UNIVERSITY OF CALIFORNIA

Los Angeles

An Application of Split Attention Networks:

Melanoma Detection

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Applied Statistics

by

Andrew Mashhadi

2023

ABSTRACT OF THE THESIS

An Application of Split Attention Networks:

Melanoma Detection

by

Andrew Mashhadi

Master of Applied Statistics

University of California, Los Angeles, 2023

Professor Yingnian Wu, Chair

(Abstract temporarily omitted)

The thesis of Andrew Mashhadi is approved.

Michael Tsiang

Frederic Paik Schoenberg

Yingnian Wu, Committee Chair

University of California, Los Angeles

2023

TABLE OF CONTENTS

LIST OF FIGURES

vi

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Background

In the past decade, machine learning has exploded in popularity. Machine learning methods have demonstrated endless applciations to a variety of industries including but not limited to engineering, science, finance, medicine, and technology. A large branch of machine learning that has recently taken the world by storm is *deep learning*. Deep learning is made up of *artificial neural networks* (ANN) and is generally trained with a form of *feature learning*.

Neural network architures have evolved and expanded since the original perceptron and ANN models. Through different areas of application, variations in neural network architecture such as *deep neural networks*, *convolutional neural networks*, *recurrent neural networks*, and more recently *transformers*, have all been proposed and adopted. It is no secret that the transformer model has quickly become a front-runner for applications in computer vision due to its successes in natural language processing. However, deep convolutional neural networks are still state-of-the-art for tasks such as image classification, object detection, semantic segmentation, and instance segmentation

Different learning methods have also been adopted. However, the type of learning method used generally depends on the model's particular objective. *Supervised Learning* is one of the most common methods for classification or regression models. It is the process of using labeled datasets to train machine learning models to classify or predict outcomes appropriately. *Unsupervised Learning* generally involves the analysis or clustering of unlabeled datsets. And *Reinforcement Learning* is based on rewarding desired behaviors and/or punishing undesired ones.

More recently, the medical community has been opening it's doors to modern deep learning techniques. In particular, convolutional neural networks (CNNs) have often been used to develop more efficient, and accurate, diagnostic tools to analyze medical images. Due to the growth of effective image recognition models, collection of medical images for specific applciations in the healthcare community have been been growing.

1.2   Problem Statement

Skin cancer is one of the most common types of cancer. Although melanoma only accounts for about 1% of skin cancer, the death rate was still about 2.1 per 100,000 men and women per year based on 2016-2020 deaths [Ins23]. In 2023, the American Cancer Society estimates that bout 7,990 people are expected to die from a total of about 97,610 new melanoma cases in the United States alone [Soc23]. It is well known that early detection of melanoma will provide the best chance for successful treatment and greater chance of survival.

Image analysis tools that automate the diagnosis of melanoma will improve dermatologist's diagnostic accuracy, and better detection of melanoma has the opportunity to positively impact millions of people. Providing an accurate machine learning model to aid dermatologist's in their evaluations of patients moles may lead to an earlier diagnoses, and could therefore provide the best chance for appropriate intervention. We want to use a new deep convolutional-neural-network architecture that uses patients skin-lesion images along with any other patient-level contextual information to determine which patients are likely to have melanoma skin cancer. In particular, the goal of this paper is to identify the presence of melanoma from images of skin lesions using the novel *ResNeSt* architecture that was designed to apply channel-wise attention on different network branches.

CHAPTER 2

Dataset

2.1 ISIC 2020 Challenge Dataset

For this paper, we used the "ISIC 2020 Challenge Dataset". This was the official dataset of the SIIM-ISIC Melanoma Classification Challenge hosted as a Kaggle sponsored competition in the Summer of 2020. It contains over 30,000 dermoscopic images of distinct skin lesions from approximately 2,000 patients. Each image has an associated record of metadata. The metadata includes the corresponding "beniegn" or "malignant" status, as well as the following patient-level features:

- *patient_id*: unique patient idenitfier

- *sex*: sex of the patient

- *age_approx*: approximate age of the of the patient

- *anatom_site_general_challenge*: the general location of imaged lesion

- *diagnosis*: additional details regarding the diagnosis

We should note that all associated malignant and benign diagnoses have been confimed using histopathology, expert agreement, or longitudinal follow-up [ISI20].

The International Skin Imaging Collaboration (ISIC) was responsible for compiling the images from the Hospital Clínic de Barcelona, Medical University of Vienna, Memorial Sloan Kettering Cancer Center, Melanoma Institute Australia, University of Queensland, and the University of Athens Medical School to form this official dataset. The ISIC Archive contains

the largest collection of quality-controlled dermoscopic images of skin lesions available to the public.

The resolution of each image varied drastically throughout the dataset, with some images reaching as high as 4000×6000 pixels. The set of images consisted of over 110GB, so we hosted the data on UCLA's Hoffman2 Linux Cluster. To support the size of each image and the large number of individual images within the dataset, we used computing resources from the Hoffman2 cluster and Google Colab to tune, train, and test our models with powerful *Graphical Processing Units* (GPUs). Although the GPU would occasionally change, we mostly used a single NVIDIA A100 GPU with approximately 40GB VRAM to train our larger models.

2.2   Data Preparation

Throughout training, we found it to be difficult to train our models without batch-sizes greater than or equal to 16 images. However, computational constraints limited our ability to utilize larger batch-sizes with such large lesion images. We ultimately found that the only way to train efficiently with suffcient batch-size ($m \geq 16$) was to center crop, or resize (using bilinear interpolation), each original lesion image to a fixed-sized 512×512 image. Further transformations and resizing are discussed in the methodology section.

The patient-level features within the metadata also required minor preprocessing. We created dummy variables with treatment coding (also known as *one-hot encoding*) for the *sex* and *anatom_site_general_challenge* features, while keeping *age_approx* as a continuous variable.

2.3   Exploratory Data Analysis

In this section, we provide an exploratory data analysis of both the images and the patient-level features within the metadata provided in our dataset.

2.3.1 Skin Lesion Images

The images of skin lesions are the most important part of data for our model. They will be used to train the image classification network, so we decided to investigate the data a bit more, before beginning to model. First, we investigated the proportion of "positives" (malignant) and "negative" (beniegn) images within our dataset. Table 2.1 presents the balance of malignant and beniegn skin lesions within our dataset. As we can see from the table, there is significantly large class imbalance. Only about 1.8% of the skin lesion image are classified as malignant.

