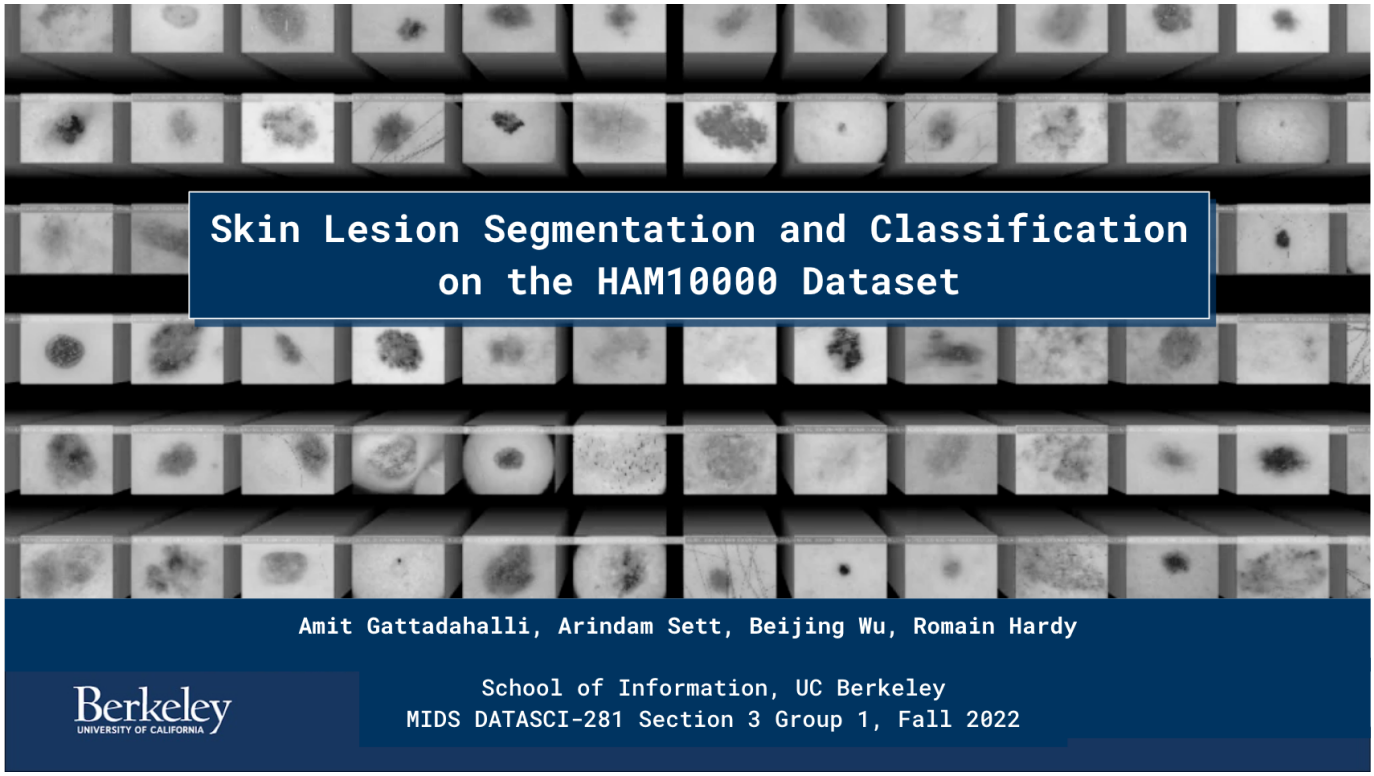



*Link to presentation slides (https://docs.google.com/presentation/d/1UY0qRBq2tQJ3aXFt8mUUVKwYYMaLsc2ETp88Briumc0/edit?usp=sharing)*


## Abstract

Skin cancer is the most prevalent type of cancer. The specific type of skin cancer a patient has is important because it influences treatment options and prognosis. Detecting dangerous skin conditions, such as malignancy, requires the identification of pigmented skin lesions. Recently, image detection techniques and computer classification capabilities have been shown to improve the accuracy of skin cancer detection. The rapid increase in the popularity of telemedicine, coupled with new advances in diagnostic artificial intelligence (AI), has prompted the need to consider the opportunities and risks of inserting AI-based support into new paradigms of care. In this report, we build upon recent advances in the accuracy of image-based AI for skin cancer diagnosis. Specifically, we evaluate various classical and deep learning models for skin lesion segmentation and classification. Our best segmentation model is a U-Net model that achieves a per-pixel accuracy of $97.1\%$ on out-of-sample data, while our best classification model is a vision transformer that achieves a macro $F_1$ score of $81.9$ on a test set.


## Introduction

A skin lesion is a growth or change in the skin that is abnormal relative to the surrounding skin. Primary skin lesions are abnormal skin conditions that can develop over time or be present at birth, while secondary skin lesions can develop from primary lesions that have been exacerbated or altered. Regardless of how they appear, skin lesions pose a great risk to patients, since they can indicate the presence of malignancy and require surgical removal.

Dermoscopy is a tool-assisted technique for examining malignant and benign skin lesions. In 1994, Binder et al. showed that a neural network trained on dermascopic images could match and perhaps outperform expert analysis. However, they were severely limited by the size of their dataset (which only contained $200$ images, $100$ for training and another $100$ for testing) and the capabilities of their computational resources.

Nearly three decades later, machine learning is an ever-more attractive approach for analyzing skin lesions. Accurate lesion models could lead to new diagnostic frameworks that provide better contextual relevance, improve clinical reliability, and assist physicians in communicating objectively. Machine learning models are also cost-effective, a significant benefit in an age of rising healthcare costs.

In the following sections, we evaluate and compare classical models and deep neural networks for segmenting and classiying pigmented skin lesions in dermascopic images.

### Image Segmentation

Image segmentation is the task of separating an image into multiple segments, usually in order to identify objects or regions of interest in the image. This is particularly useful in medical settings, as it allows experts to focus their analysis on relevant portions of an image and ignore irrelevant background pixels. In the case of skin lesion classification, image segmentation is meant to divide an image into two regions, the lesion and the background skin. If a segmentation is available for an image of a skin lesion, we can use it to crop the image and remove most of the background skin. In theory, this should help downstream classification models make accurate predictions by removing extraneous information from the input.

### Skin Lesion Classification

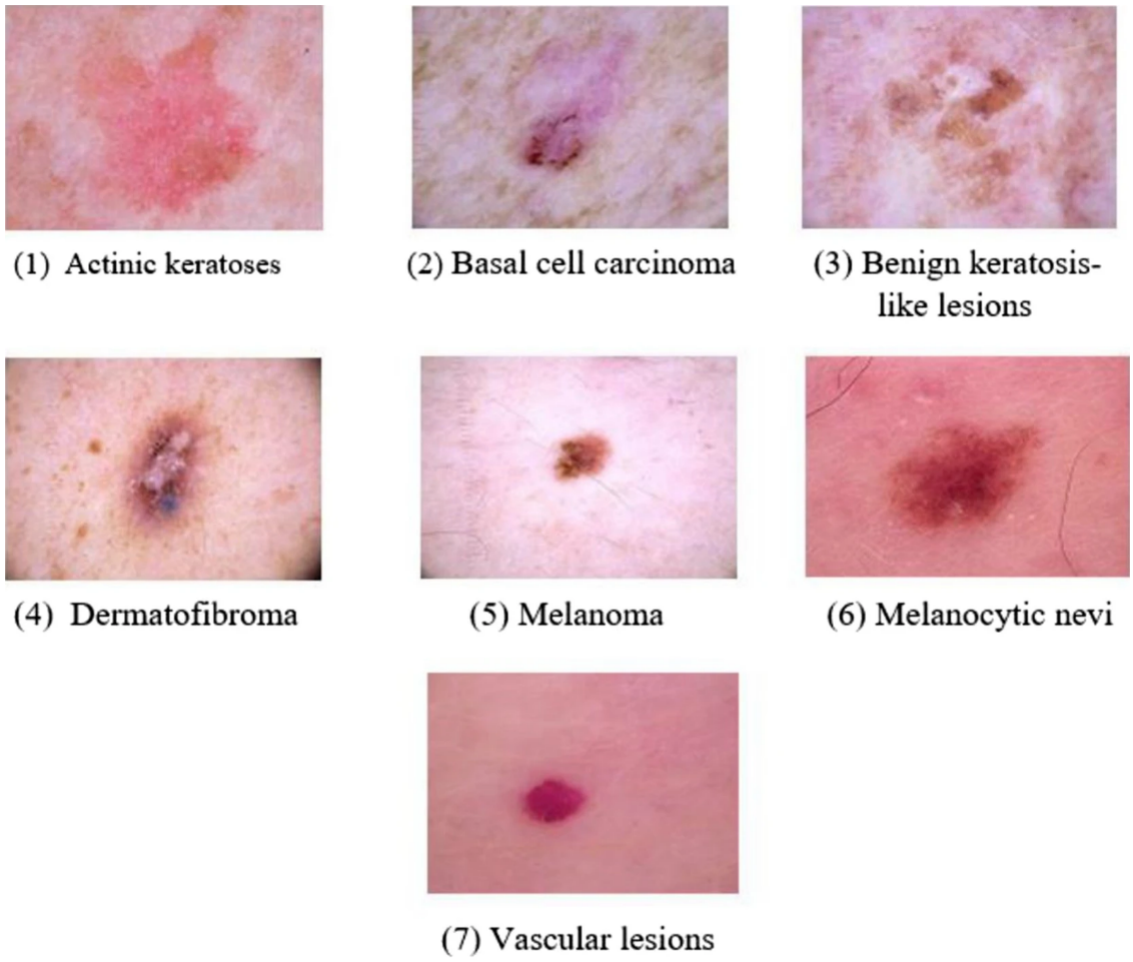
Manually detecting skin lesions is labor-intensive and time-consuming. It requires magnified and illuminated skin images to improve the clarity of spots, and many clinicians rely on their experience. The manual dermoscopy imaging process is prone to error because it requires years of training, vast amounts of visual exploration, and the ability to identify similarities and differences between different skin lesions. In recent times, deep learning has aided in the detection, classification, and diagnosis of diseases through medical image analysis. Dermoscopy produces a significant amount of well-annotated skin lesions images that can be used to train supervised machine learning models to classify, predict, and detect different skin wounds. Therefore, it is anticipated that deep learning can also assist dermatologists the tasks of detecting and classifying skin lesions.

## HAM10000 Dataset

Our work uses the HAM10000 dataset, which consists of $10015$ images of dermoscopic images of pigmented common skin lesions. The dataset images have a resolution of $600 \times 450$ pixels, in three (RGB) channels, and are stored in the JPEG format. They are manually cropped and centered around the lesion, as well as modified for visual contrast and color reproduction.

The patients in the dataset were generally between the ages of $35$ and $70$. Lesion images were divided into four categories by researchers: histopathology, confocal, follow-up, and consensus. In the histopathology category, dermatopathologists diagnosed excised lesions histopathologically, and all images were manually evaluated and confirmed for plausibility by researchers. For the confocal images, researchers used the reflectance confocal microscopy method to confirm the presence of benign keratoses on the face. In the follow-up category, researchers recognized images as evidence of biological benignity if nevi examined with digital dermoscopy showed no changes during three follow-up visits or $1.5$ years. The consensus category consists of normal benign instances with no follow-up or histology, as well as examples in which two experts gave the same benign diagnosis.

Histopathology was used to diagnose more than half of the skin lesions. The back, lower limbs, and trunk are all significantly impacted skin cancer locations, as demonstrated in the dataset's localization distribution. The dataset contains seven diagnostic classes of skin lesions. Figure $1$ depicts a selection of sample images from the HAM10000 dataset for each of the seven classes.




Sample images for the seven skin lesion categories from HAM10000 dataset.

**Figure 1**

Image Credit (https://www.nature.com/articles/s41598-022-22644-9/figures/1)

### Skin Lesion Categories

The following description of diagnostic categories is meant for computer scientists who are not familiar with the dermatology literature.

**AKIEC**
Actinic Keratoses (Solar Keratoses) and Intraepithelial Carcinoma (Bowen's disease) are common non-invasive variants of squamous cell carcinoma that can be treated locally without surgery. Some authors regard them as precursors of squamous cell carcinomas and not as actual carcinomas. There is, however, agreement that these lesions may progress to invasive squamous cell carcinoma – which is usually not pigmented. Both neoplasms commonly show surface scaling and commonly are devoid of pigment. Actinic keratoses are more common on the face and Bowen's disease is more common on other body sites. Because both types are induced by UV-light the surrounding skin is usually typified by severe sun damaged except in cases of Bowen's disease that are caused by human papilloma virus infection and not by UV. Pigmented variants exist for Bowen's disease20 and for actinic keratoses, and both are included in this set.

**BCC**
Basal cell carcinoma is a common variant of epithelial skin cancer that rarely metastasizes but grows destructively if untreated. It appears in different morphologic variants (flat, nodular, pigmented, cystic)

**BKL**
Benign keratosis is a generic class that includes seborrheic keratoses ("senile wart"), solar lentigo - which can be regarded a flat variant of seborrheic keratosis - and lichen-planus like keratoses (LPLK), which corresponds to a seborrheic keratosis or a solar lentigo with inflammation and regression. The three subgroups may look different dermatoscopically, but we grouped them together because they are similar biologically and often reported under the same generic term histopathologically. From a dermatoscopic view, lichen planus-like keratoses are especially challenging because they can show morphologic features mimicking melanoma26 and are often biopsied or excised for diagnostic reasons. The dermatoscopic appearance of seborrheic keratoses varies according to anatomic site and type.

**DF**
Dermatofibroma is a benign skin lesion regarded as either a benign proliferation or an inflammatory reaction to minimal trauma. The most common dermatoscopic presentation is reticular lines at the periphery with a central white patch denoting fibrosis.

**NV**
Melanocytic nevi are benign neoplasms of melanocytes and appear in a myriad of variants, which all are included in our series. The variants may differ significantly from a dermatoscopic point of view. In contrast to melanoma they are usually symmetric with regard to the distribution of color and structure.

