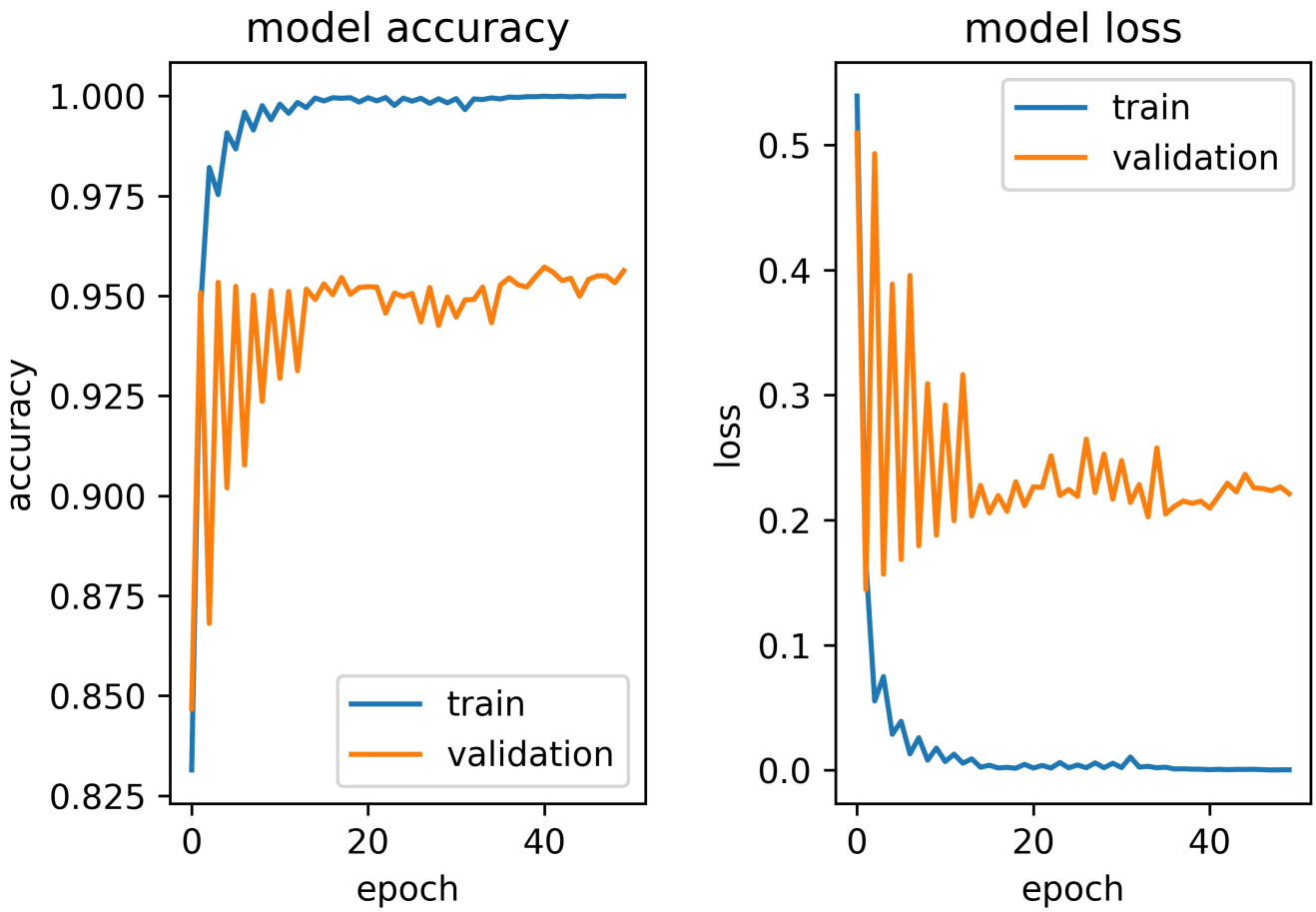


Figure 10. Convergence characteristics of the ResNet50 model. Training accuracy reaches 100% and training loss is at 2.84×10^{-4}. No overfitting is detected. Cyclic learning rate upper bound set to 10^{-2}. Batch size is 32. Model achieves the steady-state validation accuracy and loss after 2 epochs.

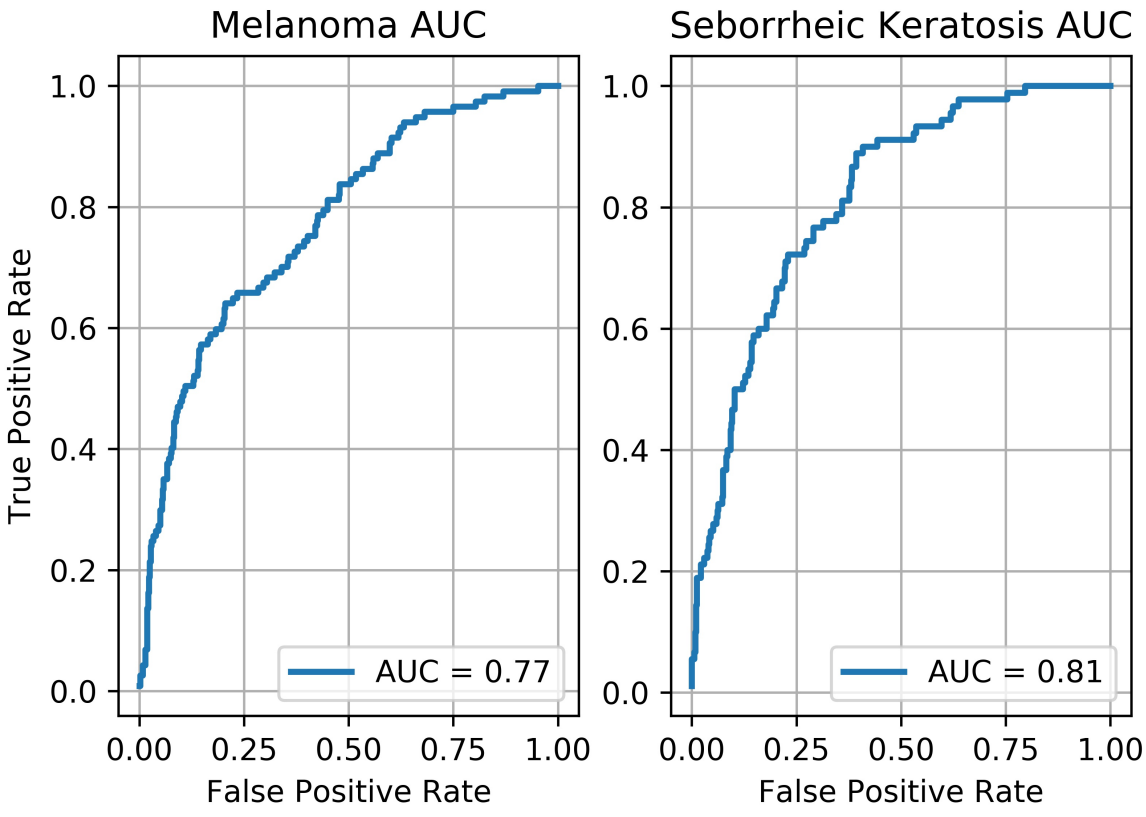


Figure 11: 2017 ISIC test data AUCs predicted by a fine tuned MobileNet model. Initialized by ImageNet weights, training sample size=$2,000$, batch size=32, upper bound learning rate= 1.0×10^{-3} , weights decay= 1.0×10^{-4}, took 478 seconds and 30 epochs.

Methodology

Data Processing

According to the "ABCDE" rules, asymmetry and diameter of a skin lesion are key attributes to look for, therefore it is important not to elongate or squish any of the images. As all the CNN models available in Keras [10] only work with square images in a size of either 224×224 pixels or 299×299 pixels, I need to run these images through several preprocessing steps:

1. Locate the center of each image
2. Determine the shorter dimension of 2 sides
3. Use the shorter dimension, crop a square image and output it in JPEG format to a folder that has a name corresponds to the skin disease type
4. These images will be resized to the appropriate dimensions during training (e.g. 224×224 for ResNet50 and VGG16)

Figure 12 is a random sample of Melanoma images that went through the above process. Cropping from the center of each image appears to be a viable option. The diseased area exhibits sufficient visual coverage and the resizing to 300×300 pixels does not show any visible degradation.