	Beniegn ($-$)	Malignant ($+$)
Count	32542	584
Proportion	0.9824	0.0176

Table 2.1: Proportion of Beniegn & Malignant Skin Lesions

As mentioned above, we fixed the resolution of the images in our dataset to 512x512 pixels. We visually inspected the quality of several randomly selected skin lesions images after cropping and resizing to make sure that the *object of interest* (the actually skin lesion itself) was still present and easily distinguishable. In Figures 2.1-2.2 , we present 6 images from our dataset. The top 3 skin lesions were randomly selected from the images flagged as malignant, while the bottom 3 skin lesisons were randomly selected from the images marked as beniegn. Although the relative and absolute size of the skin lesions differ, they are all clearly distinguishable in the images.

We also visually compared a random collection of images flagged as malignant with another random collection of images flagged as benign. The objective here was to see if our "untrained" eye could easily classfiy the skin lessions as malignant or beniegn. We can see from Figures 2.1-2.2 that without proper training and eductaion, it is very difficult to determine whether or not a skin lesion is considered malignant. Even with the proper training, it can still be difficult to accurately determine if melanoma is present. According to the Skin Cancer Foundation, features such as assymmetry shapes, irregular borders, uneven
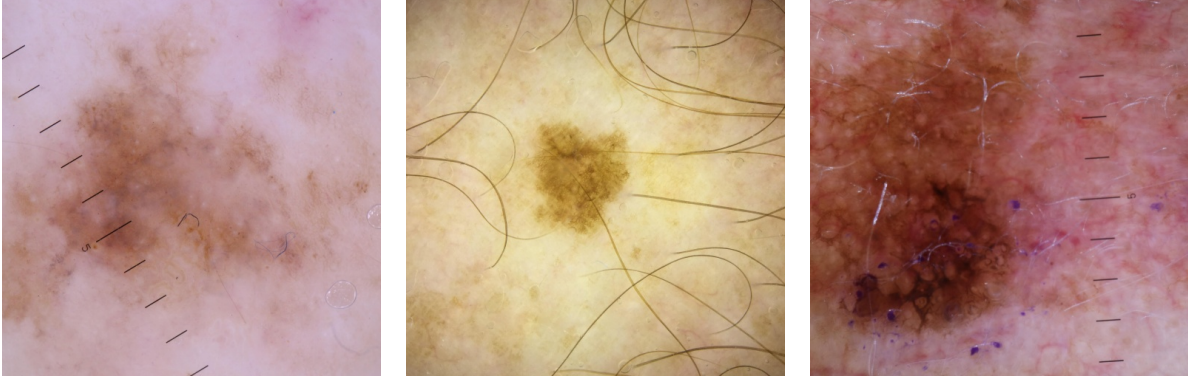
Figure 2.1: Examples of Malignant Skin Lesions
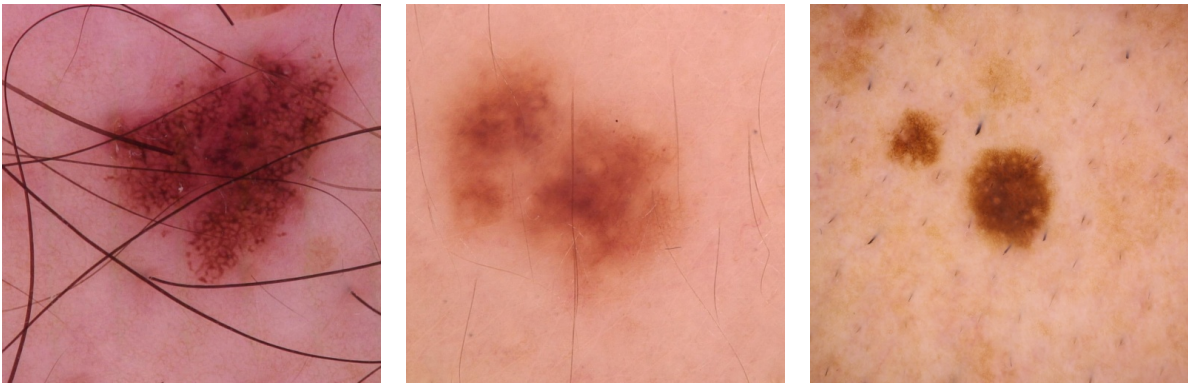
Figure 2.2: Examples of Beniegn Skin Lesions

distribution of color, and large relative size may indicate the presence of melanoma (or other skin cancers) [Fou23]. Ideally, our network will pick up on these characteristics while training.

2.3.2 Patient-Level Features

In this section we explore the one-way frequency tables for the approximate ages, patient sexes, and the locations of imaged lesions. Then we explored the two-way frequency tables for each of these variables with the response variable (malignant or beniegn).

Patient Age

We can see from the histogram of approximate ages in Figure 2.3(a) that most of the patients in this data are in their 40's, 50's, and 60's. The box-plot in Figure 2.3(b) clearly indicates that older patients are generally associated with a larger number of malignant skin lesions.

(a)

(b)
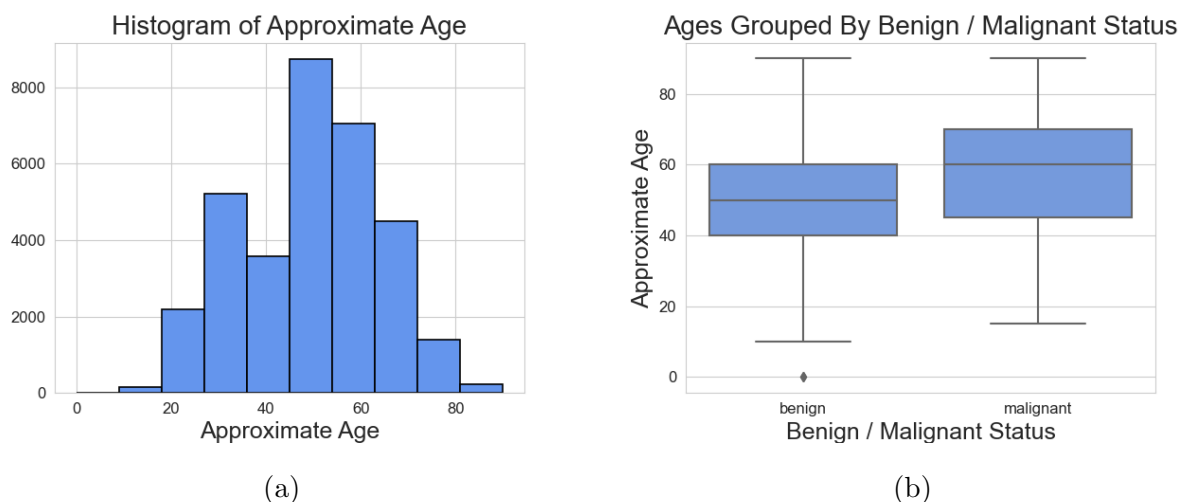
Figure 2.3: Histogram and Box-Plot For Appoximate Age