**MEL**
Melanoma is a malignant neoplasm derived from melanocytes that may appear in different variants. If excised in an early stage it can be cured by simple surgical excision. Melanomas can be invasive or non-invasive (in situ). We included all variants of melanoma including melanoma in situ, but did exclude non-pigmented, subungual, ocular or mucosal melanoma. Melanomas are usually, albeit not always, chaotic, and some melanoma specific criteria depend on anatomic site.

**VASC**
Vascular skin lesions in the dataset range from cherry angiomas to angiokeratomas and pyogenic granulomas. Hemorrhage is also included in this category. Angiomas are dermatoscopically characterized by red or purple color and solid, well circumscribed structures known as red clods or lacunes.

Note that the number of images in the dataset does not correspond to the number of unique lesions, because we also have images of the same lesion taken at different magnifications or angles or with different cameras. This should serve as a natural data-augmentation as it shows random transformations and visualizes both general and local features.

## Literature Review

Following the publication of the HAM10000 dataset, several papers have been published on different approaches to segment and classify skin lesions.

The original authors Tschandl, P. et al. (2020) (https://doi.org/10.1038/s41591-020-0942-0) tried three aproaches for lesion classification. The first approach was to use a convolutional neural network (CNN) to provide model-based multiclass probabilities. The second approach was motivated by pre-existing AI solutions for skin cancer diagnosis, which dichotomized the disease categories two classes, benign and malignant. For the third approach, the authors used the CNN to implement a content-based information retrieval machine (CBIR) that supports physicians by searching databases to retrieve similar images with known diagnoses. The authors found that the first approach general generally produced the best results.

Tajeddin et al. (https://doi.org/10.1016%2Fj.cmpb.2018.05.005) identified four different tasks for pre-processing: channel selection, hair and ruler marker inpainting (hair removal), dark corner effect removal (corner detection), and uneven illumination correction. First, they selected the channel in the image that has the most entropy. Next, hairs and ruler marks were detected and inpainted using a DullRazor filter. Because the procurement conditions were the same across all images, they did not need to obtain a new corner mask for each image and could instead use a constant border mask. Homomorphic filtering was used to correct uneven illumination, first by transforming the image from the RGB space to the L*A*B space, then performing homomorphic filtering on the L component, then finally transforming the image back to RGB space.

Shetty et al. (https://www.nature.com/articles/s41598-022-22644-9) identified image features corresponding to the ABCDE's of Melanoma (https://www.healthline.com/health/skin-cancer/abcd-rule-for-skin-cancer#). The color of the lesion image is quantified using a colour histogram, the shape of the lesion is quantified using Hu Moments, and the texture is quantified using Haralick textures. These features were chosen because the color, shape, and texture are properties that dominate in the lesion zone. Furthermore, Shetty et al. (https://www.nature.com/articles/s41598-022-22644-9) discussed horizontal flip augmentations, while Silveira et al. (https://ieeexplore.ieee.org/abstract/document/4786545) proposed several methods for skin lesion segmentation: adaptive thresholding, gradient vector flow, adaptive snake, set methods, and a fuzzy split-and-merge algorithm.

Shetty et al. (https://www.nature.com/articles/s41598-022-22644-9) used various classical machine learning models for lesion classification: decision trees (DT), random forests (RF), support-vector machines (SVM), $k$-nearest neighbors ($k$-NN), logistic regression (LR), Gaussian Naïve Bayes (NB), and linear discriminant analysis (LDA). They evaluated these models in terms of accuracy, precision, recall, and $F_1$ scores. The authors also trained two CNNs, both a custom model as well as a pre-trained InceptionV3 model. The InceptionV3 model achieved an accuracy of $95.14\%$, compared to $95.18\%$ for the custom CNN. Although the models were close in terms accuracy, the custom CNN was smaller and quicker to train than the InceptionV3 model. The authors used the Adam optimizer, a categorical cross-entropy loss function, $150$ epochs, a batch size of $32$, and an initial learning rate of $0.001$.

Tschandl, P. et al. (2020) (https://doi.org/10.1038/s41591-020-0942-0) used a ResNet34 with weights initiated by pretraining on ImageNet data. They used the cross-entropy loss function, with weights corresponding to the relative frequency of the different lesion classes in the dataset. The learning rate was initialized at $0.0001$, with a ten-fold reduction after $3$ epochs of improvement in the validation loss (and a minimum learning rate of $1 \times 10^{-9}$. They used the Adam optimizer and trained for $100$ epochs with early stopping. Images were loaded in batches of $32$, randomly cropped and resized to $224 \times 224$ pixels, and randomly augmented (random $90$-degree rotations, flips, and color, contrast, saturation and hue jitters). At inference time, images were cropped to $80\%$ of their original size and resized to $224 \times 224$ pixels, with minor augmentations.

Most recently, Basak, H. et al. (2022) (https://arxiv.org/abs/2203.14341) proposed the Multi-Focus Segmentation Network (MFSNet), which uses differently scaled feature maps for computing the final segmentation mask given RGB images of skin lesions. The MFSNet uses a Res2Net backbone to obtain features, which are then used in a Parallel Partial Decoder (PPD) module to produce a global segmentation map. In different stages of the network, convolution features and multi-scale maps are used in two boundary attention (BA) modules and two reverse attention (RA) modules to generate the final segmentation mask. MFSNet performs significantly better than several state-of-the-art methods, including double U-Net, U-Net, and SegNet; MFSNet is able to achieve $99.9\%$ sensitivity and specificity on segmentation tasks.

For lesion classification, Lan, Z. et al. (2022) (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9791221) introduced FixCaps, an improved capsule model that adds a large-kernel convolutional layer to the bottom of the network. The kernels are as large as $31 \times 31$, in contrast to the commonly used $9 \times 9$. FixCaps not only reduces the number of convolution kernels at the bottom convolution layer, but also increases fractional max-pooling (FMP) to reduce the size of the initial layer and the cost of dynamic routing in the capsule layer. Results showed that with larger convolution kernels, more picture information is "seen" and better features are learned. In addition, a convolutional block attention module (CBAM) was used to reduce the loss of spatial information caused by convolution and pooling, and group convolution (GP) was used to avoid model underfitting in the capsule layer. FixCaps was able to achieve an accuracy of $96.49\%$ on HAM10000 dataset, despite being less computationally expensive than other state of the art models.

As pointed out in Tschandl, P. et al. (2020) (https://www.nature.com/articles/s41591-020-0942-0), skin lesion classification requires extensive cognitive engagement by medical specialists, as they need to extrapolate a diagnosis from similarities between the test image and other images with known diagnoses. In this study, human raters (including $56\%$ board-certified dermatologists, $25.5\%$ dermatology residents, and $12.6\%$ general practitioners) were invited to diagnose batches of images in the HAM10000 dataset. The accuracy of diagnosis from the human raters was $63.6\%$, compared to $77.0\%$ for the deep-learning models. This showed that insights derived from large-scale neural networks can dramatically improve clinical diagnoses of skin lesions.

## Exploratory Data Analysis

| Class | # Positive |
|---|---|
| AKIEC (Actinic keratoses and intraepithelial carcinoma) | 327 |
| BCC (Basal cell carcinoma) | 514 |
| BKL (Benign keratosis-like lesions) | 1,099 |
| DF (Dermatofibroma) | 115 |
| MEL (Melanoma) | 1,113 |
| NV (Melanocytic nevi) | 6,705 |
| VASC (Vascular lesions) | 142 |

In [3]:
```
# Imports
import pandas as pd
import numpy as np
import cv2
import PIL as pil
import albumentations as A
import matplotlib.pyplot as plt
from scipy.ndimage import convolve
from skimage.color import rgb2hsv, rgb2lab
import mahotas as mh
```

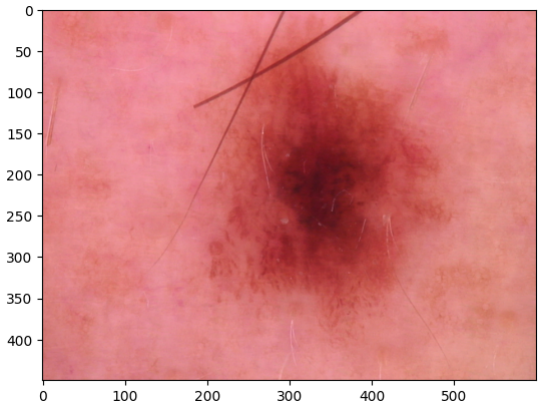
### Sample Image and Mask

#### Sample Image

In [4]:
```
ISIC_0024306 = cv2.imread("report_images/ISIC_0024306.jpg")
print(ISIC_0024306.shape)
plt.imshow(cv2.cvtColor(ISIC_0024306, cv2.COLOR_BGR2RGB))
```

```
(450, 600, 3)
```

Out[4]: <matplotlib.image.AxesImage at 0x12954aac0>

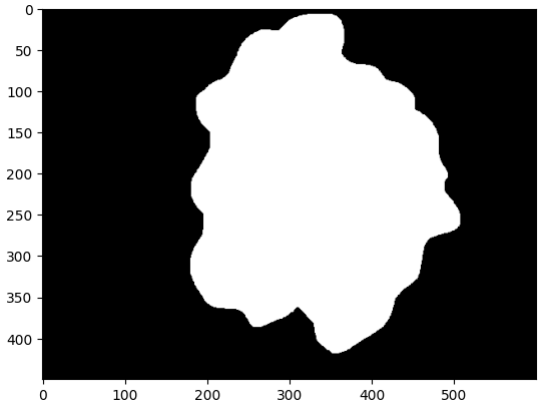



**Mask**

In [5]:
```
ISIC_0024306_mask = cv2.imread("report_images/ISIC_0024306_segmentation.png")
print(ISIC_0024306_mask.shape)
plt.imshow(cv2.cvtColor(ISIC_0024306_mask, cv2.COLOR_BGR2RGB))
```

```
(450, 600, 3)
```

Out[5]: <matplotlib.image.AxesImage at 0x129676af0>




## Manual Feature Extraction

**ABCDE's of Melanoma**

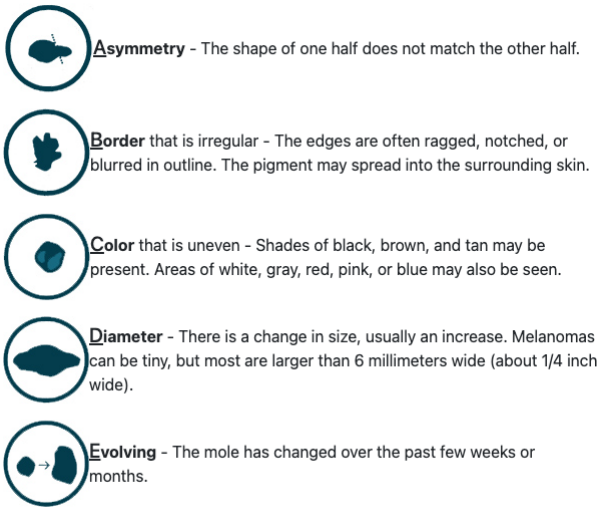


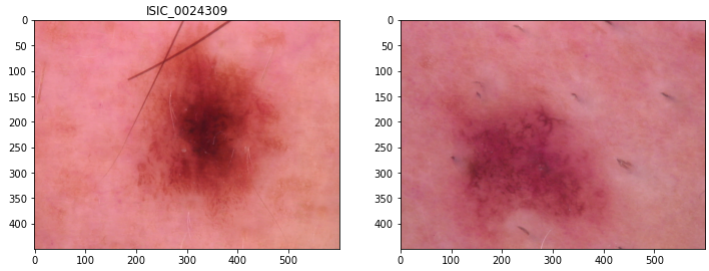

Image Credit (https://moles-melanoma-tool.cancer.gov/#/)

**Assymetry**

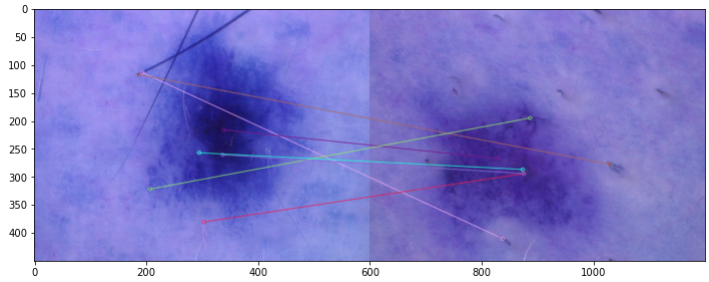
***SIFT-Based Similarity***

SIFT stands for Scale-Invariant Feature Transform. It is used to compare images and check for points of similarity between them.

In [4]:
```
ISIC_0024309 = cv2.imread("report_images/ISIC_0024309.jpg")
fig, ax = plt.subplots(1, 2, figsize = (12, 6))
ax[0].imshow(cv2.cvtColor(ISIC_0024306, cv2.COLOR_BGR2RGB))
ax[0].set_title("ISIC_0024306")
ax[1].imshow(cv2.cvtColor(ISIC_0024309, cv2.COLOR_BGR2RGB))
ax[0].set_title("ISIC_0024309")
plt.show()
```




In [5]:
```
# Check for similarities
sift = cv2.SIFT_create()

keypoints_1, descriptors_1 = sift.detectAndCompute(ISIC_0024306, None)
keypoints_2, descriptors_2 = sift.detectAndCompute(ISIC_0024309, None)

# Feature matching
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

matches = bf.match(descriptors_1, descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)

img = cv2.drawMatches(ISIC_0024306, keypoints_1, ISIC_0024309, keypoints_2, matches[:50], ISIC_0024309, flags=2)
plt.figure(figsize=(12, 10))
plt.imshow(img, cmap = "gray")
plt.show()
```



**Border**

**Local Binary Patterns (LBP)**
LBPs classify and identify textures and patterns in images. Unlike Haralick texture features that compute a global representation of texture based on the gray level co-occurrence matrix, LBPs compute a local representation of texture, which is constructed by comparing each pixel to its surrounding neighborhood.

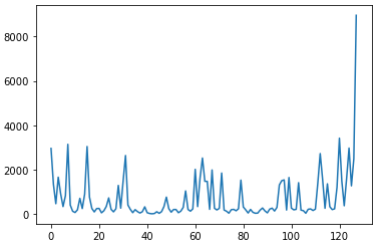
**Hu Moments**
Hu moments are used to describe, characterize, and quantify the shape of an object in an image. They are extracted from the silhouette or outline of an object in an image. By describing the outline of an object, a shape feature vector can be extracted. These shape feature vectors can be compared using a similarity metric or distance function to determine how "similar" the shapes are.