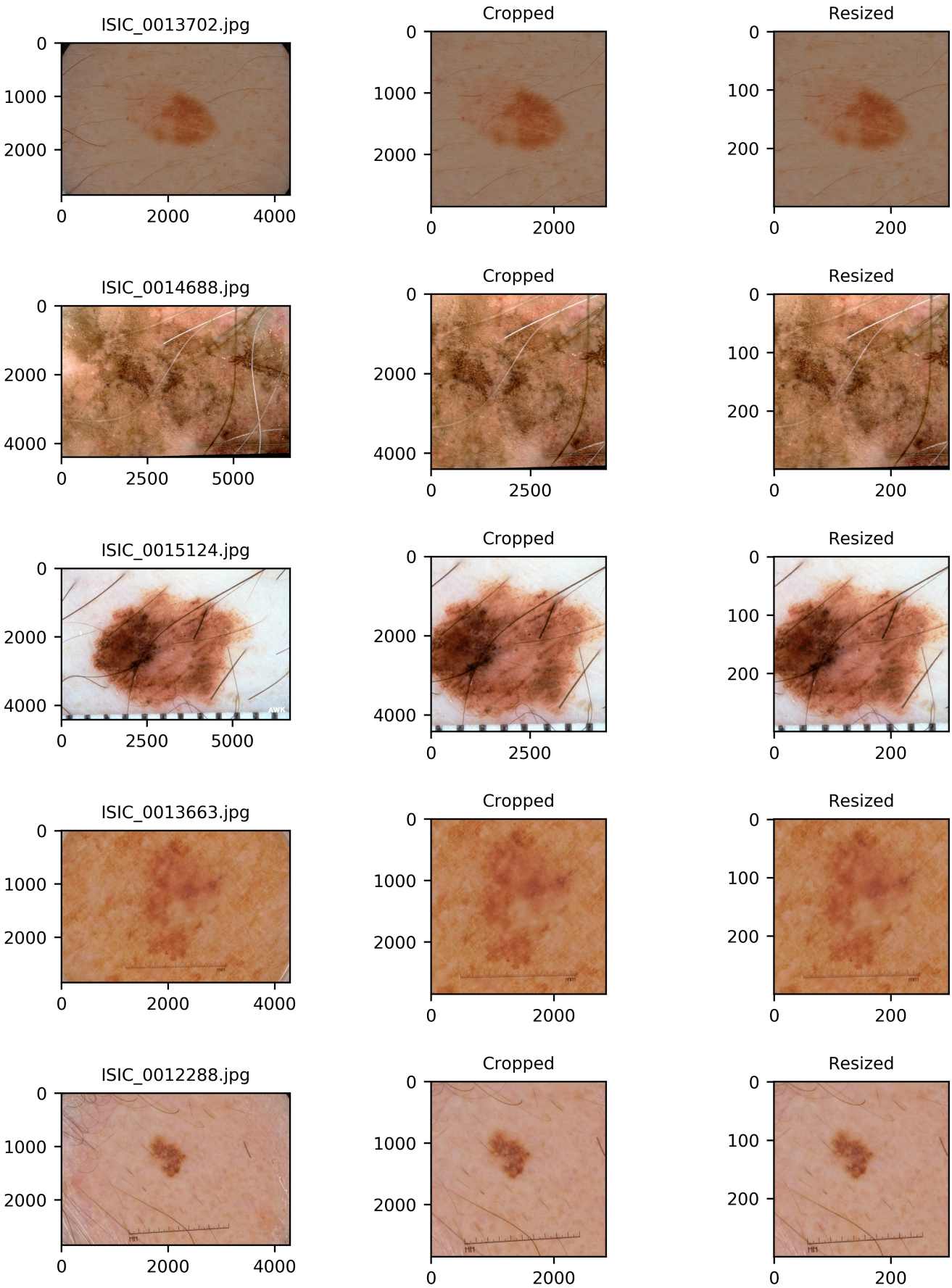


Figure 12: A random sample of Melanoma images. The file name is printed on top of each image. They are arranged from left to right: original, square images cropped with the shorter dimension and resized to 300×300 pixels.

To compensate the imbalance in training data, I randomly oversampled each image type $1,000$ times with the following geometric transformations to avoid overfitting:

- flip above the horizontal or vertical axis
- rotate by either 90 or 270 degrees

The implementation was done with the Augmentor [15] API shown below:

```
import Augmentor
from Augmentor.Operations import *
folders=['melanoma','nevus','seborrheic_keratosis']

for folder in folders:
  folder_path = root_path+folder
  p = Augmentor.Pipeline(folder_path)
  p.rotate90(probability=0.3)
  p.rotate270(probability=0.3)
  p.flip_left_right(probability=0.2)
  p.flip_top_bottom(probability=0.2)
# Now we can sample from the pipeline:
  p.sample(1000)
```

The Nevus to Melanoma ratio improved from 3.67 to 1.72 and the Nevus to Seborrheic Keratosis ratio improved from 5.4 to 1.89. As shown below, these transformations did not alter the shape of original image and maintained its squareness.

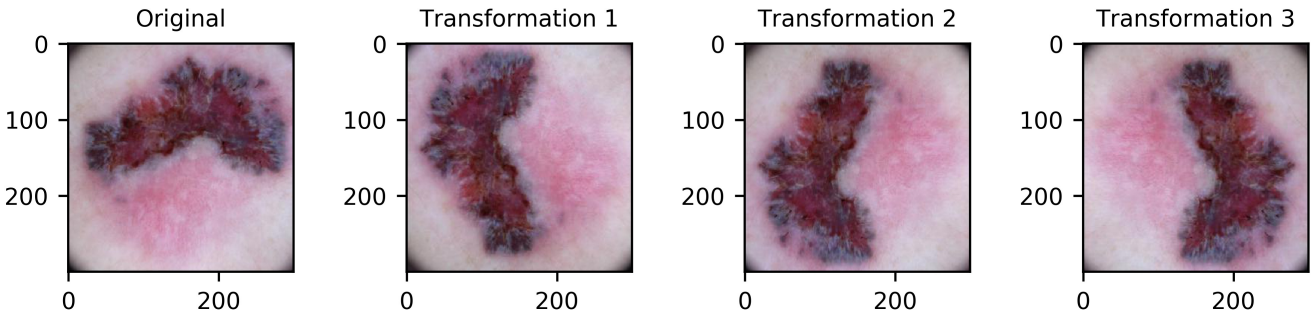


Figure 13. Augmented Melanoma images with 3 different geometric transformations.

By taking this approach, I have significantly reduced the storage space (and memory requirement). The Ubuntu Disk Usage Analyzer indicates the size of original images folder and processed images folder is 6.2 GB and 59.6 MB, respectively. The net result is a 100-time reduction in size.

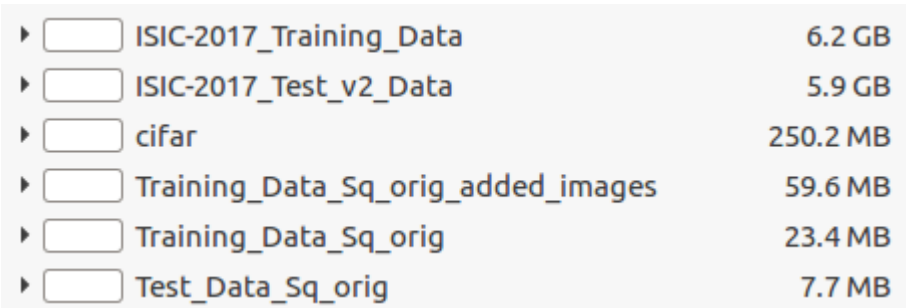


Figure 14: Storage requirement for image files showing a significant reduction in size after pre-processing.

Implementation

I explored 4 different CNN models that have been trained with ImageNet data and evaluated their capabilities and limitations in predicting 3 different skin disease types. They included VGG16, ResNet50, Xception and InceptionResNetV2. This means we are can initialize these 4 CNNs with weights obtained from the ImageNet training process that learned how to classify 1,000 different objects consisting of cats, dogs, vehicles and sailboats etc. For the 2017 ISIC Skin Cancer Classification Challenge, there are only 3 different classes, therefore, we need to replace the top layer. In Keras [10], we can load the ResNet50 model without the top layer using the following statements:

```
from keras.applications.resnet50 import ResNet50
from keras.applications.resnet50 import preprocess_input
img_width, img_height = 224,224
input_shape=(img_width, img_height, 3)
img_input = Input(shape=input_shape)
base_model = ResNet50(include_top=False, weights='imagenet', input_tensor=img_input)
```

It is important to note that using the proper $precprocess\_input$ function to rescale input images is critical in generating a good result. A new 4-layer *top_model* can be built with statements below, tunable parameters are *kernel_initializer* and *kernel_regularizer*.

```
top_model = Sequential()
top_model.add(Flatten(input_shape=base_model.output_shape[1:])) # for fine tuning
# or top_model.add(Flatten(input_shape=train_data.shape[1:])) # for using extracted
features
top_model.add(Dense(1024, activation='relu',
                      kernel_initializer='he_normal',
                      kernel_regularizer=regularizers.l2(reg_fac)))
 top_model.add(BatchNormalization())
 top_model.add(Dense(num_class, activation='softmax'))
```

There are 2 different ways to link *base_model* and *top_model* together. The first option is to use *base_model* to predict a 4-dimensional tensor which can be considered as features extracted from input images. This 4-dimensional tensor can be output to a storage device or stored in memory and input to *top_model* to predict the probability of occurrence for each skin disease type. The second option is to use the Keras Model API to combine *base_model* and *top_model* together in the following manner:

```
model = Model(inputs=base_model.input, outputs=top_model(base_model.output))
```

In this case, for layers that are not being trained, we will need to freeze them using the following statements:

```
for layer in model.layers:
       layer.trainable = False
for layer in model.layers[start_layer:]:
       layer.trainable = True
```

Both options have been exercised in this investigation. I used option 1 to train *top_model* and option 2 for fine tuning. Option 1 is a lot more computational efficient and is ideal for the purpose of optimizing hyper parameters such as learning rate, batch size and weights decay factor (*aka* regularization factor).

As indicated earlier, the training data are highly imbalance and I compensated this by assigning a weighting vector to the *class_weight* parameter in the *fit* function as shown below:

```
history = model.fit(train_data, t_labels,
          epochs=nb_epoch, batch_size=batch,
          validation_data=(validation_data, v_labels), class_weight=weights,
          callbacks=callbacks_list,
          verbose=0)
```

This weighting vector is determined by the *sklearn* function *utils.class_weight.compute_class_weight*

```
sklearn.utils.class_weight.compute_class_weight('balanced', classes, y)
```

where *classes* contains unique class labels and *y* is the class label for each training image. It is a vector which has a length equals to the number of training images.