Patient Sex

We can see from the frequencies shown in Figure 2.4(a) that there are a similar number of males and females in this dataset, with just one or two thousand more males. The two-way contingency table between the response variable and patient sex is also shown in Figure 2.4(b). We can see that the malignant cases have a significantly larger proportion of males than the beniegn cases do. In fact, the χ^2-test of independence reported a p-value very close to 0. Therefore, we reject the hypothesis that patient sex and the presence of melanoma are independent, and conclude that there is a relationship between the two variables.

Skin-Lesion Location

There were 6 categories for the general skin-lesion location: oral/genital, palms/soles, head/neck, upper extremity, lower extremity, and torso. Figure 2.5(a) presents the associated frequen-
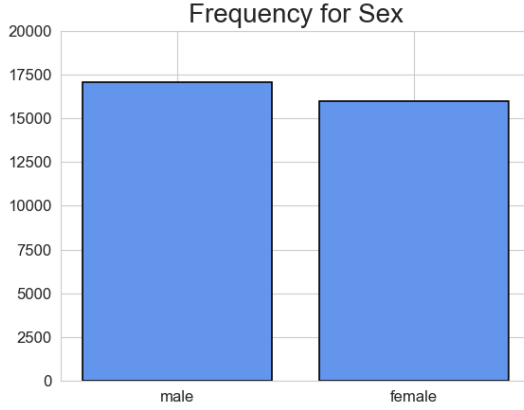
	Beniegn	Malignant
Female	15761	220
Male	16716	364

(a)   (b)

Figure 2.4: Frequency Plot and Two-way Contingency Table for Patient Sex

cies within our dataset. None of the general locations have a similar number of counts, and almost half of the skin-lesions come from the torso region. We also present the two-way contingency table between the response variable and lesion location in Figure 2.5(b). While it is somewhat difficult to tell, the does appear to be some differences in the balance between beniegn and malignant cases when conditioned on the skin-lesion location. In this case, the χ^2-test of independence also reported a p-value very close to 0, indicating that there is a relationship between the location of the skin-lesion and the presence of melanoma.
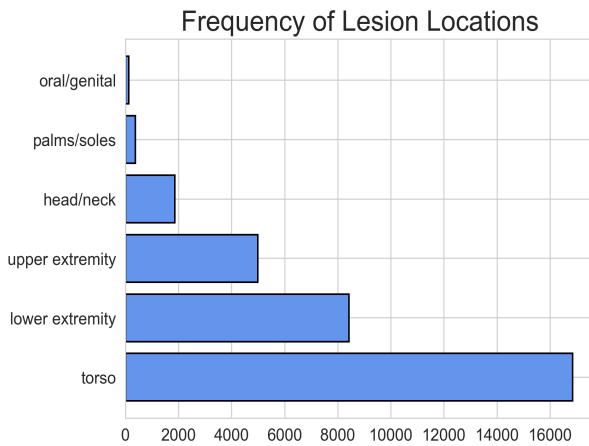
	Beniegn	Malignant
Head/Neck	1781	74
Lower Extremity	8293	124
Oral/Genital	120	4
Palms/Soles	370	5
Torso	16588	257
Upper Extremity	4872	111

(a)   (b)

Figure 2.5: Frequency Plot and Two-way Contingency Table for Lesion Location

CHAPTER 3

Methodology

In this section, we describe the methods and models used to best predict the presence of melanoma. As mentioned above, we used deep convolutional neural networks to generate classifications. More specifically, the latest variant of the residual network, the *ResNeSt*, was used to assess whether or not its channel-wise attention architecture would find further success in melanoma detection. Additionally, we trained a smaller, more traditional, convolutional neural network to compare the performance with the split-attention network.

To make use of the additional metadata features within our dataset, we simultaneously trained a standard *multi-layer perceptron* (MLP) with inputs from our patient-level features. Therefore, the deep convolutional networks is used to effectively extract any contextualized features from the skin-lesion images and the multi-layer perceptron is used to extract any important patient-level information. The two networks are then connected with eachother before generating a final probability. Since the outputted probabilities are a result from two seperate networks communicating with eachother, this model may be described as a *Multi-Network Ensemble*.

3.1 Training, Validation, and Testing Sets

Before discussing the actual models used, we breifly discuss the how the dataset was split and sampled for training, validating, and testing.

We first randomly split 80% of the data into a training set and used the other random 20% as the test set. The test set was not used by any network during the training process. Additionally, we partitioned another 20% of the training data into a validation set. Recall

9

that our dataset is largely imbalanced. As shown in the Exploratory Data Analysis section, only about 2% of the skin lesions in the dataset are flagged as malignant, while the remaining skin lesions are marked as benign. We ensured that each split of the data had a similar sample proportional of about 2% malignant skin lesions.

Since the response variable (malignant or beniegn) in the training data is heavily imbalanced, we used randomized oversampling of the malignant observations to combat any potential bias toward a "beniegn" classification. By oversampling, we hoped that the networks would better detect when skin lesions were truly malignant. It's important to note that since oversampling was used, we considered a single epoch as when the entire *oversampled* dataset was passed forward and backward through the network exactly once. Therefore, a single epoch will contain multiple repeated malignant skin lesion images.