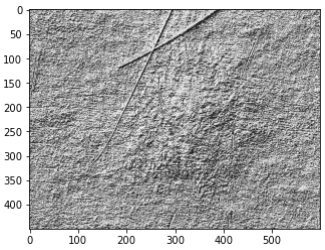
**Haralick Texture**
Haralick texture features are computed from gray-level co-occurrence matrix (GLCM), a method of quantifying the spatial relation of neighboring pixels in an image. Haralick texture features are widely used due to their simplicity and intuitive interpretations and have successfully been applied in the analysis of skin texture.

***Local Binary Patterns Example***

In [6]:
```
def local_binary_pattern(src, mask_image=None, bins=256, show_hist=False, flatten=True, show_img_lbp=True):

    def get_pixel(img, center, x, y):
        new_value = 0
        try:
            if img[x][y] >= center:
                new_value = 1
        except:
            pass
        return new_value

    def lbp_calculated_pixel(img, x, y):
        center = img[x][y]
        val_ar = []
        val_ar.append(get_pixel(img, center, x-1, y+1))     # top_right
        val_ar.append(get_pixel(img, center, x, y+1))       # right
        val_ar.append(get_pixel(img, center, x+1, y+1))     # bottom_right
        val_ar.append(get_pixel(img, center, x+1, y))       # bottom
        val_ar.append(get_pixel(img, center, x+1, y-1))     # bottom_left
        val_ar.append(get_pixel(img, center, x, y-1))       # left
        val_ar.append(get_pixel(img, center, x-1, y-1))     # top_left
        val_ar.append(get_pixel(img, center, x-1, y))       # top

        power_val = [1, 2, 4, 8, 16, 32, 64, 128]
        val = 0
        for i in range(len(val_ar)):
            val += val_ar[i] * power_val[i]
        return val

    src_gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

    src_lbp = src_gray.copy()
    for i in range(0, src_gray.shape[0]):
        for j in range(0, src_gray.shape[1]):
            src_lbp[i, j] = lbp_calculated_pixel(src_gray, i, j)

    if mask_image is None:
        hist_lbp = cv2.calcHist([src_lbp], [0], None, [bins], [0, 256])

    else:
        hist_lbp = cv2.calcHist([src_lbp], [0], mask_image[:, :, 0], [bins], [0, 256])

    if show_hist:
        plt.plot(hist_lbp)
        plt.show()

    if flatten:
        lbp = hist_lbp.flatten()
    else:
        lbp = hist_lbp

    if show_img_lbp:
        plt.imshow(src_lbp, cmap="gray")
        plt.show()

    return lbp
```

In [7]:
```
lbp = local_binary_pattern(src=ISIC_0024306, mask_image=ISIC_0024306_mask, bins=128, show_hist=True, flatten=True)
```




***Hu Moments Example***

In [8]:
```
def get_HuMoments(image):
    """Calculate Hu Moments for an image/mask.

    Args:
        src: Lesion Image or Mask
    Returns:
        hu_moments: Flat numpy array of 7 Hu Moments Elements

    References:
        https://pyimagesearch.com/2014/10/27/opencv-shape-descriptor-hu-moments-example/
    """
    img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return cv2.HuMoments(cv2.moments(img_gray)).flatten()
```

In [9]:
```
img_hum = get_HuMoments(image=ISIC_0024306)
print(img_hum)
```

```
[ 1.28502069e-03  1.36114237e-07  6.45578049e-13  3.25501869e-12
  3.88384830e-24  1.18666725e-15 -2.67955800e-24]
```

In [10]:
```
mask_hum = get_HuMoments(image=ISIC_0024306_mask)
print(mask_hum)
```

```
[ 6.55793561e-04  1.95517266e-08  7.10539003e-12  8.59719452e-14
 -8.24831564e-27  9.59427210e-19  6.66856036e-26]
```

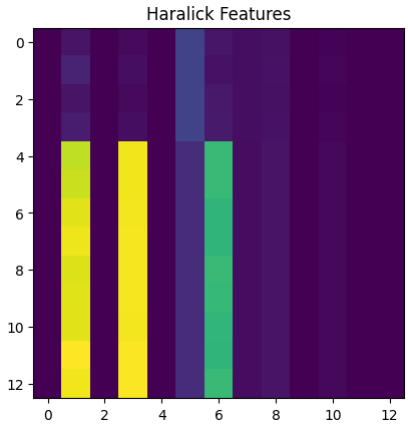
***Haralick Texture Example***

In [2]:
```
def get_haralick_texture_features(img, flatten=True):
    """Returns 169 (if flatten = True) else 13 x 13 Haralick exture analysis features.

    Args:
        img: BGR Image C x H x W (3 channels)
        flatten: Boolean True/False, default True, if False then a 13 x 13 array is returned

    References:
        https://learning.oreilly.com/library/view/python-data-analysis/9781785282287/ch11s11.html#ch11lvl2sec325
        https://murphylab.web.cmu.edu/publications/boland/boland_node26.html
    """
    if flatten:
        return mh.features.haralick(img.astype(np.uint8)).ravel()
    else:
        return mh.features.haralick(img.astype(np.uint8))
```

In [6]:
```
haralicks = get_haralick_texture_features(img=ISIC_0024306);
haralicks[:5]
```

Out[6]:
```
array([9.06137440e-04, 1.25936785e+01, 9.97954656e-01, 3.07862178e+03,
       3.23636224e-01])
```

In [7]:
```
plt.figure()
plt.imshow(haralicks.reshape(13,13).astype(np.uint8))
plt.title("Haralick Features")
plt.show()
```



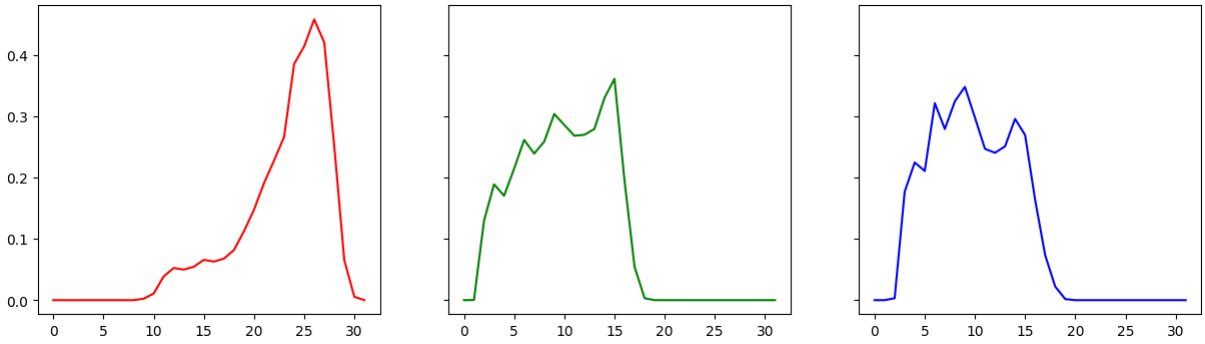

**Color:**

***Color Histograms***

Color histograms represent the frequency distribution of pixel intensities of each of the color channels in an image. By using segmentation masks, we are able to focus on only the lesion area and capture the pixel-level intensity distributions of the red, green, and blue channels.

In [8]:
```
def get_color_histogram(image, mask_image=None, bins=256, normalize=True, show_hist=False, flatten=True):
    """Calculates color histogram for an image and returns a vector of histogram values.

    Args:
        image: Original Image, expected in BGR color channels
        mask_image: [Optional] A mask image of the same dimension as `image` (we use channel 0)
        bins: No of bins for Histogram Calculation, default is 256
        normalize: Histogram should be normalized, default False
        show_hist: True/False, show the histogram, default False
        flatten: True/False, flatten the 3 channel histogram to a vector, default False
    return:
        hist_vec: Histogram Vector of 3 (RGB color channels) * bins

    References:
        https://pyimagesearch.com/2021/04/28/opencv-image-histograms-cv2-calchist/
        https://github.com/pinecone-io/examples/blob/master/learn/image-retrieval/color-histograms/00-build-histograms.ipynb
        https://github.com/pinecone-io/examples/blob/master/learn/image-retrieval/color-histograms/01-search-histogram.ipynb

    """

    if mask_image is None:
        red_hist = cv2.calcHist([image], [2], None, [bins], [0, 256])
        green_hist = cv2.calcHist([image], [1], None, [bins], [0, 256])
        blue_hist = cv2.calcHist([image], [0], None, [bins], [0, 256])
    else:
        red_hist = cv2.calcHist([image], [2], mask_image[:, :, 0], [bins], [0, 256])
        green_hist = cv2.calcHist([image], [1], mask_image[:, :, 0], [bins], [0, 256])
        blue_hist = cv2.calcHist([image], [0], mask_image[:, :, 0], [bins], [0, 256])

    if normalize:
        red_hist = cv2.normalize(red_hist, red_hist)
        green_hist = cv2.normalize(green_hist, green_hist)
        blue_hist = cv2.normalize(blue_hist, blue_hist)

    if show_hist:
        fig, axs = plt.subplots(1, 3, figsize=(15, 4), sharey=True)
        axs[0].plot(red_hist, color='r')
        axs[1].plot(green_hist, color='g')
        axs[2].plot(blue_hist, color='b')
        plt.show()

    if flatten:
        hist_values = np.concatenate([red_hist, green_hist, blue_hist], axis=0).reshape(-1)
    else:
        hist_values = np.dstack((red_hist, green_hist, blue_hist))

    return hist_values
```

In [9]:
```
masked_32_norm_show_flat = get_color_histogram(image=ISIC_0024306, mask_image=ISIC_0024306_mask, bins=32, normalize=True, show_hist=True, flatten=True)
```



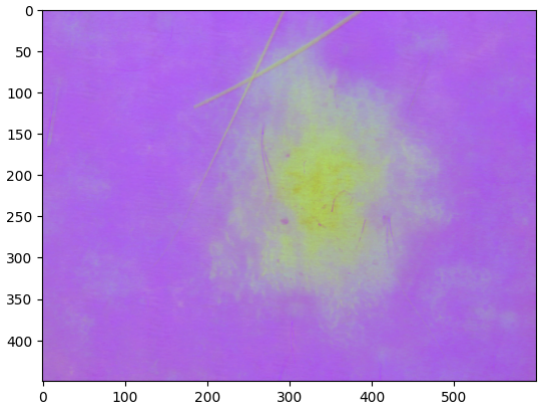

***RGB to HSV***

The HSV (hue, saturation, value) colorspace is commonly used in image processing, as it is often more natural to think about an image in terms of hue and saturation rather than in terms of additive or subtractive color components. The hue channel takes values in $[0, 179]$, whereas the saturation and value channels take values in $[0, 255]$.

In [10]:
```
def rgb_to_hsv(src):
    return (255*rgb2hsv(src)).astype('uint8')
```

In [11]:
```
hsv = rgb_to_hsv(ISIC_0024306)
plt.imshow(hsv)
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x12b5a4c70>
```



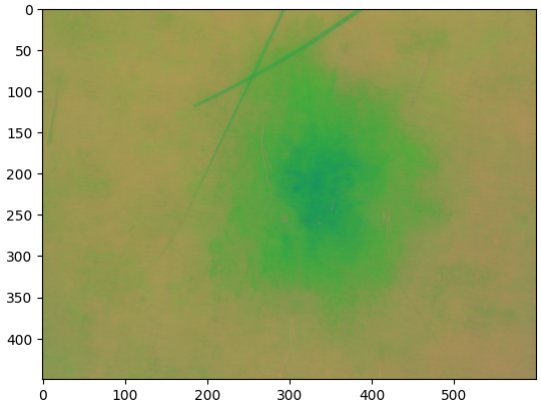

***RGB to L*A*B***

L*A*B stands for lightness, red/green value, and blue/yellow value. The L*A*B colorspace covers the entire range of human color perception, and is based on the opponent color model of human vision where red/green and blue/yellow form opponent pairs. The lightness value 0 for black pixels and 100 for white pixels. The A* axis represents the red/green opponent pair, with negative values tending toward green and positive values tending toward red. The B* axis represents the blue/yellow opponent pair, with negative values tending toward blue and positive values tending toward yellow.

In [12]:
```
def rgb_to_lab(src):
    return cv2.cvtColor(src, cv2.COLOR_RGB2LAB)
```

In [13]:
```
lab = rgb_to_lab(ISIC_0024306)
plt.imshow(lab)
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x12b626790>
```




## Data Selection

Prior to beginning model development one of our team's primary tasks was separating our 10015 labeled images into a training and test set.

### Issues with Image Variance by Class

While simple random sampling is a common approach, during EDA we observed significant amounts of variation in images within each disease class across factors such as:

- Light
- Hair
- Coloration
- Lesion Position
- Relative Lesion Size

This led to the hypothesis taking a simple random sample for training would be suboptimal in terms of capturing the natural variation of the images as they exist in feature space. Our solution to this was **within-lesion class clustering**, a form of stratified random sampling where we use unsupervised clustering to automatically learn sample frames.