The analysis pipeline is shown in Figure 15 and implemented in 4 steps. Step 1 is the preprocessing step which crops, resizes and augments the original images with new ones that have been rotated or flipped. Step 2 extracts image features using the ImageNet trained weights and the penultimate layer of chosen *base models* in Keras. Step 3 trains the *top models* and stack them together to form an ensemble. More specifically, the predicted probabilities from each *top_model* are stacked column wise to form a new matrix which I refer to as level-1 model. This matrix is then used as an input to train a Logistic Regression model [16] for predicting the labeled validation data and I call this the level-2 model. Once the level-2 model is trained, we can apply this to level-1 predictions made for testing data. The code to carry out this operation is shown below, C is a regulation parameter that requires tuning. This approach has less stringent memory requirement. I was able to use a batch size as high as 512 and complete a training session in less than 20 seconds.

```
from sklearn.linear_model import LogisticRegression
level1_features = np.hstack( ([pr[m] for m in models]) )
lgr=LogisticRegression(C=0.11)
lgr.fit(level1_features,validation['labels'])

level1_test_features = np.hstack( ([pr[m] for m in models]) )
lgr_probs = lgr.predict_proba(level1_test_features)
```

The quality of predicted results is measured by the $ROCAUC$ value. To calculate this value, I need to covert the predicted probabilities to TP and FP using a $sklearn$ function shown below:

```
from sklearn.metrics import roc_curve, auc
def calc_roc(probs,label,prob_loc=0,pos_label=0):
    fpr, tpr, thresholds = roc_curve(label, probs[:,prob_loc], pos_label)
    roc_auc = auc(fpr, tpr)
    return tpr,fpr,roc_auc
```

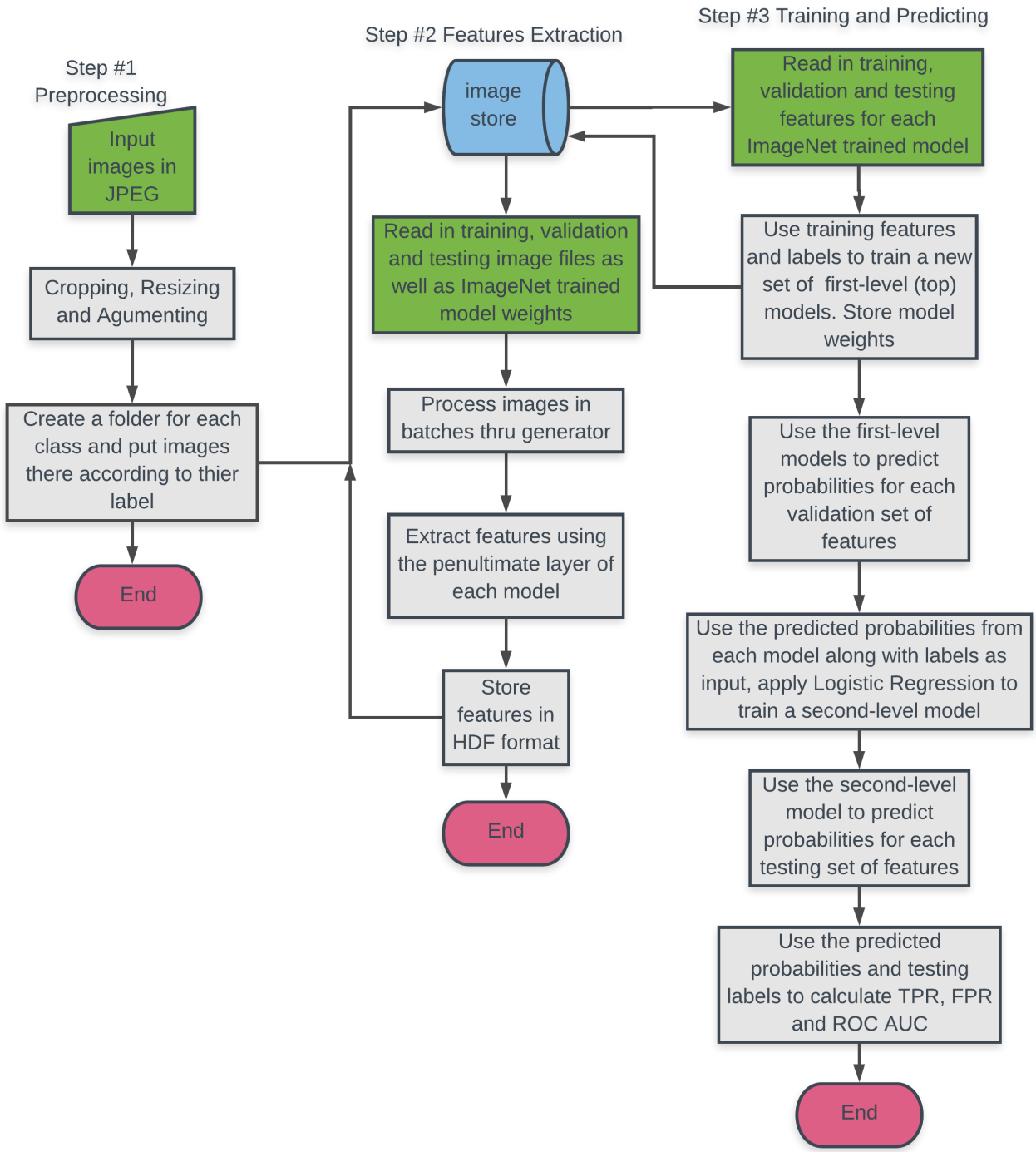


Figure 15: Pipeline for data preparation, features extractions and training of a stacking solution

The fine tuning process, shown in Figure 16, for predicting cancer is more elaborate than that for CIFAR-10, it is more iterative as some of the hyper parameters will require frequent adjustments to reach numerical convergence. It also has a higher demand for computer memory. For instance, VGG16 has over 40 million trainable parameters and requires 6.5 GB of memory, ResNet50 is even more challenging with over 126 millions trainable parameters and 9.4 GB of memory. With these stringent requirements I could only use a batch size of 32 with the current 11-GB GPU. I also discovered turning on multi-processing is paramount in feeding data to the GPU memory and keeping it busy to achieve a higher utilization rate. My computer has a 8-core Intel i7-7700K CPU, GeForce GTX 1080 Ti/PCIe/SSE2 graphics card, 16 GB of memory and 512 GB of

NVMeSSD storage. Figure 17 shows the level of resource utilization for the VGG16 fine tuning process. It demonstrates the benefit of multi-processing in keeping the GPU utilization rate at 95%.

I made an attempt to conduct fine tuning with ResNet50, but due to its use of Batchnorm layers and how the Keras framework is setup to use mini-batch statistics to do normalization when they are frozen [17]. I could not obtain consistent results even with the recommendation provided, therefore, I focused solely on using VGG16 for fine tuning.

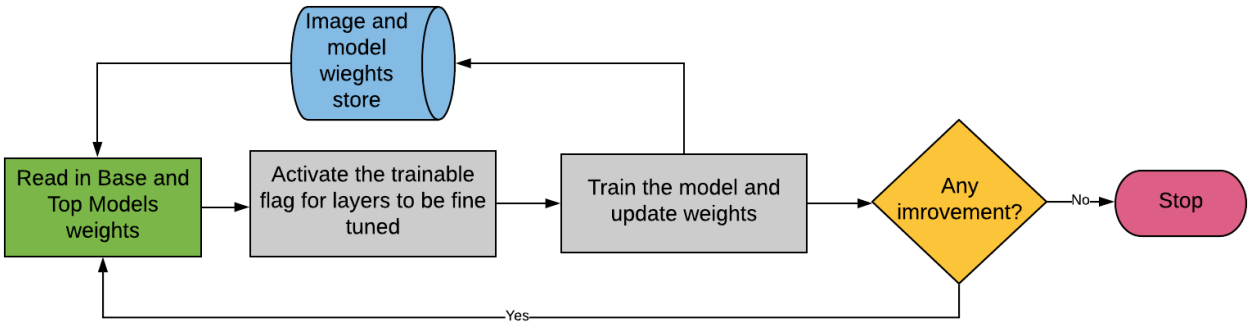


Figure 16: Pipeline of the CNN fine turning process.