Note that using standard k-fold cross validation would require a model to be trained k times for each set of hyper-parameters. Given the size of our data and our models, the time it would take to perform this procedure was simply infeasible. Therefore, we used the independent validation set to tune each network's hyper-parameters effectively. This method minimized overall training time, and produced optimal hyper-parameters. Once the hyper-parameters were obtained, we retrained the model on a new oversampled dataset made up from combining the training and validation sets.

We tuned each model with the area under the *Receiver Operating Characteristic* (ROC-AUC) as our performance metric. The ROC-AUC was chosen over other metrics because it summarizes how well our network separates the two response classes over *all* possible thresholds (as opposed to a single predetermined threshold).

3.2   Image Augmentations

Image augmentations were used as a preprocessing step for each image before training. Some image augmention methods are often used to alter the original images in the dataset to effectively create more "unseen" examples for the network to use while training. These techniques artificially extends the dataset by randomly providing alerations to existing data,

and consequently reduces the chance of overfitting. Given the limited number of malignant examples, this technique is very important to artificially extend our malignant skin-lesion training images. In our networks, we randomly employed the following image augmentations (in this order) on our training data [AI23]:

1. **HueSaturationValue**: With probability $p = 0.5$, we randomly changed the hue, saturation, and value of the input image. The shift in hue was between $(-5, 5)$, the shift in saturaion was between $(-10, 10)$, and the shift in value was between $(-5, 5)$.

2. **FancyPCA**: With probability $p = 0.2$, perform *principal component analysis* on the set of RGB pixel values throughout the imput image, then add multiples of the found principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from $\alpha_1, \alpha_2, \alpha_3 \overset{i.i.d}{\sim} \mathcal{N}(0, 0.1)$.

3. **VerticalFlip**: With probability $p = 0.5$, we veritially flipped the imput image. Note that vertically flipping the image should not change the true class of the skin lesion.

4. **HorizontalFlip**: With probability $p = 0.5$, we veritially flipped the imput image. Note that vertically flipping the image should not change the true class of the skin lesion.

5. **GaussNoise**: we applied gaussian noise to the input image. We did not want to corrupt the image too much and to ooften, so we used a mean of 0 and a variance between $(10, 50)$ with a low probability $p = 0.2$.

6. **ShiftScaleRotate**: With probability $p = 0.5$, we randomly applied the three affine transforms: translate, scale and rotate the input. The shift factor range for both height and width was set to $(-0.25, 0.25)$, the scaling factor range was set to $(-0.25, 0.25)$, and the rotation range was set from -30 degrees to 30 degrees.

7. **RandomBrightnessContrast**: With probability $p = 0.5$, we randomly modified the brightness and contrast of the input image. Since many of the images were already very dark, the factor range for changing brightness was set to $(0.9, 1.1)$. The factor range for changing contrast was also set to $(0.9, 1.1)$.

Some image augmentations techniques are also used to make sure the images in the dataset work with our models, and may promote faster convergence. For instance, in addition to the above augmentations, we also randomly cropped each 512×512 image to a 416x416 image and normalized the pixel values to have a similar distribution. Although the randomness of the cropping does somewhat "artificially" extend our training set, the main point of cropping the images to 416×416 is because the ResNeSt network we used requires an input size of 416×416 pixels. The normalization of the pixel values will reduce the risk of exploding gradients, which has been shown to increase training time and generally slows down convergence. We normalized each channel using the associated sample means and sample standard deviations calculcated from the *ImageNet* database. The ImageNet database consists of millions of images and was designed for use in visual object recognition software research. Although we could have used the sample means and sample standard deviations from our own dataset, using the sample statistics from the ImageNet database is common practice and was recommended by the authors of the *ResNeSt* network.

We should note that most of the above image augmentations were only used on the training set. The validation and testing sets only center cropped the images to 416×416 and normalized the pixels to work with the trained network.

3.3 Network Architectures & Training

In the section, we provide an overview of the foundational neural network architectures and we breifly discuss how our networks were trained. We also introduce a few of the techniques we employed to combat overfittins.

3.3.1 Multi-Layer Perceptrons

As illustrated in Figure 3.1, a standard multi-layer perceptron (MLP) model consists of an input layer and an output layer, with at least one hidden layer in between.

The lth layer within the network may be represented by a vector, $\mathbf{h}^{(l)}$, which is obtained

12
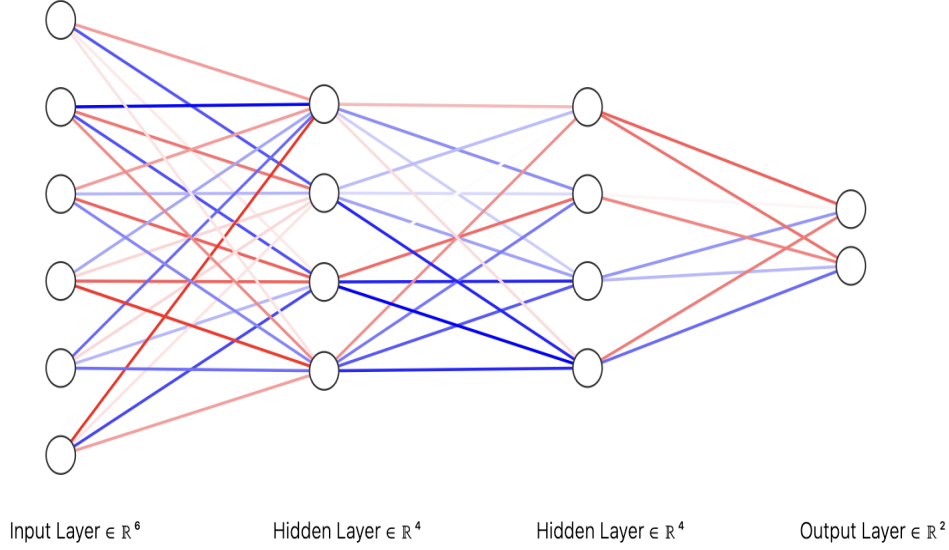
Figure 3.1: A Standard Feed-Forward Network

by applying a Sigmoid or ReLU transform to the product of the $(l-1)$th layer and a weight matrix, $\mathbf{W}^{(l)}$, plus a bias vector, $\mathbf{b}^{(l)}$. That is, the general MLP has the following recursive structure,