### Within-Lesion Class Clustering for Sample Frame Generation

Within-lesion class clustering allowed us to utilize unsupervised classical machine learning methods to learn natural sampling frames represented by clusters of image data. As a first step, we applied a sequence of image preprocessing steps to standardize images before converting them into 1D feature vector representations. This preprocessing sequence, in order, is listed below:

- Iterative blurring and downsampling with mean pooling, using a $(2, 2)$ window and a $(2, 2)$ stride size
- Hair removal
- RGB to YCrCb to decorrelate color channels
- Histogram equalization of each channel (Y, Cr, Cb)
- Normalized YCrCb to RGB for interpretability
- Array flattening to 1D vector representations

Following this preprocessing pipeline, our team applied a $k$-means clustering algorithm within each class to identify natural groupings of images that shared similar characteristics. The identification of these clusters within each disease class served as a logical sampling frame from which we could create training and test sets:

- Training set: 8056 images
- Test set: 1959 images

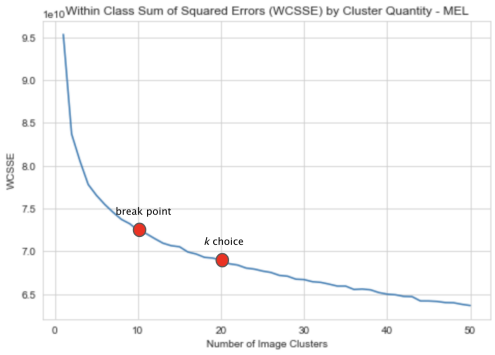
An important choice that accompanies this approach is what what choice of $k$ to use to generate optimal image clusters within each class. We utilized a metric-based heuristic make this decision, by applying $k$-means across a variety of $k$ values and measuring the intra-cluster variance $\sigma_c^2$:

$$\sigma_c^2 = \sum_{i=1}^{n} d(x_i, c_i)^2,$$

where $d$ is a distance metric of choice and $c_i$ is the cluster centroid that $x_i$ is assigned to. Taking $d$ as the Euclidean distance, the expression for the variance becomes

$$\sigma_c^2 = \sum_{i=1}^{n} (x_i - c_i)(x_i - c_i)^\top.$$

We chose an optimal $k$ by plotting the within-cluster variance against $k$ and identifying a natural breakpoint (we actually chose a value slightly beyond this breakpoint in order to be conservative).

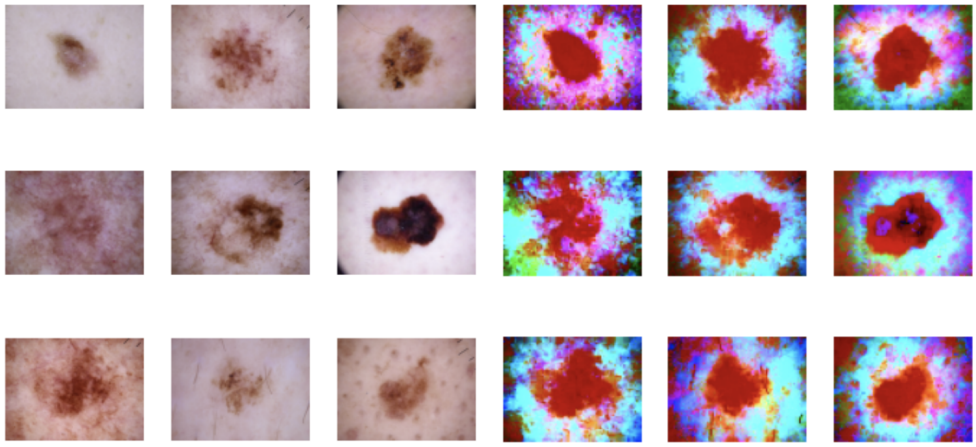


After applying this method to all seven classes, we obtained the following $k$ values:

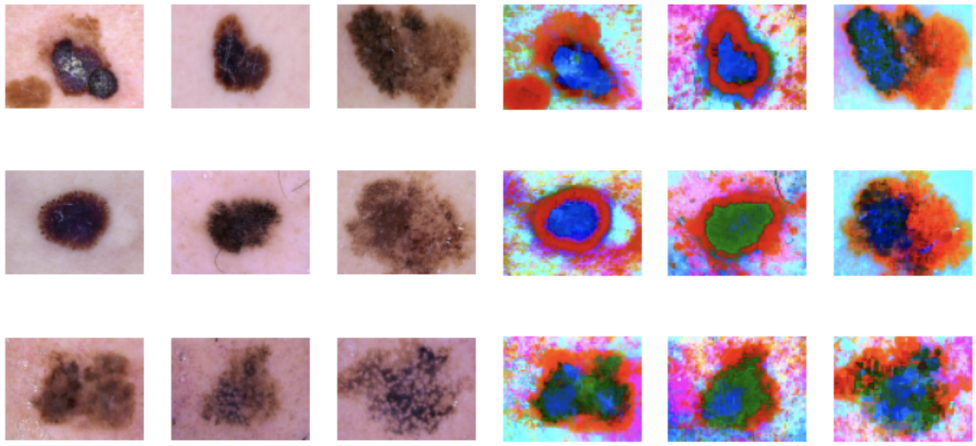
- AKIEC: $20$
- BCC: $20$
- BKL: $20$
- DF: $10$
- MEL: $20$
- NV: $20$
- VASC: $10$

Image clusters developed within the melanoma class are shown below. The left side of each panel shows a sample of raw images within a specific cluster in a $3 \times 3$ grid, while the right side shows the clustered image representations in a $3 \times 3$. It is interesting to note that the developed image clusters still show some variation across factors such as light and hair; the similarity of images in each cluster is largely based on lesion size, position, and depth of color relative to the surrounding area.

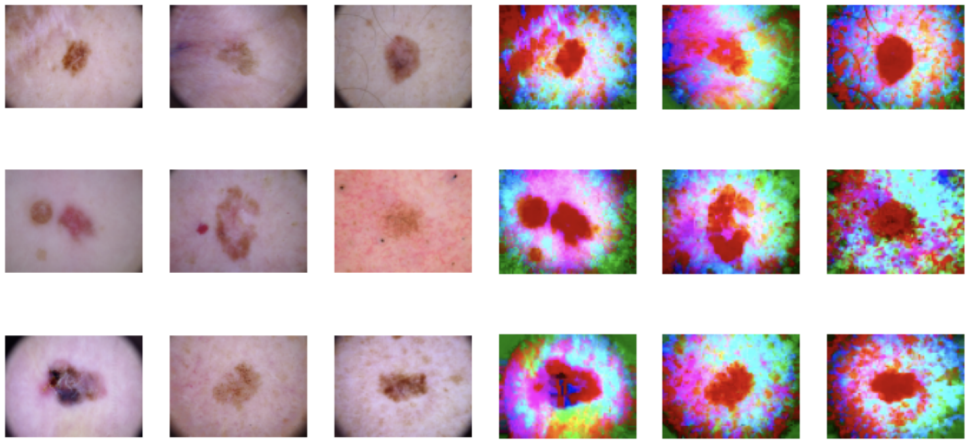
MEL Cluster 2



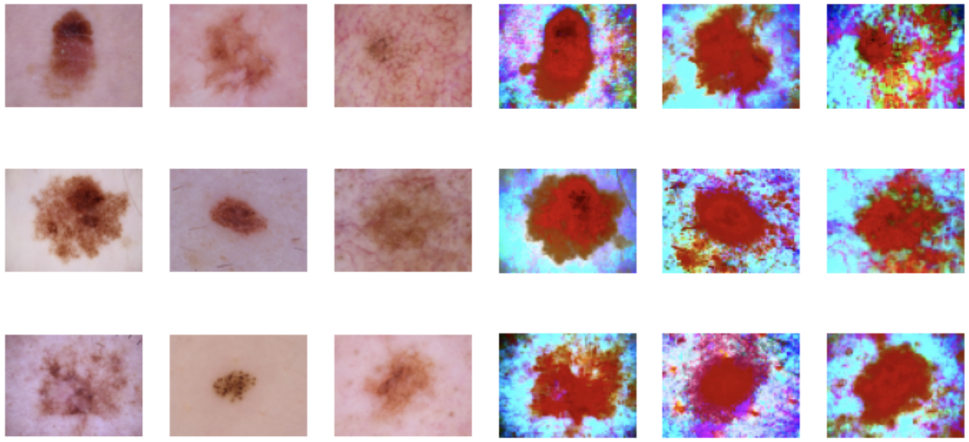
MEL Cluster 3



MEL Cluster 9

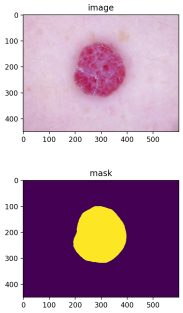


MEL Cluster 10



## Modeling and Results (Segmentation)

The HAM10000 dataset provides segmentation masks for all 10015 images in the dataset, each of which was hand-labeled by a medical expert. An example of a lesion and its corresponding mask is shown below:




In a real-world setting, accurate segmentation masks may not be readily available. Therefore, there is value in building a model that can automatically predict the segmentation for a given image of a skin lesion. A **segmentation model** is a model that predicts a class for every pixel in an image. In this case, there are only two classes: lesion and background. In the following sections, we go over four models that we developed for segmenting skin lesion images:

1. Naive thresholding
2. Improved thresholding
3. $k$-means clustering
4. Deep learning

### Naive Thresholding

One naive approach to segment an image of a skin lesion is to threshold the image. In general, lesions are easy to separate from background skin using to differences in intensity. Therefore, it is natural to create a segmentation model which simply thresholds the image at some value. To select these values, we use Otsu's method, which seeks the value that minimizes the intra-class variance $\sigma_w^2(t)$, defined as

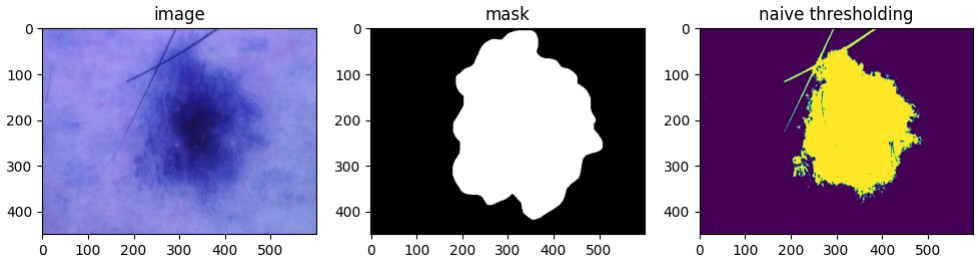
$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t),$$

where $\omega_0$ and $\omega_1$ are the probabilities of the two classes (background and lesion) separated by a threshold $t$, and $\sigma_0^2$ and $\sigma_1^2$ are the variances of the two classes.

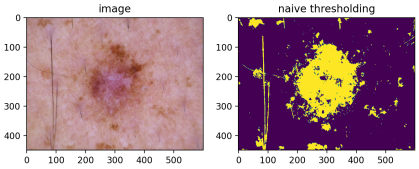
An illustration of a segmentation mask produced by this model is shown below:

In [1]:
```python
def naive_threshold_segment(img):
    """Predicts a segmentation mask for a skin lesion image using thresholding
    and Otsu's method.

    Args:
        img: The image to predict a segmentation mask for.

    Returns:
        segmented: The segmented image.
    """
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    h, w = img.shape
    _, segmented = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    segmented = 1.0 - (segmented > 0.5).astype(np.uint8)
    return segmented
```

In [6]:
```python
thresh = naive_threshold_segment(ISIC_0024306)

fig, axs = plt.subplots(1, 3, figsize=(12, 4))
axs[0].imshow(ISIC_0024306)
axs[0].set_title(f"image")
axs[1].imshow(ISIC_0024306_mask)
axs[1].set_title(f"mask")
axs[2].imshow(thresh)
axs[2].set_title(f"naive thresholding")
plt.show()
```




Although this approach is a sensible starting point, it has visible flaws. Most notably, this approach is sensitive to the presence of hair or subtle variations in skin pigmentation. In general, the predicted masks are noisier than the hand-labeled masks, since every morphological element on an image that is similar in intensity to the lesion is predicted as belonging to that lesion. The following image shows an example of a lesion where our naive thresholding model performs poorly.




### Improved Thresholding

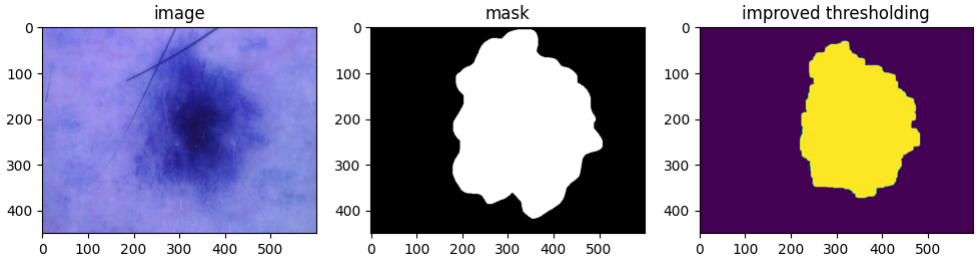
We improved our naive thresholding model by adding more advanced processing steps to remove thin morphological elements such as hair from the image. After several rounds of experimentation, we settled on the following logic:

1. Convert the image from RGB to grayscale
2. Apply a $3 \times 3$ Gaussian blur to remove small point-like artifacts
3. Erode the image using a $9 \times 9$ kernel to remove noisy artifacts
4. Apply Otsu's method to produce a thresholded image
5. Use a Canny edge detector to find the edges in the thresholded image
6. Fill in the contour from the previous step

In the next cell we show the mask prediction for the same lesion, but with the additional logic described above. The resulting mask is smoother and lacks the undesirable noise from the previous image.

In [14]:
```python
def improved_threshold_segment(img):
    """Predicts a segmentation mask for a skin lesion image using improved
    image processing methods.

    Args:
        img: The image to predict a segmentation mask for.

    Returns:
        segmented: The segmented image.
    """
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.GaussianBlur(img, (3, 3), 0)
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 9))
    img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
    h, w = img.shape
    _, t = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    edges = cv2.dilate(cv2.Canny(t, 0, 255), None)
    cnt = sorted(cv2.findContours(edges, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[-2], key=cv2.contourArea)[-1]
    segmented = np.zeros((h, w), np.uint8)
    for y in range(h):
        idx = (cnt[:, 0, 1] >= y - 5) & (cnt[:, 0, 1] <= y + 5)
        if idx.sum() > 1:
            x_range = cnt[idx, 0, 0]
            x_min, x_max = x_range.min(), x_range.max()
            segmented[y, x_min:x_max] = 1
    segmented = cv2.dilate(segmented, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9, 9)))
    segmented = (segmented > 0.5).astype(np.uint8)
    return segmented
```

In [15]:
```
thresh = improved_threshold_segment(ISIC_0024306)

fig, axs = plt.subplots(1, 3, figsize=(12, 4))
axs[0].imshow(ISIC_0024306)
axs[0].set_title(f"image")
axs[1].imshow(ISIC_0024306_mask)
axs[1].set_title(f"mask")
axs[2].imshow(thresh)
axs[2].set_title(f"improved thresholding")
plt.show()
```




### K-Means Clustering

Another method for segmenting images is $k$-means clustering. The objective of $k$-means clustering is to partition an image into $k$ clusters $C = \{C_1, \ldots, C_k\}$ such that the within-cluster variance is minimized:

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2,$$

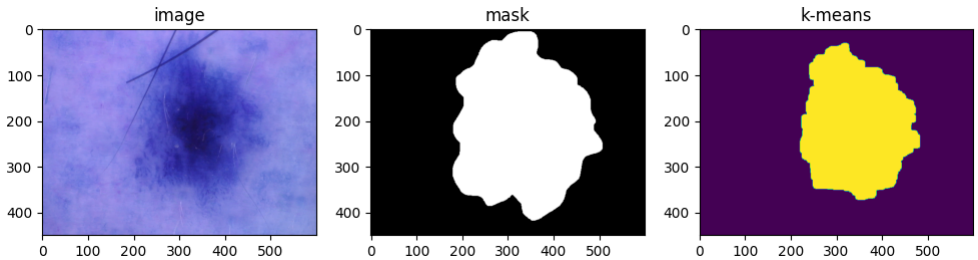
where $\mu_i$ is the mean of cluster $C_i$.