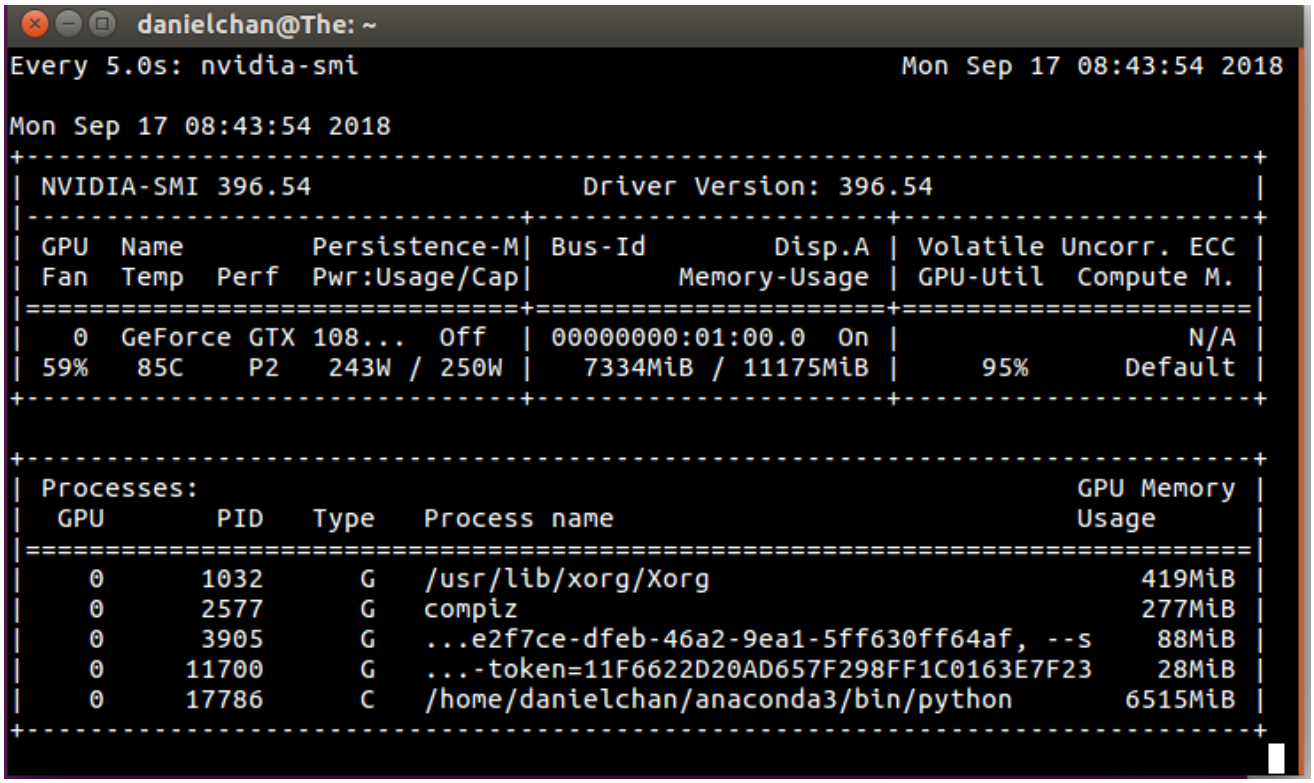


Figure 17. A screen shot showing the GPU memory utilization for fine tuning the VGG16 model. GPU consumption rate is at 95% indicating a high computing efficiency

Results

Model Evaluation and Validation

I started the evaluation process by collecting the 5-fold cross-validation results for all the CNN models selected. Five folds were selected due to a relatively small training dataset; at $2,000$ samples, each fold will have approximately 94 Melanoma, 345 Nevus and 61 Seborrheic Keratosis images.

Figure 18 shows the unique convergence characteristic of the cyclic learning rate approach which supports an unusually high learning rate of 0.1 with a 512 batch size and 1×10^{-4} weight decay factor. Neither numerical convergence nor overfitting (sign of increasing validation loss) was experienced. The 5-fold simulation took 143 seconds and converged to a steady state, as shown in Figure 19, in less than 30 epochs. To understand the impact of using a small batch size, I lowered the value to 32. In order to maintain numerical convergence, I had to decrease the upper bound learning rate to 0.01 (still a relatively high learning rate) which resulted in a higher computational time of 381 seconds and slightly higher losses. However, the AUCs for both runs are nearly identical. This confirms the robustness of cyclic learning algorithm.

Figure 20 shows the 5-fold cross validation results which predicted Melanoma AUCs between 0.71 and 0.80, whereas the Seborrheic Keratosis AUCs vary between 0.86 and 0.89. The results produced by other models exhibit a similar level of variation, except for XCP whose standard deviation is 2 times higher (0.8 vs. 0.4). They are summarized in Figure 21 which demonstrates ResNet50 has the best overall predictive capability.

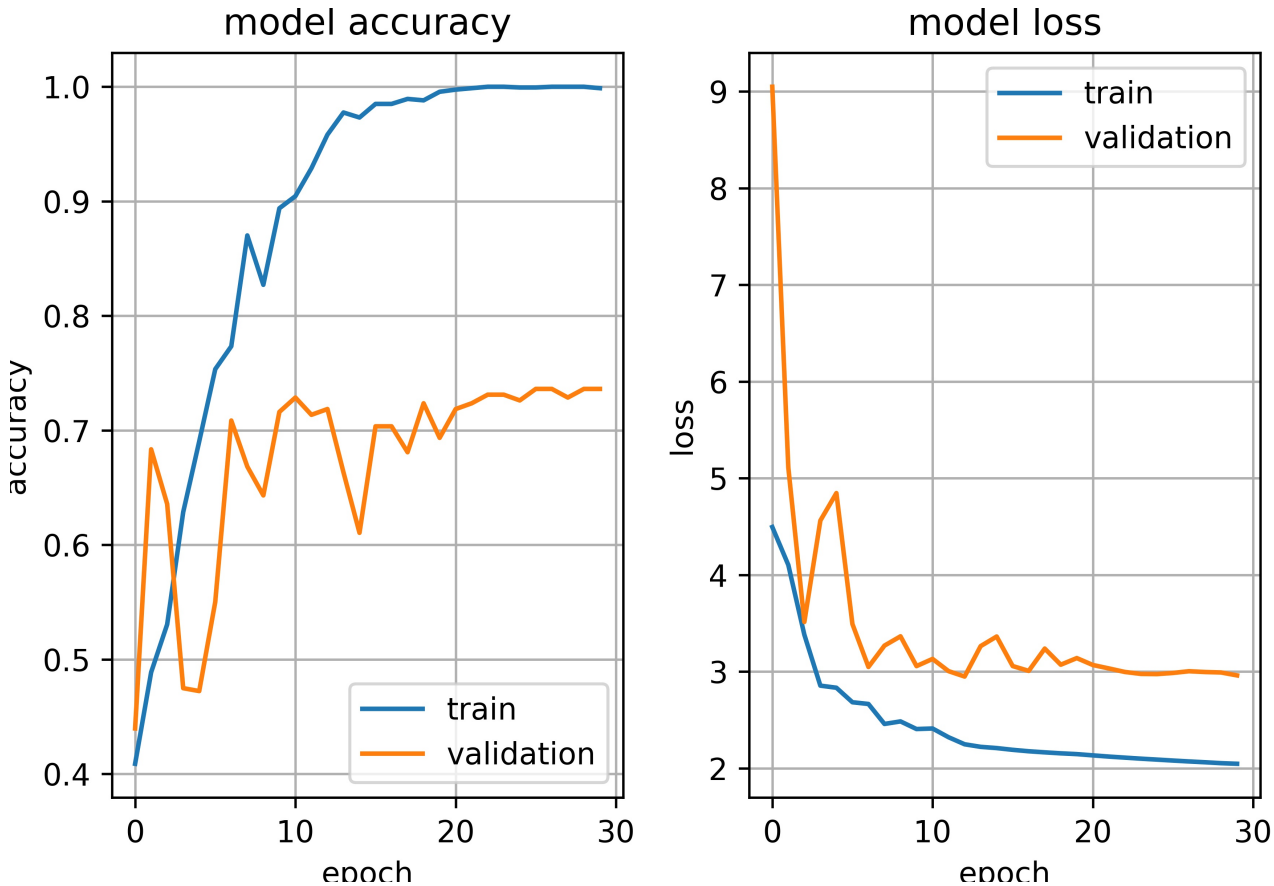


Figure 18: Convergence rate of the ResNet50 *top model*, batch size=512, upper bound learning rate=0.1, lower bound learning rate=1.0×10^{-3}, weight decay=1.0×10^{-4}, training size=$2,000$. Time taken=143 seconds.