$$\mathbf{h}^{(l)} = f_l(\mathbf{s}^{(l)})$$

$$\mathbf{s}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$$

for $l = 1, 2, ..., L$. Here, L is the number of layers in the network, $\mathbf{h}^{(0)}$ represents our input features $\mathbf{X}$, $\mathbf{h}^{(K)}$ represents the outputted probabilites used to make the final calssification for our response, $\mathbf{Y}$, and f_l is an element-wise transformation. Following industry standards, our networks used the ReLU activation function for all hidden layers such that $f_l(s_j^{(l)}) = \text{ReLU}(s_j^{(l)}) = \max(0, s_j^{(l)})$ for $l = 1, 2, ..., L-1$ and the softmax function (or sigmoid function if $s^{(K)} \in \mathbb{R}$) for the output layer transformation such that the jth outputted probability is:

$$\hat{p}(\mathbf{X})_j = f_K(\mathbf{s}^{(L)})_j = \frac{e^{s_j^{(L)}}}{\sum_i e^{s_i^{(L)}}} \in (0,1)$$

3.3.2 Methods of Learning

In general, neural networks are trained by minimizing a chosen loss function L, with some form of gradient descent. The gradients for each layer's weights $\mathbf{W}_l$ and bias $\mathbf{b}_l$ are calculated using *back-propogation*. Back-propogation is essentially the process of applying chain rule as many times as necessary to obtain the gradient for a specified weight or bias. Given the network structure described above, the following relationships between gradients can be easily shown with chain rule:

$$\frac{\partial L}{\partial \mathbf{h}^{(l-1)\,T}} = \frac{\partial L}{\partial \mathbf{h}^{(l)\,T}} f_l' \mathbf{W}^{(l)}, \qquad \frac{\partial L}{\partial \mathbf{W}^{(l)}} = f_l' \frac{\partial L}{\partial \mathbf{h}^{(l)}} \mathbf{h}^{(l-1)\,T}, \qquad \frac{\partial L}{\partial \mathbf{b}^{(l)}} = f_l' \frac{\partial L}{\partial \mathbf{h}^{(l)}}$$

where $f_l' = \mathrm{diag}(f'(s_j^{(l)}))$; for $j = 1, 2, ...d)$.

For notational convenience, we can represent our training data as $(\mathbf{x}_i, y_i)$ for $i = 1, 2, ..., N$ and our network parameters as $\theta = (\mathbf{W}_l, \mathbf{b}_l$; for $l = 1, 2, ...d)$. Since we have a binary response variable (the skin lesion is either malignant or beniegn), we trained our networks using the *Binary Cross-Entropy* loss function defined as:

$$L(y_i, \mathbf{x}_i; \theta) = -\left[y_i \log(\hat{p}(\mathbf{x}_i)) + (1 - y_i) \log(1 - \hat{p}(\mathbf{x}_i)) \right]$$

As mentioned above, this loss function is used to train the network's parameters with a form of gradient descent. With $\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \mathbf{x}_i; \theta)$ as the averaged loss over the entire training set, the standard gradient descent algorithm has the following form:

$$\theta_{t+1} = \theta_t - \eta_t \mathcal{L}'(\theta_t)$$

where $\mathcal{L}'(\theta_t)$ is the gradient and η_t is the learning rate ($\eta_t \propto \frac{1}{t}$). The general idea of this algorithm is to pass the training data through the network (known as the *forward pass*), use these predictions with the loss function to calculate the resulting gradients (known as the *back-propagation pass*), then use these gradients to update the parameterss as shown above. Ideally, we find this algorithm finds the weights and biases θ that minimizes the binary cross-entropy loss function.

Practically speaking, computing $\mathcal{L}'(\theta_t) = \frac{1}{N} \sum_{i=1}^{N} L'(y_i, \mathbf{x}_i; \theta_t)$ would be far too time consuming with massive traning set and large number of parameters. Instead, we use a

randomly selected mini-batch of size m to replace $\mathcal{L}(\theta_t)$ with its estimate:

$$\hat{L}(\theta_t) = \frac{1}{m} \sum_{j=1}^{m} L(y_j, \mathbf{x}_j; \theta_t)$$

when updating the parameters. This replacement is known as *Stochastic-Gradient Descent*. This reduces the computation cost at each iteration, and will ideally lead to similar results (assuming a true randomly sampled batch and a large enough m). One *Epoch* occurs after the entire training set has been passed though the network exactly once. Therefore, with stochastic gradient descent, a single epoch occurs after about $t = \frac{N}{m}$ updates to the network paramters.

At each iteration, the stochastic gradient descent algorithm described above will always update the parameters in the direction of the steepest darnward direction. This direction may not always be the ideal choice. Depending on the level of noise and the loss function itself, it may be valuable to consider characteristics such as the momentum of the movement and the unevenness of the individual parameter components. The *Adam* optimizer modifies the stochastic gradient descent algoithm by considering the direction of the momentum and includes an adaptive mechanism to reduce the effects of extremely uneven gradient components. At the tth iteration, momentum is characterized by:

$$\nu_t = \frac{1}{(1-\gamma)} \left[\gamma \nu_{t-1} + (1-\gamma)\hat{L}(\theta_t) \right]$$

and the mechanism for adjusting the uneven components is:

$$G_t = \frac{1}{(1-\beta)} \left[\beta G_{t-1} + (1-\beta)\hat{L}(\theta_t) \odot \hat{L}(\theta_t) \right]$$

where $\gamma, \beta \in \mathbb{R}$ are tunable hyperparameters. However, they are usually set between $\gamma, \beta \in (0.9, 1)$.

Consequently, the network parameters are now updated such that:

$$\theta_{t+1} = \theta_t - \eta_t \frac{\nu_t}{\sqrt{G_t + \varepsilon}}$$

Here $\varepsilon > 0$ is a very small constant to avoid division by 0. Note that the generall structure of the parameter update still remains similar to stochasic gradient descent, however, each

15

update also considers the momentum from the previous updates and the variation in the gradient components.