Unfortunately, $k$-means clustering is also susceptible to noisy artifacts. In order to alleviate these undesired effects, we apply similar processing methods as in our improved thresholding model. The full logic is as follows:

1. Convert the image from RGB to grayscale
2. Apply a $3 \times 3$ Gaussian blur
3. Erode the image using a $9 \times 9$ kernel
4. Apply $k$-means clustering with $k = 2$
5. Use a Canny edge detector to find the edges in the clustered image
6. Fill in the contour from the previous step

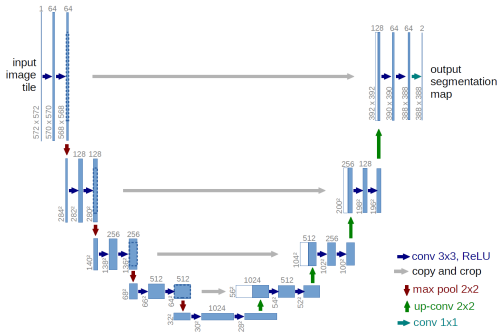
In [16]:
```
def kmeans_segment(img):
    """Predicts a segmentation mask for a skin lesion image using k-means
    clustering.

    Args:
        img: The image to predict a segmentation mask for.

    Returns:
        segmented: The segmented image.
    """
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.GaussianBlur(img, (3, 3), 0)
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 9))
    img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
    h, w = img.shape
    img = img.reshape(-1, 1).astype(np.float32)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    attempts = 10
    ret, label, center = cv2.kmeans(img, 2, None, criteria, attempts, cv2.KMEANS_PP_CENTERS)
    center = np.uint8(center)
    res = center[label.flatten()]
    cluster = res.reshape((h, w))
    edges = cv2.dilate(cv2.Canny(cluster, 0, 1), None)
    cnt = sorted(cv2.findContours(edges, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[-2], key=cv2.contourArea)[-1]
    segmented = np.zeros((h, w), np.uint8)
    for y in range(h):
        idx = (cnt[:, 0, 1] >= y - 5) & (cnt[:, 0, 1] <= y + 5)
        if idx.sum() > 1:
            x_range = cnt[idx, 0, 0]
            x_min, x_max = x_range.min(), x_range.max()
            segmented[y, x_min:x_max] = 1
    segmented = cv2.dilate(segmented, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9, 9)))
    segmented = (segmented > 0.5).astype(np.uint8)
    return segmented
```

In [17]:
```
thresh = kmeans_segment(ISIC_0024306)

fig, axs = plt.subplots(1, 3, figsize=(12, 4))
axs[0].imshow(ISIC_0024306)
axs[0].set_title(f"image")
axs[1].imshow(ISIC_0024306_mask)
axs[1].set_title(f"mask")
axs[2].imshow(thresh)
axs[2].set_title(f"k-means")
plt.show()
```



## Deep Learning

We also tried to leverage deep learning models to segment skin lesion images. One well-known model for image segmentation in medical applications is U-Net, introduced in 2015 (https://arxiv.org/abs/1505.04597). This model consists of two parts: an encoder which captures context and an encoder which enables precise localization.



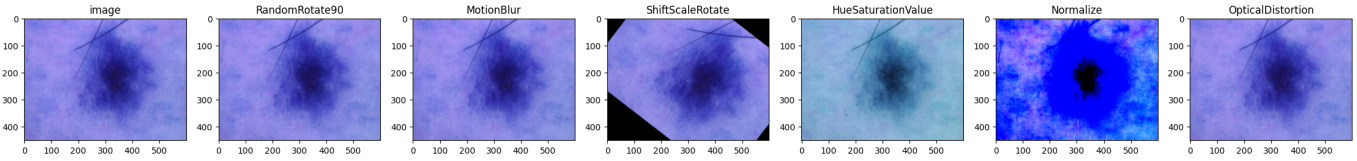

### Image Augmentations

Before we trained the U-Net model, we padded the images to a minimum of $512 \times 512$, then cropped them to exactly those dimensions (this ensured that the aspect ratio was maintained). We also applied various image augmentations during training. Although the HAM10000 dataset is well-curated, we cannot expect most lesions images in the real world to be clean. Therefore, we believe that this augmentation step is crucial for improving the model's ability to generalize to unseen data. Some examples of image augmentations are shown below.

It is important to note here that the masks (which served as the labels to train the U-Net model with) were augmented in the same way as the images. For instance, if the image was rotated, then the mask was also rotated by the same amount. This ensured that the images and labels were never misaligned. When normalizing the color channels, we followed the convention of using means of $[0.485, 0.456, 0.406]$ and standard deviations of $[0.229, 0.224, 0.225]$.