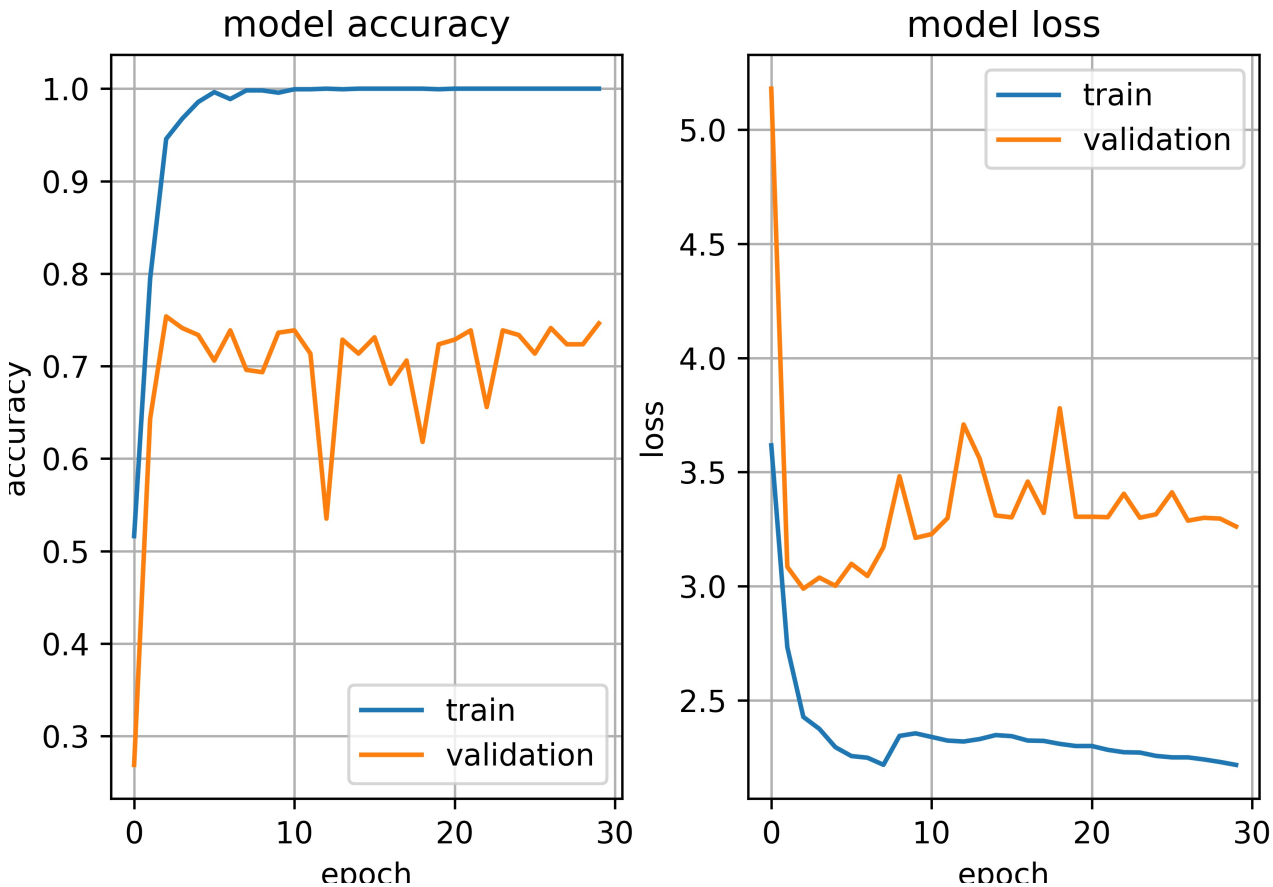


Figure 19: Convergence rate of the ResNet50 *top model*, batch size=32, upper bound learning rate=0.01, lower bound learning rate=1.0×10^{-3}, weight decay=1.0×10^{-4}, training size=$2,000$. Time taken=381 seconds.

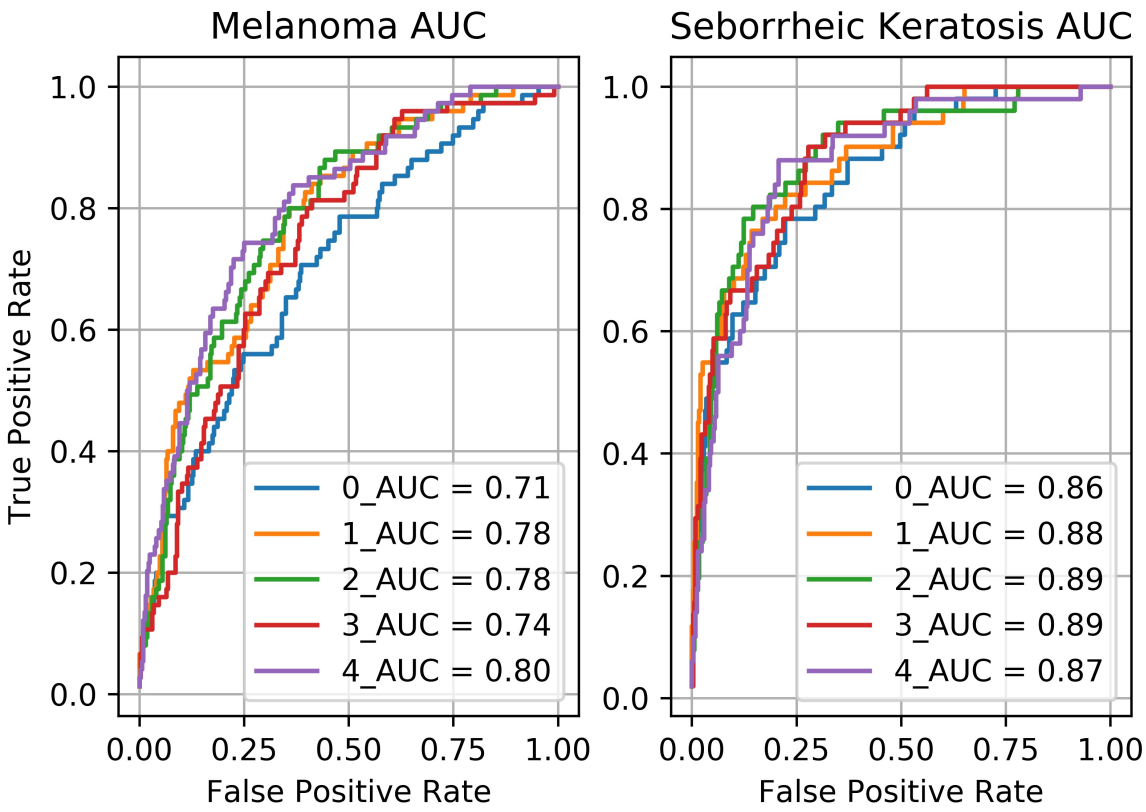


Figure 20: Predictive capability of the ResNet50 *top model* measured by two AUC scores, training sample size=$2,000$,batch size=512, upper bound learning rate= 0.1, weights decay=1.0×10^{-4}, weight vector of {0: 1.7825311942959001, 1: 0.4859086491739553, 2: 2.6246719160104988} applied to compensate for class imbalance. Each line represents one 5-fold run.

Model Name	Melanoma AUC - mean	Melanoma AUC - std	SB AUC - mean	SB AUC - std
VGG16	0.75	0.04	0.83	0.03
RES50	0.76	0.03	0.88	0.01
IRV2	0.75	0.04	0.84	0.02
XCP	0.68	0.08	0.83	0.02

Figure 21: 5-fold cross-validation results for 4 CNN models. ResNet50 has the best overall performance, with XCP being the least effective, especially in predicting Melanoma

The effect of image augmentation is investigated next. I added $1,000$ transformed images to each class and brought the total number of training samples to $5,000$. Five-fold cross-validation for ResNet50 in Figure 22 shows a significant improvement in predicted AUCs with both being in mid to high 0.9s.

When I applied the ResNet50 model trained with this augmented dataset to the test data, the outcome is not as favorable, both predicted AUCs dropped to the 0.8s as shown in Figure 23. Impact of using a smaller training dataset is shown in Figure 24 which reveals only a marginal decrease in AUCs at 0.01.

These results indicate that further effort on training the *top_model* would be unlikely to yield competitive predictions. Therefore I will turn to fine tuning next.

Unlike the CIFAR-10 benchmark case, conducting fine tuning for skin cancer prediction required a lot more effort. I could not fine tune the layers all at once, instead, a block-by-block approach was needed. I attribute this to the fact theses skin disease images look so much different from those $1,000$ objects in CIFAR-10 such that weights at the top layers would have to go through a significant evolution to capture gross image features and then cascading them down to finer details toward the front of the neural network.

For the VGG16 network, I started with the *top_model* (layer 19) , then followed the below sequence:

1. trained layers 11 through 19 (last 2 convolution blocks), initialized with weights from *top_model*
2. trained layers 7 through 19 (last 3 convolution blocks), initialized with the weights trained in step 1
3. trained layers 1 through 19 (full stack), initialized with weights trained in step 2

The fine tuning results for VGG16 on a $2,000$ training samples are shown in Figures 25 and 26, they predicted higher AUCs than the case where only the *top_model* had been trained but lower than the stacked ensemble scores. They also demonstrate for a relatively small training data set, fine tuning the last 2 blocks (i.e. layers 11 through 19) is more appropriate than fine tuning the entire neural network as the AUC actually dropped by 0.02 for having done so. Therefore, trying to fine tune the entire CNN with a small dataset is not a recommended solution.

The results for the $5,000$ samples dataset are more promising. As I progressively fine tuned the entire VGG16 network, the Melanoma AUC went up to 0.83 which yields a FP of 0.25 at 0.8 TP. **This would have been the top score in the 2017 ISIC Challenge and surpasses an average dermatologist.** From a computational perspective, the entire effort took about 150 epochs at 80 seconds each for a total of 3.3 hours.

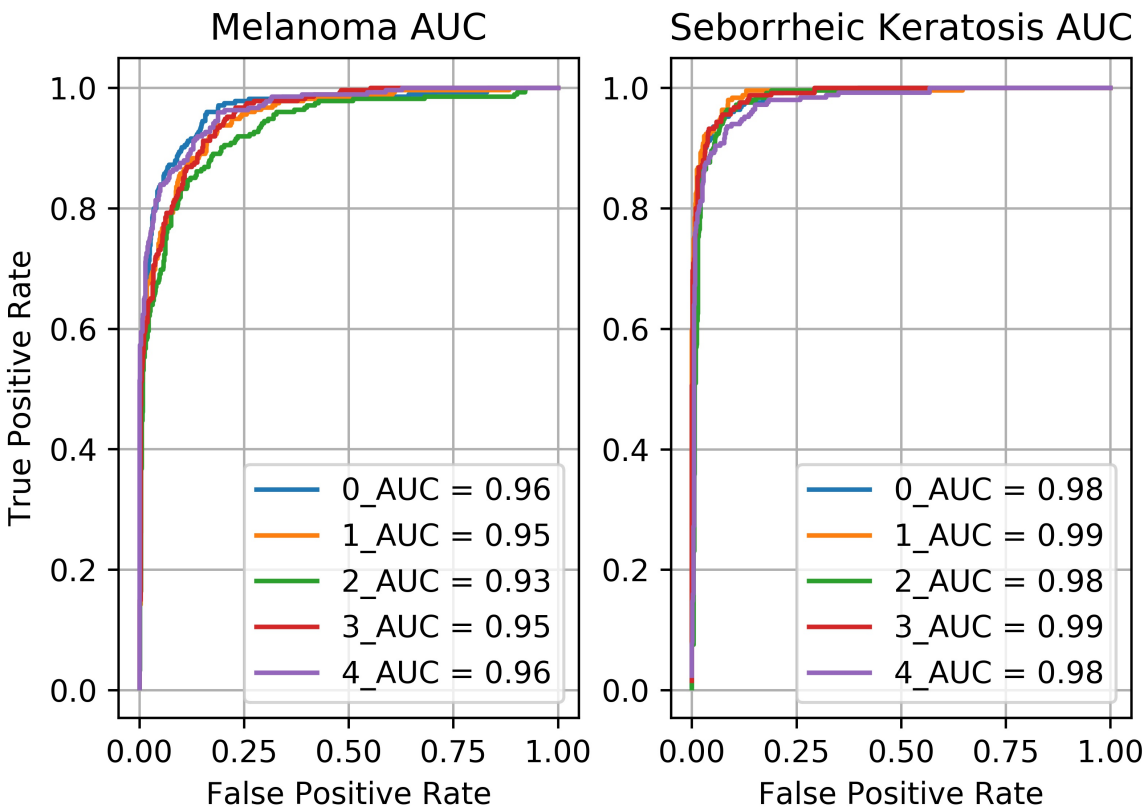


Figure 22: Effect of image augmentation, 5-fold cross-validation for RES50 shows a significant improvement in predicted AUCs with both in mid to high 0.9s