Our melanoma detection networks were trained using this Adam optimizer with γ and β fixed at 0.9 and 0.999, respectively. Each mini-batch consisted of $m = 20$ training examples. Throughout training, the initial learning rate η_t was tuned using our validation set, however, we customized our networks to tune different learning rates for different branches of our network. That is, the weights and biases from the convolutional network in our ensemble had a *seperately tuned* learning rate from the rest of our ensemble network. This is discussed with more detail in Section 3.3.8. Additionally, all learning rates were scheduled to decay by a multiplicative factor of γ_{LR} after each epoch.

3.3.3   Preventing Overfitting

Given the large number of parameters, neural networks will often overfit the training data if precations are not taken. Overfitting occurs when the network effectively fits the random noise within the data, and lacks the ability to generalize on unseen data. Figure 3.2 shows the general trend of the training vs validation loss as we train our network. The idea is that if we train our model too long, it will overfit the training data, and the predictive performance will be suboptimal for the validation (or test) dataset.

To prevent our networks from overfitting the training data, we used the validation set to obtain performance updates throughout training, and stoped the training once the validation performance appears to decline. The goal would be to use the parameters that correspond to the minimum validation loss, or maximum validation performance. The vertical dotted line in Figure 3.2 indictes this ideal stopping point.

In our case, we generated a validation performance update after each half-epoch of training. At each half-epoch, we calculated the ROC-AUC score and the binary cross-entropy loss for the valdiation data, and discontinued the training if these scores dropped or stayed the same in back-to-back performance updates.

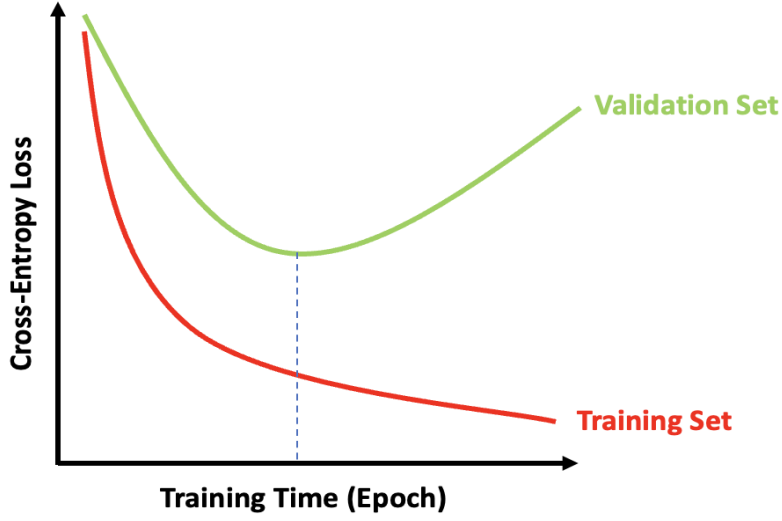There are other methods to help prevent overfitting from occuring. As mentioned above,

Figure 3.2: Overfitting Illustration

image augmentation helps prevent the chances of overfitting by artifically increasing size of the training set. We also used *L2-Regularization* for our network parameters. This form of regularization within neural networks is also referred to as *weight-decay*. In general, L2-regularization adds a penalty term, $J(\theta)$, to our loss function such that we want to minimize $\mathcal{L}(\theta) + \lambda J(\theta)$ where:

$$J(\theta) = \sum_{l,j,k} W_{j,k}^{(l)^2} + \sum_{l,k} b_k^{(l)^2}$$

and $\lambda > 0$ is a hyperparameter that we tune while training [HTF01].

In addition to regularization, we also included *dropout* layers to prevent overfitting. Dropout layers refer to the dropping of a random selection of nodes within a specified layer of the network. All of the connections associated with a dropped node are consequently removed, and the "new" network is made up of a subset of original nodes and connections. Dropout can be thought of as another form of regularization because it effectively prevents neurons in a layer from heavily depending on one input. Preventing the overreliance of a single input will reduce the variance of the network parameters.

Dropout regularization was frequently used throughout our network ensemble with $p = 0.2$. Meaning that a random 20% of the layer's corresponding neurons were masked while training.

17

3.3.4 Batch Normalization

When training the neural network with back-propagation, the distribution of each layer's inputs, $\mathbf{h}^{(l)}$ for $l = 1, 2, ..., L$, may often change because the parameters of previous layers $(< l)$ are also changing with every mini-batch update. This phenomenon is known as *internal covariate shift*, and it has been shown to drastically slow training time down [IS15]. In our networks, we stabilize the distributions of each layer's inputs, $\mathbf{h}^{(l)}$ for $l = 1, 2, ..., L$, using *batch normalization* [IS15]. Therefore, between $\mathbf{h}^{(l)}$ and $\mathbf{h}^{(l+1)}$, we added a batch normalization layer that transformed $\mathbf{h}^{(l)}$ to $\tilde{\mathbf{h}}^{(l)}$ to help stabilize its distribution and reduce the internal covariate shift. Then $\tilde{\mathbf{h}}^{(l)}$ is fed into the next layer for the computation of $\mathbf{h}^{(l+1)}$.

More formally, consider a mini-batch size of m, and a layer input $\mathbf{h}^{(l)} = (h_1^{(l)}, ..., h_d^{(l)}) \in \mathbb{R}^d$. Batch-normalization is applied to each element of $\mathbf{h}^{(l)}$ independently, so for each $k \in \{1, 2, ..., d\}$, we have exactly m values of this activation from the mini-batch. For simplicity, we denote the ith value from the kth activation by $x_{i,k}$ for $i \in \{1, 2, ..., m\}$ and $k \in \{1, 2, ..., d\}$. The batch normalization layer computes

$$\mu_k = \sum_{i=1}^{m} x_{i,k}, \quad \sigma_k = \frac{1}{m} \sum_{i=1}^{m} (x_{i,k} - \mu_k)^2, \quad \hat{x}_{i,k} = \frac{x_{i,k} - \mu_k}{\sigma_k}, \quad y_{i,k} = \beta + \gamma \hat{x}_{i,k}$$

where β and γ are learned while training. Here, $y_{i,k}$ represents the ith value (from the mini-batch) for the kth activation of the batch-normalized, layer inputs $\tilde{\mathbf{h}}^{(l)}$, where $i \in \{1, 2, ..., m\}$ and $k \in \{1, 2, ..., d\}$.