In [7]:
```
augments_list = [
    A.RandomRotate90(p=1.0),
    A.MotionBlur(p=1.0),
    A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.2, rotate_limit=45, interpolation=cv2.INTER_LINEAR, border_mode=cv2.BORDER_CONSTANT, p=1.0),
    A.HueSaturationValue(p=1.0),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225), p=1.0),
    A.OpticalDistortion(p=1.0)
]
n = len(augments_list) + 1

fig, axs = plt.subplots(nrows=1, ncols=n, figsize=(n * 4, 4))

axs[0].imshow(ISIC_0024306)
axs[0].set_title("image")

for i, augment in enumerate(augments_list):
    t = augment(image=ISIC_0024306, mask=ISIC_0024306_mask)
    name = augment.__class__.__name__
    axs[i + 1].imshow(t["image"])
    axs[i + 1].set_title(name)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```




### Training Set-Up

We used a 5-fold cross-validation scheme to train the U-Net model. For a given fold, we would train on $80\%$ of the data and evaluate on the remaining $20\%$. Each model was trained for $3$ epochs with a batch size of $8$. Instead of using a constant learning rate, we used a learning rate schedule that initially starts at $3.0 \times 10^{-5}$, peaks at a maximum of $2.0 \times 10^{-4}$ near the halfway point, and then slowly decreases to zero. We found that using a scheduler in this way improved the stability of the model training. Our optimizer was the Rectified Adam optimizer (Liu et al. 2019 (https://arxiv.org/abs/1908.03265)), and our loss function was an average of the DICE loss and the pixel-wise binary cross-entropy loss. Specifically, given a $n \times n$ binary segmentation label $y$ and a predicted $n \times n$ mask $\hat{y}$ whose pixel values are probabilities that they belong to the lesion in the image, the loss is given by

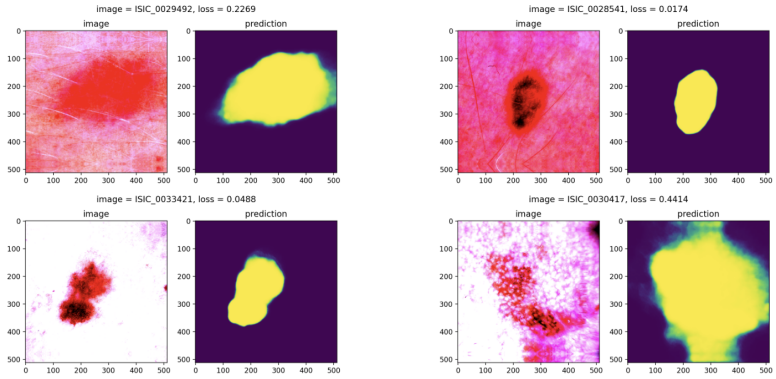
$$L(y, \hat{y}) = \frac{1}{2}\left[1 - \frac{2|y \cap \hat{y}|}{|y| + |\hat{y}|}\right] - \frac{1}{2n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left[y_{ij}\log \hat{y}_{ij} + (1 - y_{ij})\log(1 - \hat{y}_{ij})\right].$$

After each epoch, we evaluated the U-Net model on the out-of-fold data and computed the average loss. At any given point, we only saved the model if the average validation loss had improved over the previous epoch. For more details, please refer to the notebook `w281_final_project/romain/segmentation_gp.ipynb`.

### Sample Predictions

Below, we show 4 sample out-of-fold predictions by the U-Net model. Notably, we see that the U-Net model is able to reliably differentiate between hairs, background skin, and lesions, despite the lack of extensive preprocessing on our part. We also notice that for skin lesions with soft edges, the predicted pixel probabilities slowly decrease as we move away from the lesion center. The model accurately conveys uncertainty regarding the boundary between lesions and skin, despite the fact that all of the label masks that the model trained on had sharp edges.

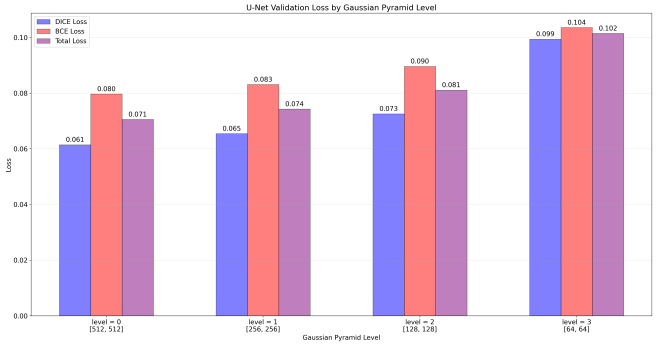
We also note that the pink-red hue of the images is a result of the channel normalization that occurs before an image is fed through the model.




**U-Net Results**

Our best model had $5$ encoding blocks and decoding blocks, and achieved average out-of-fold DICE, BCE, and total losses of $0.0614$, $0.0797$, and $0.0706$, respectively. Because the images we trained on were quite large, we were curious to see if we could obtain comparable results by training on downsampled images rather than the full-resolution images. If so, then implementing such a model in production would be both cheaper and faster.

In order to test the effect of downsampling on the performance of the U-Net model, we created a Gaussian pyramid of the images and masks with three levels. For each level, we trained a separate U-Net model with the same parameters as above and evaluated its out-of-fold performance. The following plot shows the average validation loss of each model by level (the zeroth level is the full resolution image).



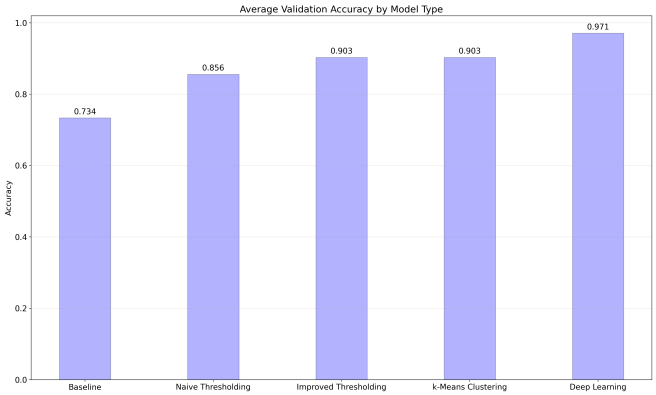

As expected, the performance is worse on downsampled images. However, the loss in performance is minimal for the first two levels of the pyramid. In production, it may be preferable to use a downsampled model instead of the full-resolution model in the interest of making quicker predictions.

### Model Comparisons

Finally, we provide a systematic comparison of all of the models we have discussed thus far: the naive thresholding model, the improved thresholding model, the $k$-means clustering model, and the U-Net model. Because the first three models do not output probabilities, we cannot use the DICE score or the BCE loss as a comparison metric. Instead, we show the average pixel accuracy of the four approaches. Given a $n \times n$ binary segmentation label $y$ and a predicted $n \times n$ binary mask $\hat{y}$, the average pixel accuracy $A$ is defined as

$$A(y, \hat{y}) = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left[ 1 - |y_{ij} - \hat{y}_{ij}| \right].$$

The plot below shows the average validation pixel accuracy of all four models discussed thus far. For reference, we have also included the baseline accuracy of a model that predicts "skin" for every pixel in an image. Immediately, we see that all models are better than the baseline by a significant margin. We also notice an increase in accuracy from the naive thresholding model to the improved thresholding and $k$-means models, which indicates that our additional processing steps were at least partially successful. Interestingly, the validation performance of the improved thresholding and $k$-means models are identical; this is likely due to the fact that Otsu's method and the $k$-means algorithm both seek to minimize the intra-class variance. Perhaps unsurprisingly, our deep learning model greatly outperforms the other models with an average pixel accuracy of $97.1\%$. Notably, this model is able to learn features that help it differentiate between skin and lesions with only minimal preprocessing and cleaning.




## Skin Lesion Classification

Image classification models take images as inputs and predict discrete classes as outputs. In the context of skin lesion classification, this translates to ingesting dermascopic images and returning probabilities for each of the seven lesion classes.

## Error Metrics

Our primary metric for evaluating our models' performance on held-out data is the macro $F_1$ score, or the average $F_1$ across all seven classes. For a given class, the $F_1$ score is the harmonic mean between precision and recall:

$$F_1 = \frac{2 \times (\text{precision} \times \text{recall})}{\text{precision} + \text{recall}}.$$

**Precision** is the ratio of true predictions to the number of predictions made for a class, while **recall** is the ratio of true predictions to the number of true instances of a class.

In addition to the macro $F_1$ score, we also considered the accuracy, top-2 macro $F_1$, and top-2 accuracy scores. Top-2 metrics treat predictions as true positives if one of the two classes with the highest predicted probabilities are the true class; this flavor of metrics is common practice in medical machine learning in order to give medical practitioners a probabilistic breakdown when making diagnostic decisions.

## Classical Approaches

We began our experimentation with lesion classification by evaluating the efficacy of classical supervised machine learning models:

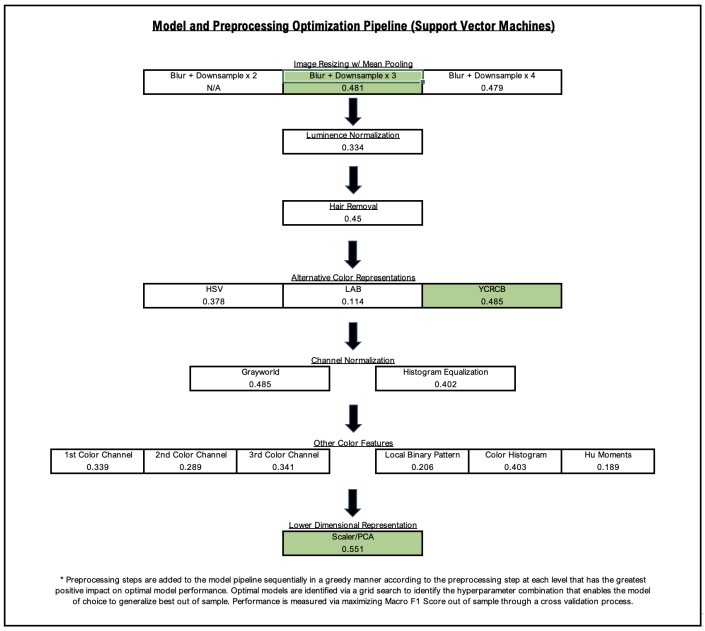
- **Distance-based**
  - $k$-nearest neighbors ($k$-NN)
- **Partition-based**
  - Random forests
  - Gradient-boosted classification trees
- **Generative**
  - Gaussian Naive Bayes
- **Parametric (linear and non-linear)**
  - Logistic regression (linear)
  - Support-vector machines (radial basis kernel, non-linear)
- **Custom ensembles**
  - Stacked models

Our reasoning for experimenting with such a wide variety of models was that they all learn from labeled data in fundamentally different ways, with different assumptions and formulations. We were interested to see which models and assumptions would be well-suited to this type of task.

#### Optimal Model and Preprocessing Identification

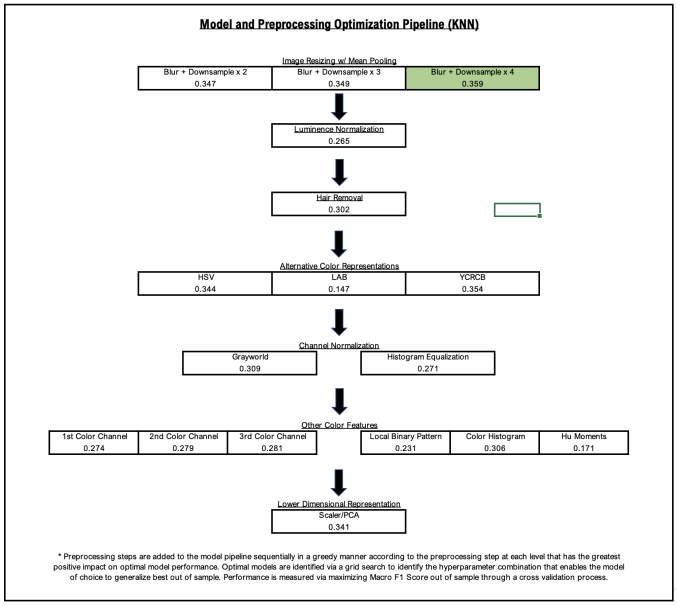
We used a grid search algorithm with five-fold cross-validation to find the hyperparameter combination that maximized the out-of-sample macro $F_1$ score. While this approach is simple and effective, it assumes that we know the appropriate preprocessing steps to apply and does not inform us how to preprocess our images to optimize model performance for a specific type of model. This is an example of the "no free lunch theorem", which states that there is no one-size-fits-all approach to machine learning. For example, the optimal image size for a $k$-NN model may be small due to the curse of dimensionality and assumptions about closeness, while the optimal image size for a logistic regression model may be large due to the highly non-linear nature of the problem, which requires many more dimensions to effectively separate classes in feature space.

To address this issue, our team designed an intuitive AutoML greedy approach to identify preprocessing sequences that optimize out-of-sample model performance. This approach builds on the grid search algorithm by adding a preprocessing identification pipeline that evaluates the impact of preprocessing steps sequentially and only adds steps to the pipeline if their inclusion improves model performance given previously added steps.

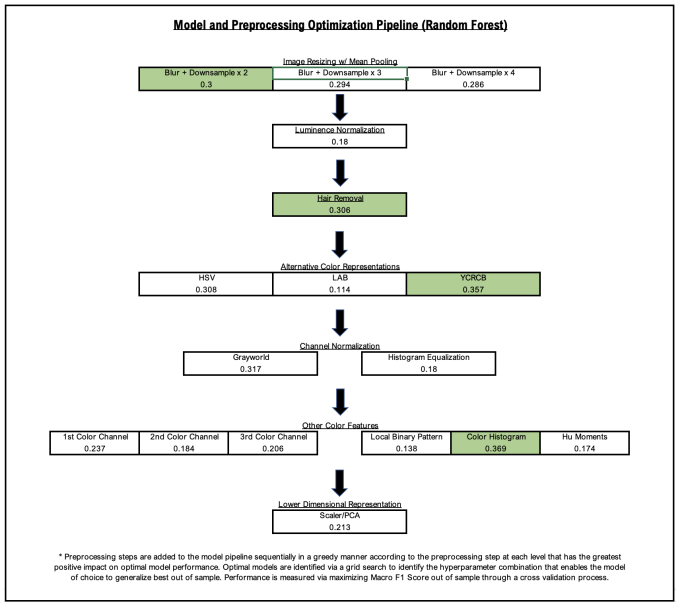


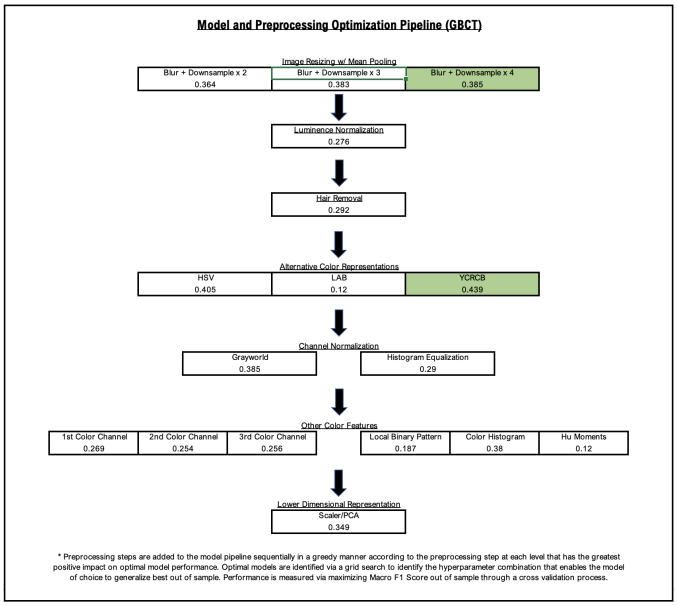

The following steps demonstrate an example of how this AutoML framework operates in practice for a support-vector machine model.

- We first evalaute model performance when trained on images of various sizes that were generated with blurring and downsampling. Applying this procedure three times had the best impact on performance, so this step is added to the preprocessing pipeline.
- We then evaluate the impact of various standardization techniques like hair removal and luminance normalization on the images. Since the neither technique improves the model performance, we do not add them to the pipeline.
- Next, we investigate alternative color channel representations and notice that translating our smaller images to YCrCb is able to improve model performance slightly. We add this step to the pipeline.
- We then inspect other normalization schemes such as gray-world and histogram equalization, but again we do not see any improvements.
- At this stage, our preprocessing pipeline is still generating three-channel images that our model uses as inputs. The impact of smaller image representations such as single-channel images and extracted feature vectors (local binary patterns, color histograms, and hu moments) are evaluated. These representations do not improve model performance, so they are not added to the pipeline.
- Finally, we apply scaling and principal-component analysis (PCA) to our images (currently downsampled and translated to YCrCb), and notice a significant improvement in model performance. This last step is added to the preprocessing pipeline.








A large benefit of this approach is that it enables us to identify optimal preprocessing pipelines in parallel with optimal models.

**Model Results**

The classical model with the best performance was an optimized support-vector machine, which achieved a macro $F_1$ of $60\%$ and an accuracy of over $78\%$ on the test set. Our stacked ensemble, which took in the probabilistic predictions of five other models as features, performed best in top-2 metrics with a top-2 $F_1$ score of over $73\%$ and a top-2 accuracy near $90\%$. While the stacked ensemble was a slight improvement over the SVM in terms of top-2 metrics, the latter model remained competitive. The choice of one model over the other depends on (1) which metric family (top-1 or top-2) is more significant practically, and (2) whether model performance is more important than model simplicity.

The table below compares the performance of the support-vector machine and stacked ensemble by class.

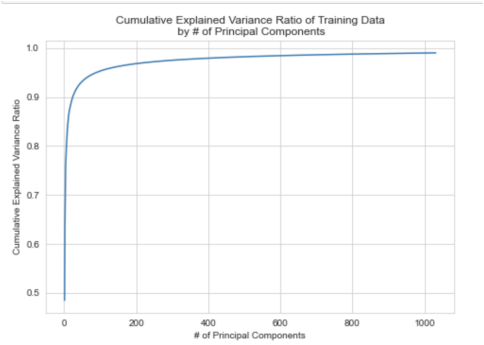
| | | Top 1 | | Top 2 | |
|---|---|---|---|---|---|
| Model | Preprocessing Summary | Macro F1 | Accuracy | Macro F1 | Accuracy |
| Baseline (Predict Majority) | | 11.5 | 66.9 | 26.6 | 78.1 |
| K Nearest Neighbors | Blur + Downsample x 4 | 33.7 | 69.1 | 44 | 71.7 |
| Gaussian Naïve Bayes | Blur + Downsample x 4, YCRCB | 29.4 | 49.7 | 42.6 | 58.4 |
| Random Forest | Blur + Downsample x 2, Hair Removal, YCRCB, Color Histogram | 34.5 | 73.5 | 60.7 | 87.5 |
| Gradient Boosted Trees | Blur + Downsample x 4, YCRCB | 45.8 | 76.7 | 62.8 | 88.4 |
| Support Vector Machine (RBF) | Blur + Downsample x 3, YCRCB, Scale/PCA | 60 | 78.3 | 71.5 | 89.3 |
| Stacked Ensemble (Logistic Regression Final Layer) | Above 5 Models Develop Input Features | 54.5 | 77.3 | 73.2 | 89.8 |

| | Support Vector Machine (RBF) | | | | | | Stacked Ensemble (Logistic Regression Final Layer) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Top 1 | | | Top 2 | | | Top 1 | | | Top 2 | | |
| Class | Precision | Recall | F1 | Precision | Recall | F1 | Precision | Recall | F1 | Precision | Recall | F1 |
| AKIEC | 44 | 46 | 45 | 60 | 56 | 58 | 53 | 41 | 46 | 73 | 68 | 70 |
| BCC | 46 | 63 | 53 | 59 | 69 | 64 | 54 | 49 | 52 | 75 | 64 | 69 |
| BKL | 44 | 59 | 51 | 73 | 70 | 71 | 52 | 45 | 48 | 77 | 69 | 73 |
| DF | 82 | 45 | 58 | 100 | 45 | 62 | 45 | 25 | 32 | 58 | 35 | 44 |
| MEL | 63 | 58 | 60 | 79 | 74 | 77 | 51 | 38 | 44 | 88 | 80 | 84 |
| NV | 94 | 88 | 91 | 97 | 99 | 98 | 86 | 93 | 89 | 94 | 99 | 96 |
| VASC | 57 | 64 | 60 | 69 | 72 | 71 | 83 | 60 | 70 | 94 | 64 | 76 |

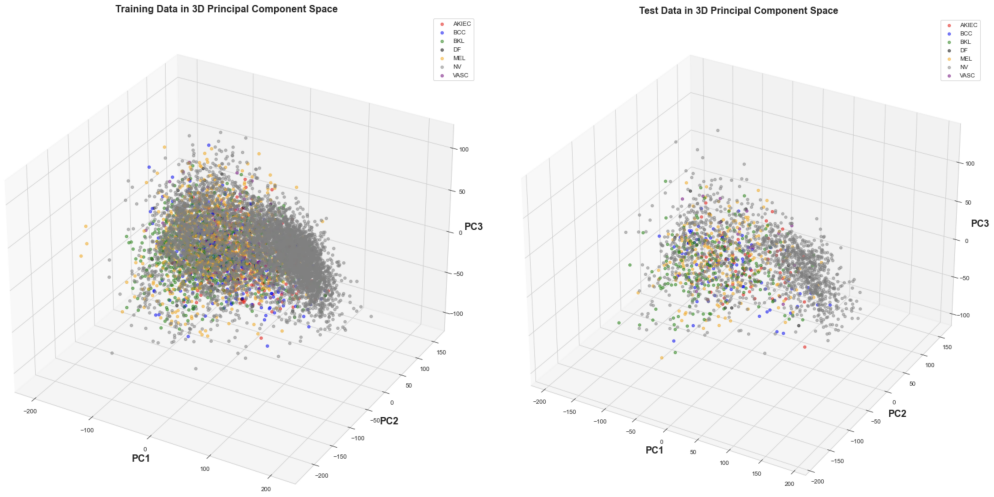
Some interesting findings from this table are:

- The performance of both models on the NV class is similar.
- The stacked ensemble was significantly worse at detecting melanomas relative to the support-vector machine when considering top-1 metrics, but better when considering top-2 metrics.
- The stacked ensemble consistently struggled with the DF class.
- The support-vector machine consistently underperformed on the VASC class relative to the stacked ensemble.
- The SVM generally achieves better top-1 recall on the AKIEC, BCC, and BKL classes, while the stacked ensemble achieves better precision on those classes. Their $F_1$ scores on these segments are similar.

Given that our optimized SVM pipeline included scaling PCA steps, this allowed us to visualize lower-dimensional representations in order to better understand the variance in the lesion images.