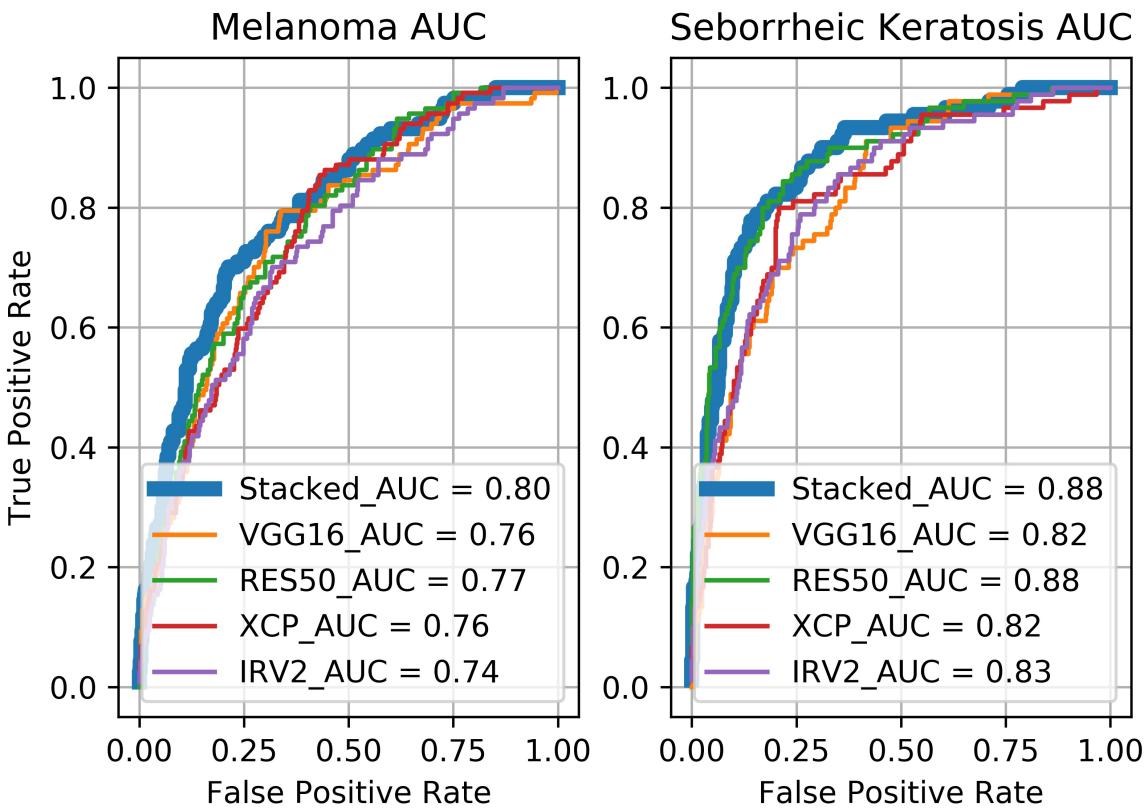


Figure 23: Predicted results of the stacked ensemble with the augmented $5,000$ samples training dataset. The ensemble AUC is higher than any of the individual model prediction, demonstrating its effectiveness.

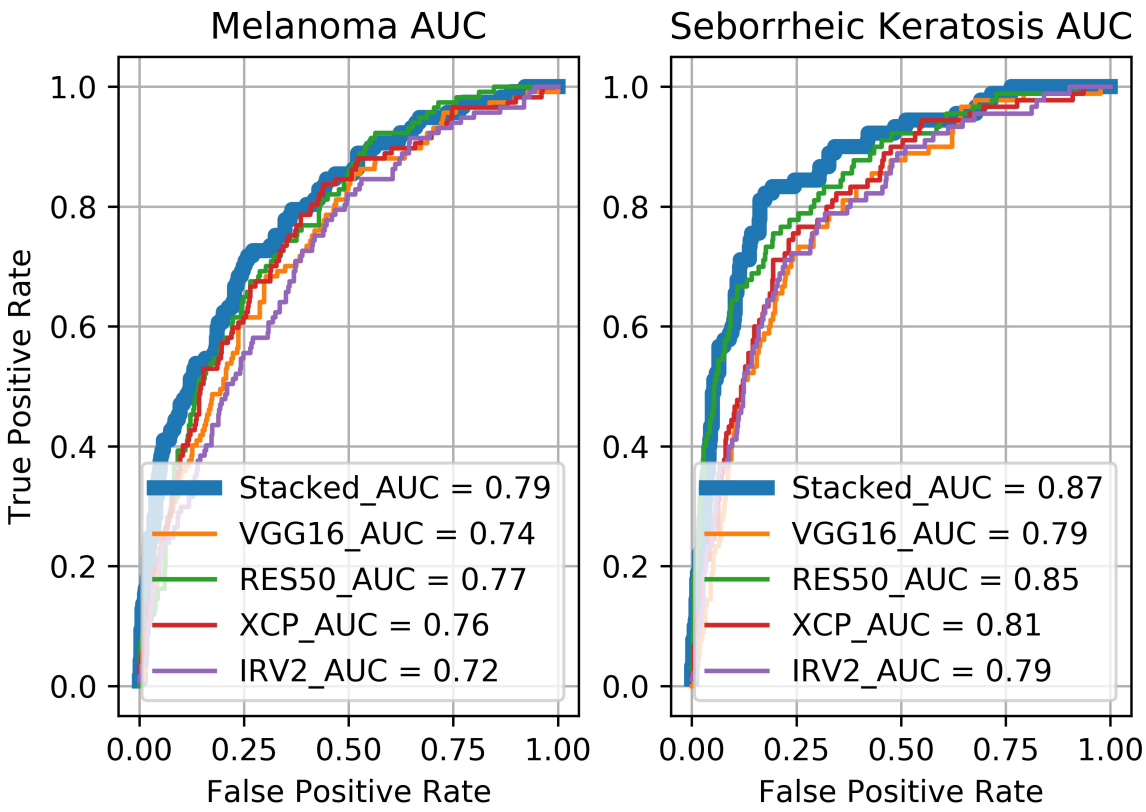


Figure 24: Predicted results of the stacked ensemble with the original $2,000$ samples training dataset. Predicted ensemble AUCs are slightly lower than the $5,000$ samples case.

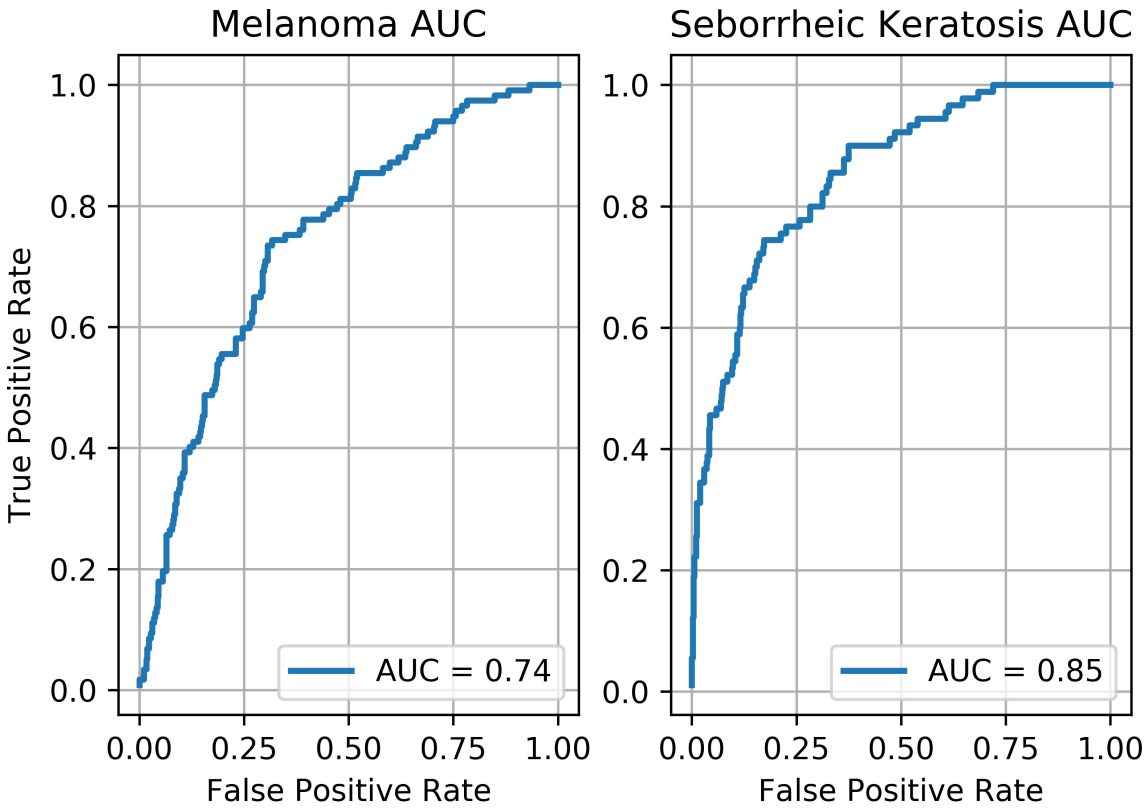


Figure 25: VGG16 fine tuning result for layers 11 through 19, i.e. the last 2 blocks of the neural network. sample size=$2,000$, lower learning rate=1×10^{-4}, upper learning rate=1×10^{-3}, number of steps per learning rate cycle=16, batch size=32, weight decay factor=7×10^{-4}.