These batch-normalization transformations were used after most of the convolutional and linear layers within our ensemble of networks. Although no tuning was necessary for this technique, we found that the inclusion of batch-normalization layers significantly sped up training time.

3.3.5 Convolutional Neural Networks

The convolutional neural network (CNN) structure is often used for computer vision objectives. In general, a CNN is made up by a series of convolutional layers that filter down the training images to a single "thought" vector. This "thought" vector is a larger linear layer

with many nodes/elements that represents extracted features from the images. The final "thought" vector is then followed by a series of fully-connected linear layers as if it were a standard multi-layer perceptron. We can think of the "thought" vector as a characterization of its associated image. Figure 3.3 illustrates the general idea. Through a variety of transformations and subsampling (or pooling) techniques, the original image is filtered down into a longer, linear, layer that is ultimately used as input for another feed-forward network.
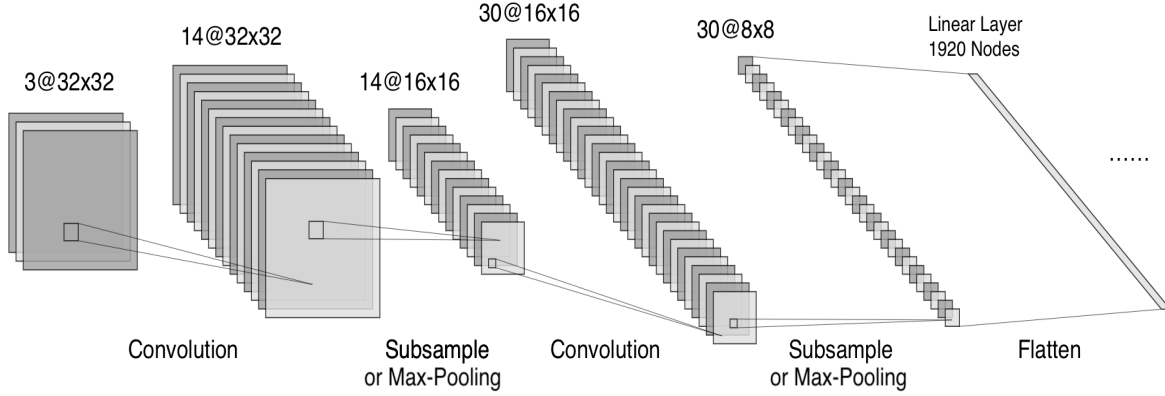

Figure 3.3: Convolution Neural Network Basic Structure

Each convolutional layer $\mathbf{h}^{(l)}$ is organized as a series of channels, where each channel is obtained by a kernel, $\mathbf{W}_{i,j}$ operating on the previous layer's $\mathbf{h}^{(l-1)}$ which was also organized as a series of channels. Each kernel operation is a local weighted sum of the values within each channel. That is, when there are multiple channels, this weighted sum is over the multiple channels. Once the weighted summations have been performed, a bias term is generally added, and a non-linear transformation, such as the ReLU activation function, is applied to the result. For example, consider Figure 3.3. The first layer, $\mathbf{h}^{(0)}$ is the input image with 3 channels (which coresspond to Red, Green, and Blue), and after applying the kernel operations with the bias and ReLU transformation, the subsequent layer, $\mathbf{h}^{(1)}$, increases to 14 channels.

We may also present the operations of a convolutional layer more formally. Let $\mathbf{x}_{ij} \in \mathbb{R}^{C_{l-1}}$ represent the vector of values at the (i,j)th coordinate for each of the C_{l-1} channels in the $(l-1)$th layer, and let $\mathbf{y}_{ij} \in \mathbb{R}^{C_l}$ represent the vector of values at the (i,j)th coordinate

for each of the C_l channels in the lth layer. The kernel operation is equivalent to the following linear transformation:

$$\tilde{\mathbf{x}}_{ij} = \sum_{\Delta i=-k_1}^{k_1} \sum_{\Delta j=-k_2}^{k_2} \mathbf{W}_{\Delta i\ \Delta j}\ \mathbf{x}_{i+\Delta i\ j+\Delta j}$$

where $\mathbf{W}_{\Delta i \Delta j} \in \mathbb{R}^{C^l \times C^{l-1}}$ is learned and the kernel dimensions are fixed at $2k_1$ by $2k_2$. As mentioned above, the bias term, $\mathbf{b}_l$, is added and the ReLU transformation is applied such that:

$$\mathbf{y}_{ij} = \text{ReLU}(\tilde{\mathbf{x}}_{ij} + \mathbf{b}_l)$$

where $\mathbf{b}_l$ is learned and the ReLU transformation is applied element-wise.

For each convolutional layer, a method of *pooling* or *sub-sampling* is generally done after the above transformations are performed. Pooling and sub-sampling methods are performed to effectively downsample the output of the convolutional layer along its spatial dimensions. That is, the size of each channel is reduced. While the convolutional layer of $h^{(l-1)}$ is generally meant to increase the number of channels, the pooling or sub-sampling method is applied to decrease the height and width of each channel. The primary goal of pooling and sub-sampling is to reduce the total number of parameters within the network. Consequently, this reduces the chance of overfitting our training data and significantly increases training time.

3.3.6   Ensemble #1

As mentioned previously, we trained a total of two multi-network ensembles. The first ensemble includes a standard CNN for the image data and a multi-layer perceptron for the patient-level metadata, while the second ensemble includes a ResNeSt network for the image data and a similar multi-layer perceptron for the patient-level metadata. The first ensemble was used as a model to compare the ResNeSt model with. It can be thought of as a baseline model to assess the improvements provided by the split-attention networks within the second ensemble.