```
90% Variance Explained with 24 Principal Components
95% Variance Explained with 90 Principal Components
98% Variance Explained with 425 Principal Components
99% Variance Explained with 1031 Principal Components
Original Image Size: 12825 Dimensions
```




This PCA analysis highlights the fact that most of the variance in the lesion images can be captured in a low-dimensional principal component space ($90\%$ explained variance in $24$-dimensional space). It also validates our sampling strategy, since the distributions of PCA representations in the training and test sets are closely similar.

## Deep Learning

Eventually, our model experimentation for lesion classification transitioned from classical machine learning models to deep learning models. Our team was aware that an AutoML style of preprocessing and model optimization in parallel was likely not feasible for deep learning models, simply due to the amount of compute resources necessary for training such models. Instead, we elected to utilize transfer learning and pretrained models in order to take advantage of the information that these models had already learned on other tasks.

We designed two sets of deep learning experiments:

1. **Blurred/downsampled:** Models were trained and evaluated on blurred and downsampled images using mean pooling to both reduce noisy artifacts while also speeding up training and prediction time.
2. **Segmentation-informed:** Our U-Net segmentation model was used as a preprocessing tool for cropping the region around the lesion to focus training and inference on modified images that were localized to just the lesion area.

Both setups involved re-training the entire network rather than freezing the convolutional layers due to differences between our task and the pre-training tasks.

#### Blurred/Downsampled Approach

For this approach, we iteratively blurred and downsampled images with a mean pooling procedure using a $(2, 2)$ window and $(2, 2)$ stride size, reducing the image size by a factor of $16$. We held out approximately $1000$ images as a validation set to use for early stopping and prevent overfitting. In addition, we used a decayed learning rate and a single dense hidden layer (ReLU activation) after the convolutional layers and before the output layer.

#### *Model Results*

We evaluated several state of the art transfer learning architectures including two variants of VGG16 ($300$-neuron dense hidden layer versus $1000$-neuron dense hidden layer), two variants of VGG19 ($300$-neuron dense hidden layer versus $1000$-neuron dense hidden layer), XCeption, InceptionResNetV2, and ResNet152V2. Initially, the models were trained with a low learning rate ($0.0000175$) so as to not "wipe out" the kernels learned during pre-training.

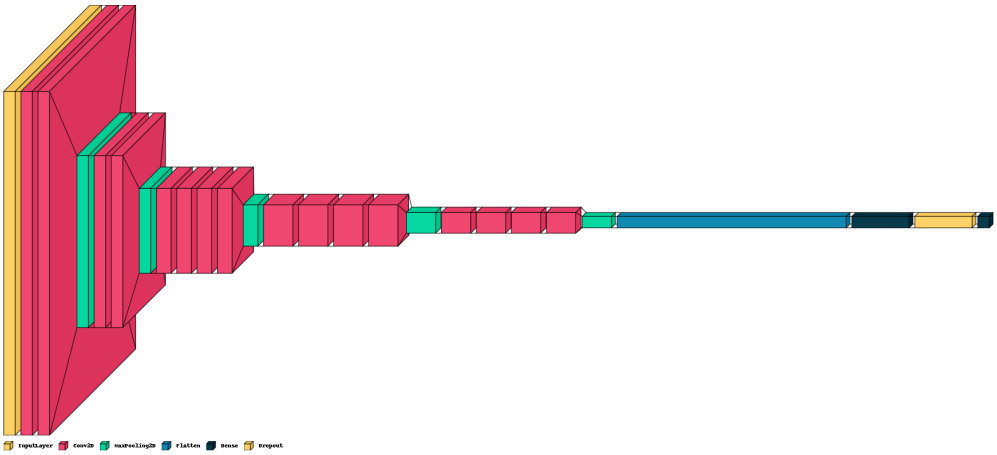
The VGG family of models outperformed the other models in virtually every metric category. In particular, the VGG19 model with a $1000$-neuron hidden dense layer performed best on the test set with top-1 scores of $73\%$ and $84\%$ for macro $F_1$ and accuracy, and top-2 scores of nearly $90\%$ and $95\%$ respectively.

| | Top 1 | | Top 2 | |
|---|---|---|---|---|
| **Model** | **Macro F1** | **Accuracy** | **Macro F1** | **Accuracy** |
| VGG16 (300 Neuron Hidden Layer) | 67.9 | 84.1 | 86.3 | 94.1 |
| VGG16 (1000 Neuron Hidden Layer) | 68.7 | 82.2 | 87.9 | 93.4 |
| VGG19 (300 Neuron Hidden Layer) | 67.1 | 84 | 89.4 | 94.6 |
| VGG19 (1000 Neuron Hidden Layer) | 73.1 | 84.2 | 89.2 | 94.7 |
| Xception | 61.2 | 79.8 | 83.4 | 91.4 |
| InceptionResNetV2 | 65.3 | 81.6 | 84.6 | 93.1 |
| ResNet152V2 | 64.9 | 82 | 83.9 | 92.6 |

When inspecting the class-wise performance of the VGG19 model, we noticed its consistent ability to detect the NV class, similarly to the classical models discussed above. However, its ability to accurately detect the six minority classes went above and beyond the classical models across all top-1 and top-2 metrics.

| | Top 1 | | | Top 2 | | |
|---|---|---|---|---|---|---|
| **Class** | **Precision** | **Recall** | **F1** | **Precision** | **Recall** | **F1** |
| AKIEC | 72 | 53 | 61 | 92 | 78 | 84 |
| BCC | 71 | 75 | 73 | 87 | 87 | 87 |
| BKL | 65 | 69 | 67 | 87 | 89 | 88 |
| DF | 71 | 85 | 77 | 76 | 95 | 84 |
| MEL | 57 | 61 | 59 | 86 | 92 | 89 |
| NV | 94 | 93 | 93 | 98 | 98 | 98 |
| VASC | 90 | 76 | 83 | 98 | 92 | 94 |

***VGG19 Architecure***

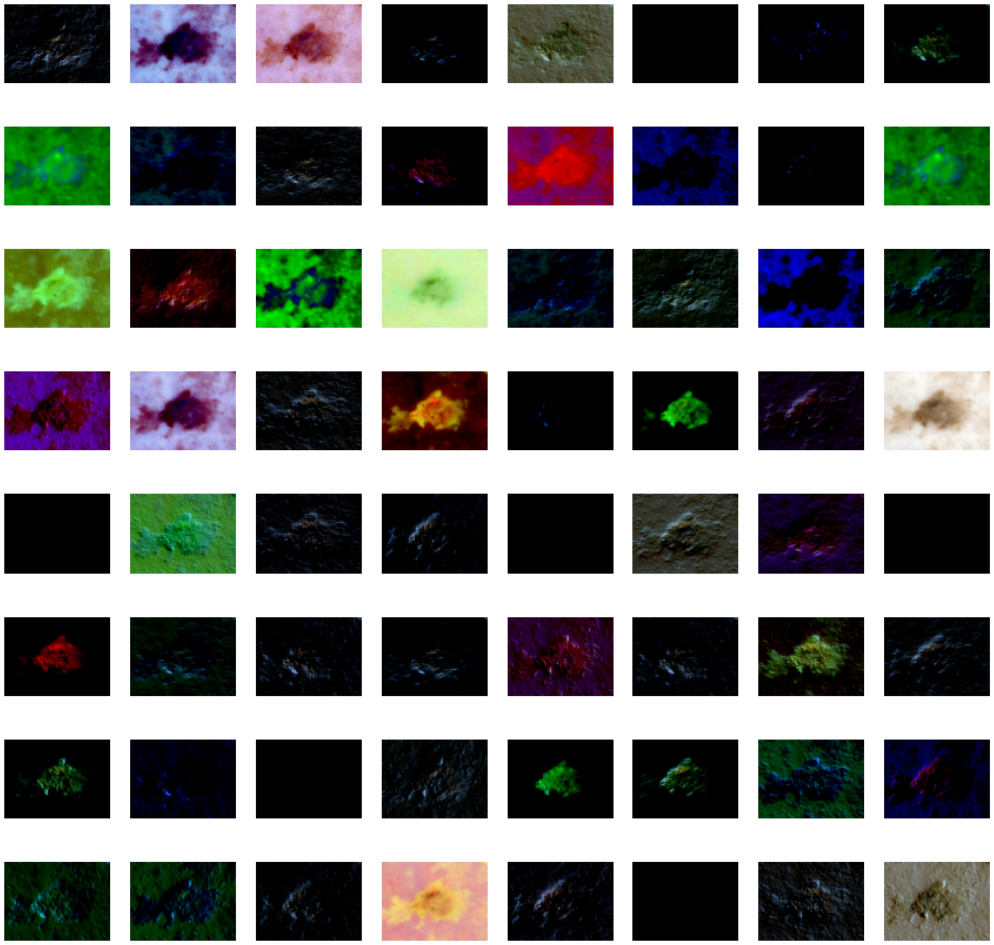



***Learned Kernels***

Once we had trained the deep learning models, we wanted to visualize the kinds of features that the models had learned to extract. To do so, we ripped out the kernels of the first convolutional layer of the large VGG19 model and convolved them with images in the dataset, visualized below. We can see that the model extracts a wide variety of features, such as:

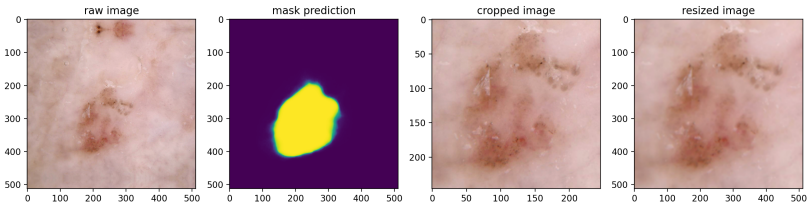
- The entire lesion body
- Edges of the lesion
- Gradients
- Texture extraction
- Raising the lesion to the foreground of the image
- Lowering the lesion (forming a sort of "canyon")

As we can see, convolutional models (and deep learning models in general) are powerful due to their ability to learn image representations that are optimal for a given task. In classical approaches, the onus was on us to decide what features to use; the models were only as good as the features we engineered. Deep learning models have an inherent flexibility that is not constrained by human imagination, which often leads to better performance.



**Segmentation-Informed Approach**

In addition to training classifiers on the raw images, we wanted to leverage the insight of our image segmentation models. We did this by using the segmentation mask predictions to crop the raw images. For a given lesion image, we would first feed it through the U-Net segmentation model to produce a segmentation mask. Since the U-Net model outputs probabilities for each pixel, we classified every pixel with a predicted probability greater than $0.5$ as belonging to the lesion, and every pixel as belonging to the background skin. Once the mask was generated, we would then find its boundary and remove any portions of the image outside. In case the segmentation prediction was poor, we always expanded the boundary by a $5\%$ before cropping. Ultimately, the cropped image was padded such that the $x$ and $y$ dimensions were equal, and resized to $515 \times 512$ (or sometimes $384 \times 384$ depending on the input dimensions of the model). The following plot illustrates this process:




***Model Architectures***

For this set of experiments, we restricted our attention to pre-trained image models, listed below. Whereas the U-Net model was initialized randomly, each of these models was initialized from a pre-trained checkpoint, with an additional $7$-output classification layer added on top.

1. Visition Transformer (ViT)
2. EfficientNet
3. Swin Transformer

Typically, the pre-trained portion of the model is left frozen during fine-tuning tasks, so as not to destroy the features it has learned. However, we suspected that since each pre-trained model was trained on a task with little similarity to skin lesion classification, retraining the models from scratch was sensible. Still, we made sure to throttle the learning rate until the models started converging. With a few exceptions, we found that this strategy achieved good results.

***Training Set-Up***

As with the image segmentation task, we used a five-fold cross-validation scheme. We also found success using the same hyperparameters: $3$ epochs, a batch size of $8$, a one-cycle learning rate schedule peaking at $2.0 \times 10^{-4}$, and a Rectified Adam optimizer. Our loss function, however, was different. Our target has seven classes, one for each different type of lesion. Therefore, it made sense to use cross-entropy as our training objective. Given a one-hot encoded $7 \times 1$ label $y$ and a $7 \times 1$ predicted label vector $\hat{y}$ (whose $i$-th entry corresponds to the probability that the image is a skin lesion of class $i$), the cross-entropy loss is given by

$$L(y, \hat{y}) = -\sum_{i=1}^{7} y_i \log \hat{y}_i.$$

After each epoch, we evaluated the model on the out-of-fold data and computed the average loss. At any given point, we only saved the model if the average validation loss had improved over the previous epoch. For more details, please refer to the notebook `w281_final_project/romain/classification_deep_learning.ipynb`.

***Results***

In the following table, we show the test accuracy and macro F1 scores of each pre-trained model. The first column lists the model checkpoint, and the second column contains the number of parameters in the model. We show the top-1 metrics as well as the top-2 metrics (which are calculated by taking the two classes with the highest predicted probabilities as the final prediction).

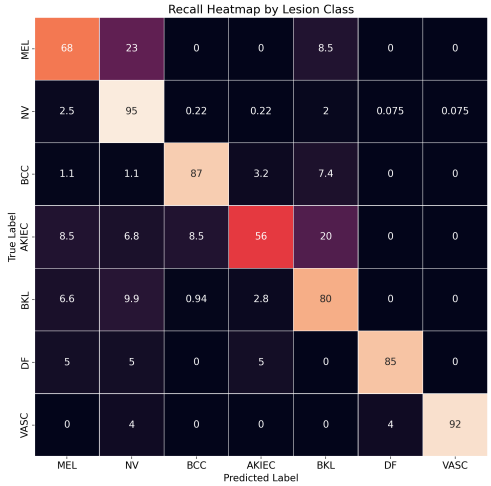
| Model | Parameters | Top-1 Macro F1 | Top-1 Accuracy | Top-2 Macro F1 | Top-2 Accuracy |
|---|---|---|---|---|---|
| efficientnet_b1 | 6.5M | 72.39 | 84.99 | 82.42 | 94.69 |
| tf_efficientnetv2_b1 | 6.9M | 65.26 | 83.00 | 77.35 | 94.13 |
| tf_efficientnetv2_b3 | 12.8M | 70.48 | 85.20 | 85.89 | 95.61 |
| swinv2_small_window16_256 | 50.0M | 81.80 | 88.72 | 90.73 | 96.68 |
| vit_base_patch8_224_dino | 85.8M | 77.39 | 86.68 | 87.15 | 95.61 |
| vit_large_patch32_384 | 305.6M | 81.86 | 88.62 | 90.42 | 96.84 |
| vit_large_patch32_384 | 305.6M | 81.86 | 88.62 | 90.42 | 96.84 |

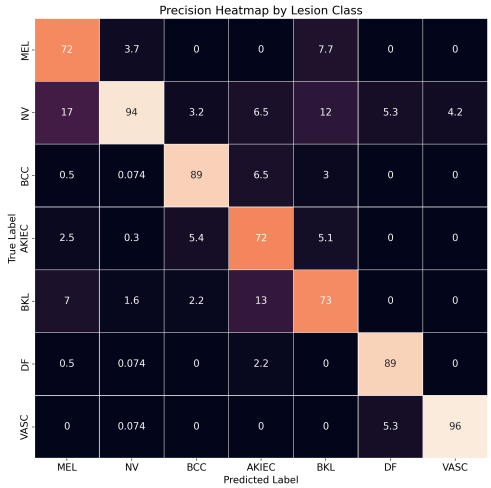
Immediately, we notice that each model demonstrates admirable performance on the test set, with most top-1 macro F1 scores lying above $70$. Increasing the complexity of the model appears to improve model performance, even when the underlying architecture remains the same. For example, EfficientNetV2-B3 achieves a top-1 macro F1 score that is $5.22$ points higher than its B1 counterpart, despite having nearly twice as many parameters.

Notably, the convolutional networks are outperformed by the vision transformers; the large ViT model and the Swin transformer have top-1 macro F1 scores nearly $10$ points higher than the best convolutional network. One possible explanation is that attention is better than convolutions at capturing context relevant to classifying skin lesions. However, it is also possible that convolutional architectures could match the performance of vision transformers with additional hyperparameter tuning.

Although the different models seem closely matched in terms of accuracy, this metric is misleading when taken at face value. Unlike the macro F1 score, accuracy is biased towards the majority class; by comparison, errors on the minority classes matter less. Therefore, the overall accuracies of the models appear similar, despite the fact that the vision transformers have, on average, better performance across the different lesion classes.