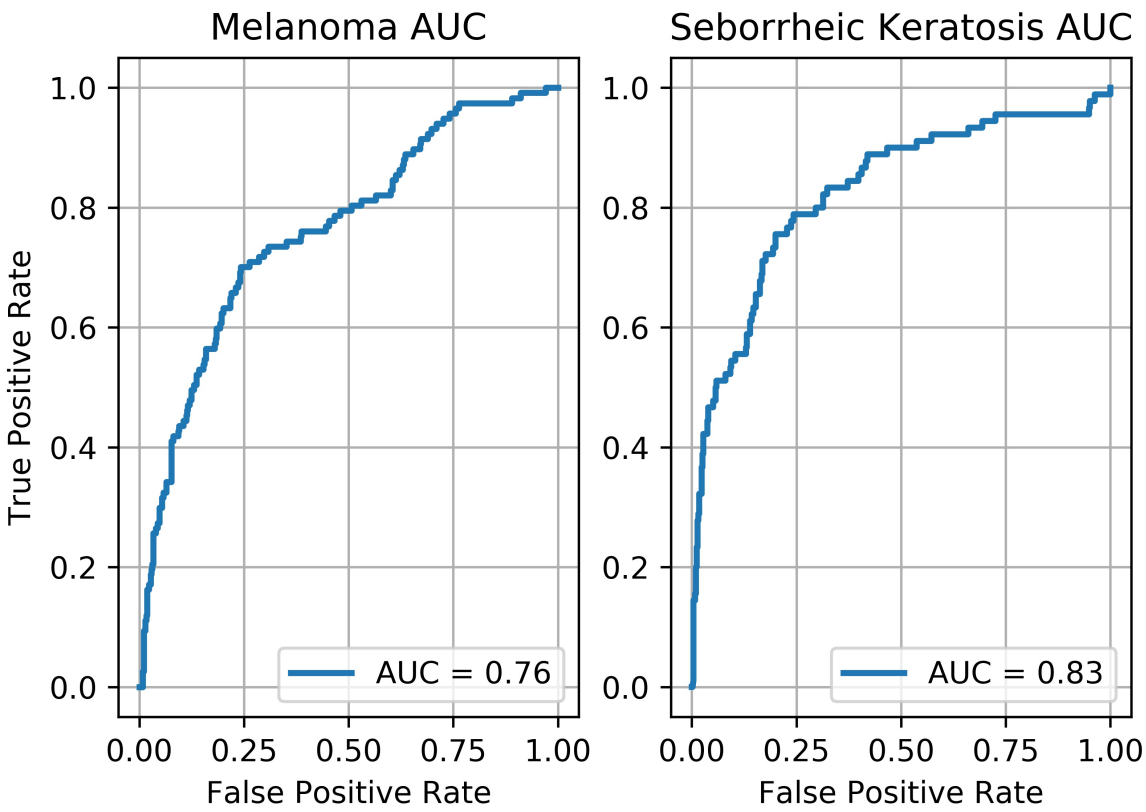


Figure 26: VGG16 fine tuning result for the entire neural network. sample size=$2,000$, lower learning rate=1×10^{-4}, upper learning rate=1×10^{-3}, number of steps per learning rate cycle=16, batch size=32, weight decay factor=7×10^{-4}.

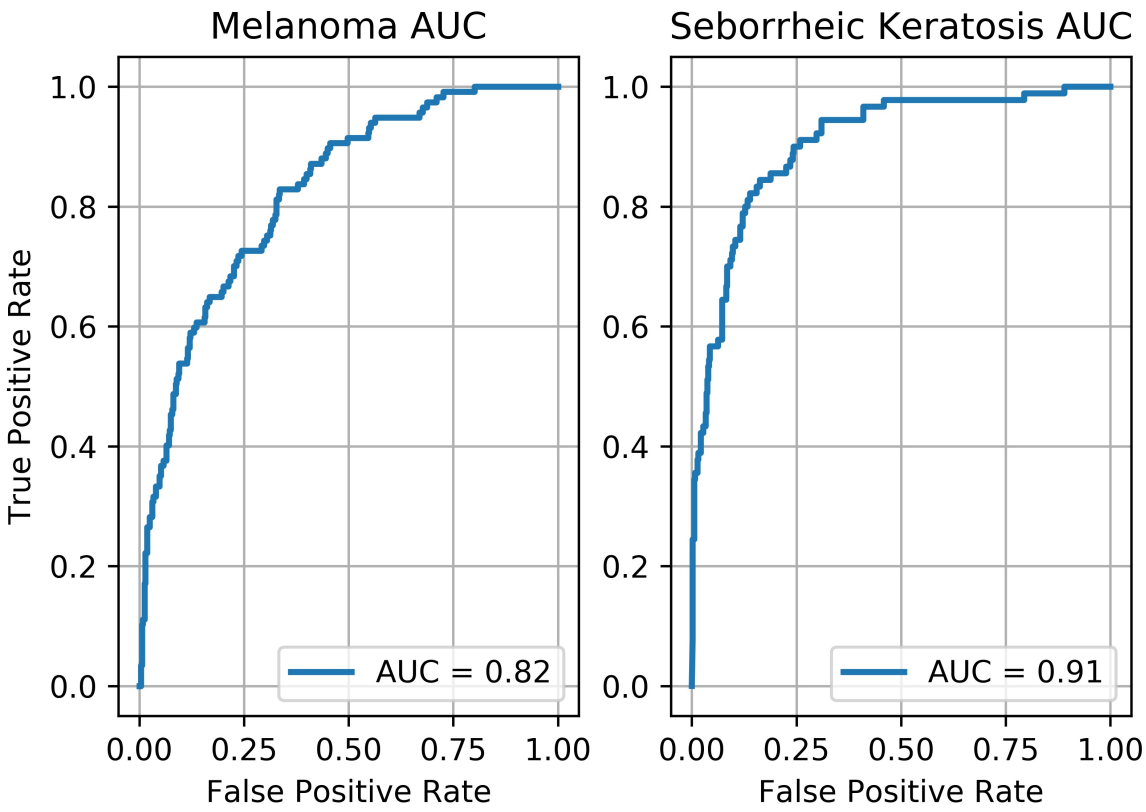


Figure 27: VGG16 fine tuning result for layers 7 through 19, i.e. the last 3 convolution blocks of the neural network. sample size=$5,000$, lower learning rate=1×10^{-4}, upper learning rate=1×10^{-3}, number of steps per learning rate cycle=16, batch size=32, weight decay factor=7×10^{-4}.

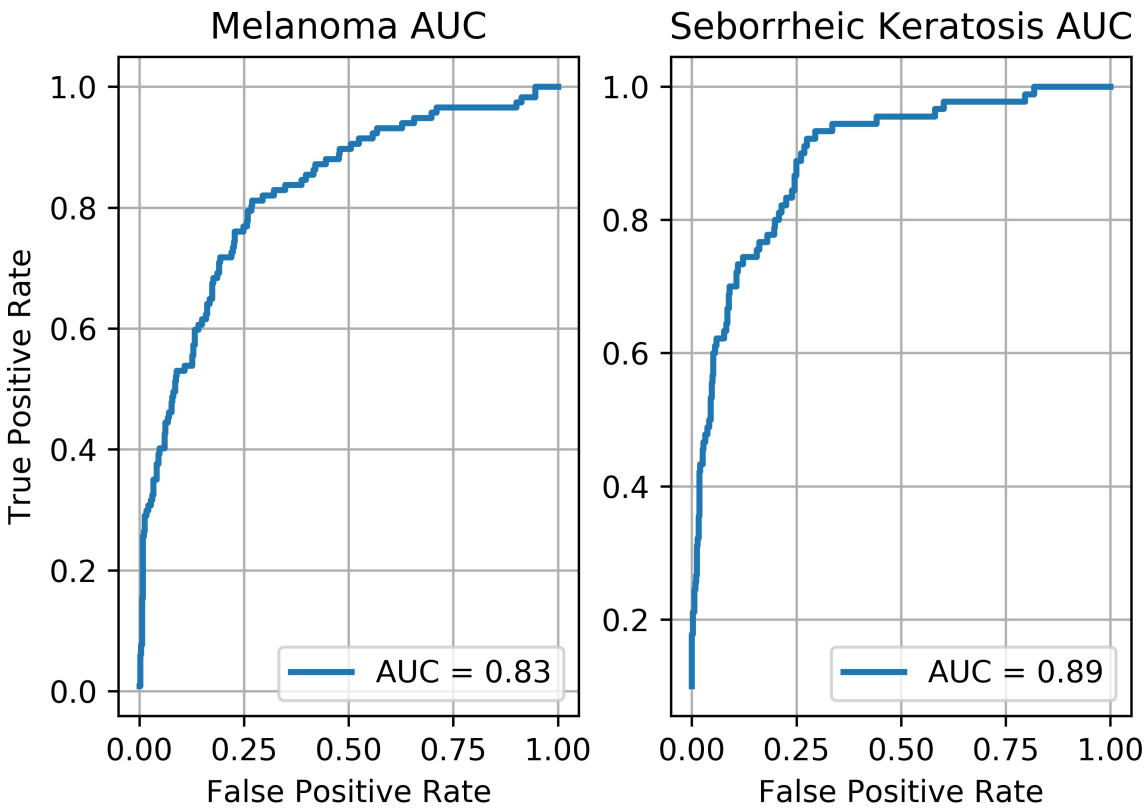


Figure 28: VGG16 fine tuning result for the entire model, sample size=$5,000$, lower learning rate=1×10^{-4}, upper learning rate=1×10^{-3}, number of steps per learning rate cycle=16, batch size=32, weight decay factor=7×10^{-4}.

Conclusion

I have explored 3 different methods in using Deep CNN for cancer detection. First one is a simple bolt-on method where the top layer of a fully trained model is replaced by couple of dense layers. Second one is a 2-level method where an ensemble is constructed and trained with predictions made by 4 different networks. The third method trains the entire VGG16 network initialized with weights trained for $1,000$ ImageNet objects. The quality of prediction improves as we go from method 1 to 3. The degree of difficulty and computational resources required also increase from 1 to 3.

It has been a fruitful endeavor, as I have achieved the objective in identifying a viable solution for skin cancer detection. Here is how I came to this conclusion. As I compare the results from this investigation to the top-3 submissions received by the 2017 ISIC Challenger organizer in Figure 29 for detecting malignant skin disease, I plotted the current predictions directly on the published graph for Melanoma ROC. Square shapes are from fully tuned VGG16 model and diamond shapes are from stacked ensemble results using $5,000$ training images. When compare to the golden color line, **the prediction from the fully tuned VGG16 model would have been a top entry in 2017**. In addition, the current approach is a lot simpler than those submitted entries. It does not require access to external data sources and an elaborate pipleline with custom codes. It requires good image preprocessing strategy and reliable Deep Learning framework such as Keras/Tensorflow that can easily reproduce the current results.

I believe the key enablers for this accomplishment are:

- cyclic learning with its unique rapid convergence rate allowed me to use higher learning rates that speed up the training process, especially with fine tuning.
- Multi-processing saved the day by feeding data to the GPU memory in a fast enough fashion to keep the utilization rate at 95% and produced short turnaround times that allow extensive parametric studies and hyper parameter optimization.

Possible improvements are:

- As indicated above, deep CNNs have high demand for memory (about 10 GB for 64-bit accuracy), most game-oriented GPU cards have at most 11GB, so the batch size could be severely limited, may be in the 16 to 32 range. For cyclic learning, the higher the batch size is, the higher the learning rate one can use, so for future work, I would consider lowering the accuracy to 32-bit floating point to facilitate larger batch sizes for faster speed.
- ResNet50 looks promising as the basis for Methods 1 & 2, in order to user it for fine tuning, it will require some customization to change the process in update Batchnorm layers during training, this will be investigated as a way to further improvement the cancer detection capability.

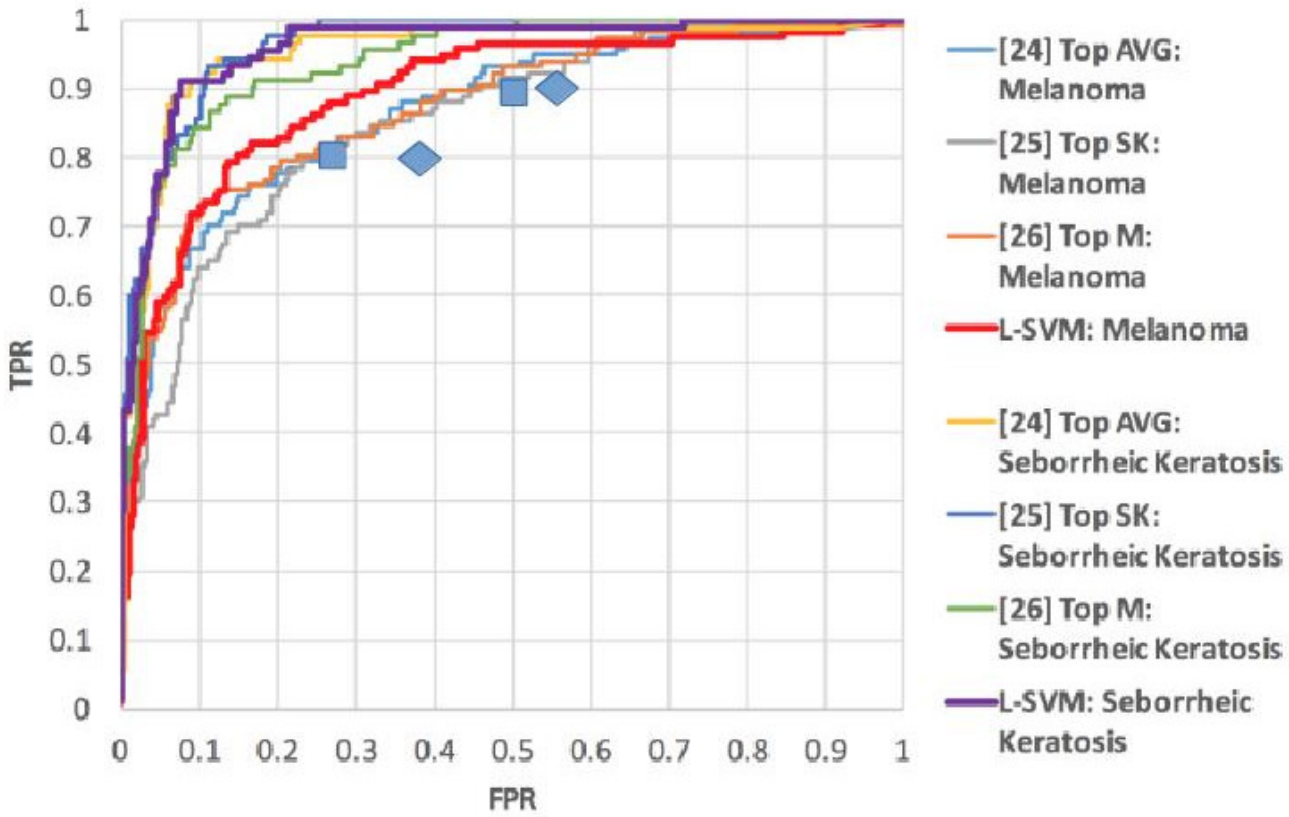


Figure 29: ROC curves for top 3 submissions to the 2017 ISIC Challenge [7]. The best Melanoma FPR is 0.25 at a TPR of 0.8. Square symbols represent VGG16 fine tuned results. Diamond symbols represent Stacked Ensemble results. Comparison is made to the golden color line.

References

[1] Skin Cancer Facts and Statistics, https://www.skincancer.org/skin-cancer-information/skin-cancer-facts

[2] How to spot cancer? https://www.cancer.org/latest-news/how-to-spot-skin-cancer.html

[3] The cost of a pocket dermascope in Amazon as of April 28, 2018,https://www.amazon.com/3Gen-DermLite-DL100-Dermatology-Dermascope/dp/B000X2INPY/ref=sr_1_1?ie=UTF8&qid=1524953402&sr=8-1&keywords=dermatoscope

[4]How does a dermatoscope work? http://www.dermatoscope.info/how-does-a-dermatoscope-work

[5] International Skin Imaging Collaboration (ISIC) Challenges https://challenge.kitware.com/ - challenges

[6] Skin Lesion Analysis toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), hosted by the International Skin Imaging Collaboration (ISIC) https://www.arxiv-vanity.com/papers/1710.05006/

[7] A second set of eyes - Using computers to aid melanoma detection, by Michael Marchetti, Oct 4, 2017, IBM Blog Research https://www.ibm.com/blogs/research/2017/10/computers-to-aid-melanoma-detection/

[8] Sensitivity and specificity, Wikipedia, https://en.wikipedia.org/wiki/Sensitivity_and_specificity

[9] What you wanted to know about AUC, Sept 19, 2013, http://fastml.com/what-you-wanted-to-know-about-auc/

[10] Keras Documentation, https://keras.io/

[11] A Disciplined Approach to Neural Network Hyper-Parameters, by Leslie N. Smith US Naval Research Laboratory, https://arxiv.org/pdf/1803.09820.pdf

[12] CIFAR-10, https://www.cs.toronto.edu/~kriz/cifar.html

[13] CIFAR-10 Benchmark Results, http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130

[14] DAWNBench: An End-to-end Deep Learning Benchmark and Competition, https://dawn.cs.stanford.edu/benchmark/index.html#cifar10-train-time

[15] Augmentor, https://augmentor.readthedocs.io/en/master/

[16] Scikit-learn, Machine Learning in Python, http://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

[17] Change BN layer to use moving mean/var if frozen https://github.com/keras-team/keras/pull/9965

[18] MobileNets: Open-Source Models for Efficient On-Device Vision https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html

[19] Introduction to Convolutional Neural Networks, Stanford University, March, 2018 https://web.stanford.edu/class/cs231a/lectures/intro_cnn.pdf

[20]Stanford University CS231n Convolution Neural Networks for Visual Recognition, http://cs231n.github.io/convolutional-networks/