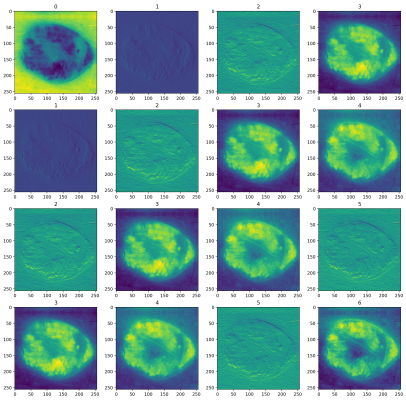
We conclude this section with a short analysis of the per-class performance of the large ViT model. The first figure below shows a recall heatmap of its test set predictions, where each cell at position $(i, j)$ is the percentage of samples with true label $i$ that are predicted as belonging to class $j$ (each row sums to $100\%$). The second figure is a precision heatmap; here, the $(i, j)$-th entry is the percentage of samples with predicted label $j$ that have true label $i$ (each column sums to $100\%$). With a few exceptions, the model demonstrates impressive performance across all classes. Two pairs of classes that the model sometimes confuses are NV, MEL and BKL, AKIEC. $23\%$ of MEL lesions are mistaken as NV, while $20\%$ of AKIEC classes are mistaken as BKL. The first mistake is potentially quite dangerous, as it could lead to missed diagnoses of malignant lesions.

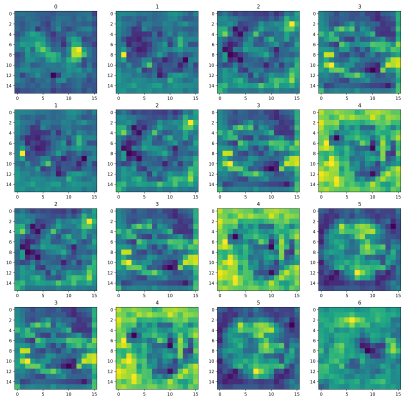



***Learned Kernels***

Although neural networks often operate as black boxes for a given task, we can get a glimpse of the learned features by visualizing images at different stages throughout the model. Below, we display the first $16$ consecutive channels of an image after it has gone through one convolutional layer (top) and three convolution layers (bottom) of an EfficientNet model.





The exact physical interpretation of these features is ambiguous, especially in the lower-dimensional representation. Clearly, the model captures information relating to skin texture, intensity, and gradients. What is not apparent is how the model combines this information to make its final prediction. In addition to being highly performant on held-out data, models should also seek to be explainable. This is especially true in medicine, as diagnoses need to be validated by experts and relayed to patients. Future work in this domain should seek to better understand the decision-making pathways of large convolution and transformer models.

## Future Work

1. We could improve our deep learning model with additional fine-tuning of hyperparameters such as the number of layers, and type of layers. Additional experiments could help bridge the performance gap between CNNs and vision transformers.
2. We could work on developing a portable application that takes an image of a skin lesion and transmits it to a cloud-hosted classifier, returning the predicted diagnosis.
3. Limited physical access to health-care providers during the recent COVID-19 pandemic has prompted changes in health-care delivery and is accelerating the adoption of telemedicine. AI-based triage and decision support could assist doctors in managing workloads and improving their diagnostic performance.
4. The HAM10000 dataset is dominated by patients with lighter skin tones and dark lesions. Including samples of darker skin colors (or artificially generated images) would reduce the dataset's bias and help improve our models' ability to generalize.
5. Model explainability remains a significant challenge in deep learning. Further work is needed to understand why models make certain decisions, so that doctors can validate those and explain them to patients.
6. Our best models still struggle with certain classes. Expanding the dataset and experimenting with improved architectures may help to improve performance on these segments.
7. It would be interesting to analyze skin lesions over time, as there may be evolutionary patterns of the skin that could help us better identify the presence of skin lesions.

## Conclusions

In this report, we have applied classical and deep learning techniques to the tasks of segmenting and classifying skin lesions in the HAM10000 dataset. On the classical side, we created an AutoML framework for identifying optimal model and preprocessing sequences for any given model type. On the deep learning side, we experimented with two approaches for classification, a downsampling approach to reduce the input size and a segmentation-informed approach to localize the lesion before feeding it to a model. Our best segmentation model was a U-Net model that achieved $97.1\%$ per-pixel accuracy, while our best classification model was a segmentation-informed ViT model that achieved macro $F_1$ and accuracy scores of $81.9$ and $88.62\%$ on the test set, respectively. This considerably improves upon the baseline accuracy of human raters, which was estimated to be $63.6\%$.

## References:


1. Skin cancer: HAM10000 (https://www.kaggle.com/datasets/surajghuwalewala/ham1000-segmentation-and-classification)
2. HAM10000 Lesion Segmentations (https://www.kaggle.com/datasets/tschandl/ham10000-lesion-segmentations)
3. HAM10000 Dataset in Harvard Dataverse (https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T)
4. Tschandl, P. et al. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5:180161 doi: 10.1038/sdata.2018.161 (2018) (https://www.nature.com/articles/sdata2018161)
5. ISIC Challenge Datasets (2018) (https://challenge.isic-archive.com/data/#2018)
6. International Skin Imaging Collaboration (ISIC) (https://www.isic-archive.com/#!/topWithHeader/tightContentTop/about/aboutIsicOverview)
7. HAM10000 Dataset: (c) by ViDIR Group, Department of Dermatology, Medical University of Vienna (https://doi.org/10.1038/sdata.2018.161)
8. Shetty, B., Fernandes, R., Rodrigues, A.P. et al. Skin lesion classification of dermoscopic images using machine learning and convolutional neural network. Sci Rep 12, 18134 (2022) (https://doi.org/10.1038/s41598-022-22644-9)
9. Tschandl, P., Rinner, C., Apalla, Z. et al. Human–computer collaboration for skin cancer recognition. Nat Med 26, 1229–1234 (2020) (https://www.nature.com/articles/s41591-020-0942-0)
10. Hair Removal Algorithm (https://github.com/BlueDokk/Dullrazor-algorithm)
11. Skin Lesions Classifier (https://github.com/biagiom/skin-lesions-classifier)
12. Skin lesions: HAM 10000 dataset BIODS220 Project (https://web.stanford.edu/class/archive/biods/biods220/biods220.1204/autumn2020/biods220_butskova.pdf)
13. CBIR Skin Cancer Lesions (https://www.kaggle.com/code/jaimemorillo/cbir-skin-cancer-lesions)
14. MFSNet: A Multi Focus Segmentation Network for Skin Lesion Segmentation (https://arxiv.org/abs/2203.14341v2)
15. A DE-ANN Inspired Skin Cancer Detection Approach Using Fuzzy C-Means Clustering (https://link.springer.com/article/10.1007/s11036-020-01550-2)

16. Khan, M. A. et al. Attributes based skin lesion detection and recognition: A mask RCNN and transfer learning-based deep learning framework. Pattern Recogn. Lett. 143, 58–66 (2021) (https://doi.org/10.1016%2Fj.patrec.2020.12.015)
17. Rosendahl, C., Tschandl, P., Cameron, A. & Kittler, H. Diagnostic accuracy of dermatoscopy for melanocytic and nonmelanocytic pigmented lesions. J Am Acad Dermatol 64, 1068–1073 (2011) (https://doi.org/10.1016%2Fj.jaad.2010.03.039)
18. Binder, M. et al. Application of an artificial neural network in epiluminescence microscopy pattern analysis of pigmented skin lesions: a pilot study. Br J Dermatol 130, 460–465 (1994) (https://doi.org/10.1111%2Fj.1365-2133.1994.tb03378.x)
19. Vienna Dermatologic Imaging/Informatics Research (https://www.meduniwien.ac.at/hp/dermatologie/wissenschaft-forschung/vienna-dermatologic-imaging-research-group-vidir/)
20. Tajeddin et al. Melanoma recognition in dermoscopy images using lesion's peripheral region information (https://doi.org/10.1016/j.cmpb.2018.05.005)
21. N. Z. Tajeddin and B. M. Asl, "A general algorithm for automatic lesion segmentation in dermoscopy images," 2016 23rd Iranian Conference on Biomedical Engineering and 2016 1st International Iranian Conference on Biomedical Engineering (ICBME), 2016, pp. 134-139, doi: 10.1109/ICBME.2016.7890944. (https://ieeexplore.ieee.org/document/7890944)
22. K. Korotkov, R. Garcia, Computerized analysis of pigmented skin lesions: a re- view., Artif. Intell. Med. 56 (2012) 69–90 (10.1016/j.artmed.2012.08.002)
23. I. Maglogiannis , C.N. Doukas , S. Member , Overview of advanced computer vi- sion systems for skin lesions characterization, IEEE Trans. Inf. Technol. Biomed. 13 (2009) 721–733 (https://ieeexplore.ieee.org/document/4801738)
24. N.K. Mishra, M.E. Celebi, An overview of melanoma detection in der- moscopy images using image processing and machine learning (https://arxiv.org/abs/1601.07843)
25. M. Silveira et al., "Comparison of Segmentation Methods for Melanoma Diagnosis in Dermoscopy Images," in IEEE Journal of Selected Topics in Signal Processing, vol. 3, no. 1, pp. 35-45, Feb. 2009, doi: 10.1109/JSTSP.2008.2011119. (https://ieeexplore.ieee.org/abstract/document/4786545)
26. J. Daugman, "How iris recognition works," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 14, no. 1, pp. 21-30, Jan. 2004, doi: 10.1109/TCSVT.2003.818350. (https://ieeexplore.ieee.org/document/1262028)
27. Löfstedt T, Brynolfsson P, Asklund T, Nyholm T, Garpebring A. Gray-level invariant Haralick texture features. PLoS One. 2019 Feb 22;14(2):e0212110. doi: 10.1371/journal.pone.0212110. PMID: 30794577; PMCID: PMC6386443 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6386443/#pone.0212110.ref028)
28. Transfer Learning: CS231n: Convolutional Neural Networks for Visual Recognition (https://cs231n.github.io/transfer-learning/)
29. Skin Cancer - Cancer.org (https://www.cancer.org/cancer/skin-cancer.html)
30. Lan Z, Cai S, He X, Wen X. FixCaps: An Improved Capsules Network for Diagnosis of Skin Cancer. IEEE. 2022 June 4;10.1109/ACCESS.2022.3181225 (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9791221)
31. Basak H, Kundu R, Sarkar R. MFSNet: A Multi Focus Segmentation Network for Skin Lesion Segmentation. eess.IV. 2022 March 29; doi.org/10.48550/arXiv.2203.14341 (https://doi.org/10.48550/arXiv.2203.14341)

***Note:*** HAM10000 Dataset is hosted in multiple sites, we gained critical insights from each citation, hence listed here

### Glossary:

- LBP: Local Binary Pattern
  - LBP (https://en.wikipedia.org/wiki/Local_binary_patterns)
  - Local Binary Pattern Algorithm: The Math Behind It ❗ (https://medium.com/swlh/local-binary-pattern-algorithm-the-math-behind-it-%EF%B8%8F-edf7b0e1c8b3)
- GLCM: A statistical method of examining texture that considers the spatial relationship of pixels is the gray-level co-occurrence matrix (GLCM), also known as the gray-level spatial dependence matrix.)
  - GLCM (https://www.mathworks.com/help/images/texture-analysis-using-the-gray-level-co-occurrence-matrix-glcm.html)
- ABCDE:
  - ABCDE (https://www.healthline.com/health/skin-cancer/abcd-rule-for-skin-cancer#)