The CNN within the first ensemble contained a total of 6 convolutional layers. Each convolutional layer was immediately followed by batch-normalization and downsampling. The convolutional layer within this ensembles exclusively applied *max-pooling* as its downsam-

pling method. For each channel, max-pooling simply replaces the value of each pixel by the maximum of a fixed-size patch surrounding the pixel. In this case, we decided to use max-pooling with a kernel size of 2×2 and a stride of 2. Figure 3.4 demonstrates a simple example of how max-pooling is applied to the spatial data within each channel.
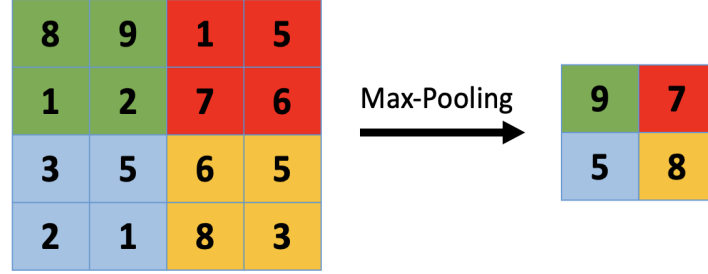

Figure 3.4: Max-Pooling Example with Filter-Size 2×2 and Stride 2

Layer Name	Output Size	Operation	Output Channel Number	Normalization	Activation	Pooling
Conv1	253×253	7×7	6	Batch-Norm (2D)	ReLU	Max-Pool (2x2, stride=2)
Conv2	124×124	7×7	15	Batch-Norm (2D)	ReLU	Max-Pool (2x2, stride=2)
Conv3	59×59	7×7	30	Batch-Norm (2D)	ReLU	Max-Pool (2x2, stride=2)
Conv4	26×26	5×5	60	Batch-Norm (2D)	ReLU	Max-Pool (2x2, stride=2)
Conv5	11×11	5×5	80	Batch-Norm (2D)	ReLU	Max-Pool (2x2, stride=2)
Conv6	3×3	3×3	100	Batch-Norm (2D)	ReLU	Max-Pool (2x2, stride=2)
Linear1	1×1	Fully Connected	512	Batch-Norm (1D)	ReLU	N/A
Linear2	1×1	Fully Connected	512	Batch-Norm (1D)	ReLU	N/A

Table 3.1: CNN Architecture for Multi-Network Ensemble #1

After all 6 convolutional, normalization, and downsampling operations are applied, the original 3-channel image is filtered to a linear layer with exactly 900 nodes. As described

21

above, this output should respresent the high-level features within the images. We include two additional fully-connected layers, each with 512 nodes, in another attempt to learn any remaining non-linear combinations of these features. Note that all convolutional and linear layers associated with this network used the ReLU activation function. Table 3.1 includes the details regarding the specific architecture used for the CNN within our first ensemble.

In parallel to this convolutional network, the multi-layer perceptron takes the 7 patient-level features (after one-hot encoding) and connects it to a linear layer with 256 nodes. A batch-normalization layer is also included, and dropout regularization with $p = 0.2$ is used to prevent overfitting. Again, we note that the ReLU activation function was used for this multi-layer perceptron.
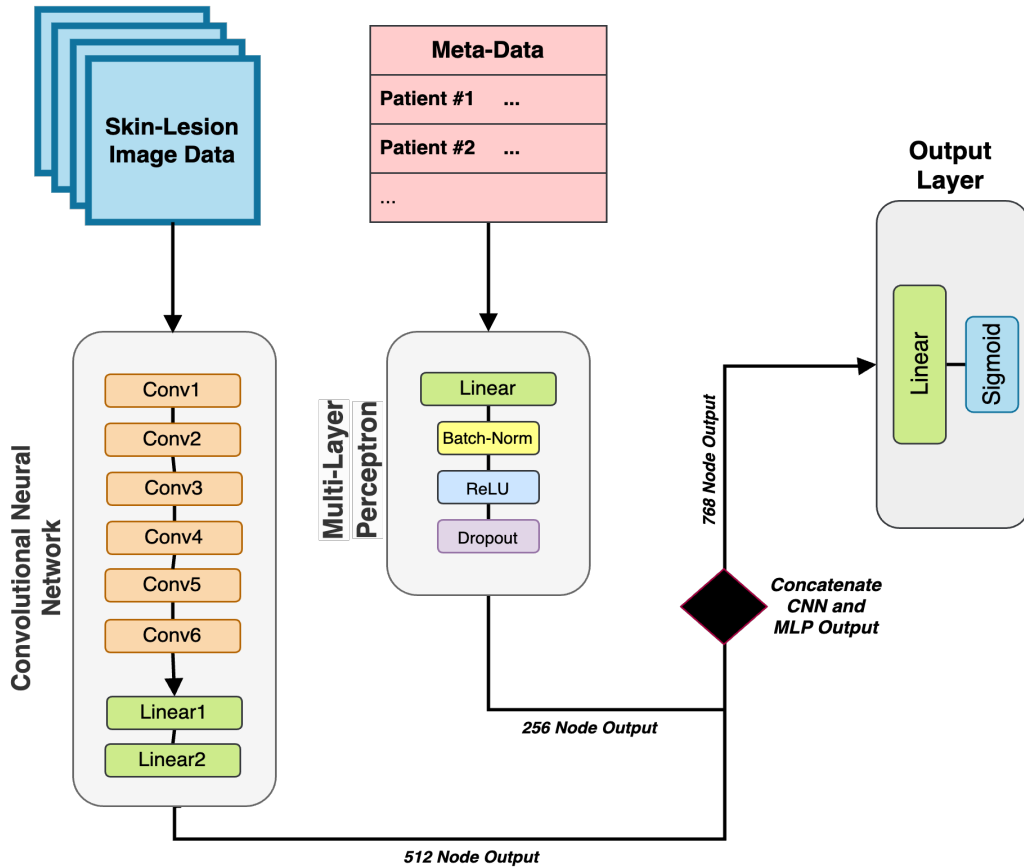

Figure 3.5: Multi-Network Ensemble #1

Figure 3.5 illustrates how these two seperate networks are effectively fused together for our

first ensemble. The 512 node output corresponding to the image-data CNN is concatenated with the 256 node output corresponding to the patient-level metadata. This concatenated layer is then fed into the output layer, with a single node. Lastly, we applied the sigmoid function to the score of the output node in order to estimate the probability that the associated skin-lesion is malignant. Throughout training, we searched through a variety of potential learning rates between $\eta \in (10^{-3}, 10^{-1})$ to find the optimal initial learning rate, $\hat{\eta}$.

The framework of this first multi-network ensemble is very similar to the framework used in the second ensemble. In the next two sections, we introduce the ResNeSt network architecture and how it may be leveraged to better extract the high-level features within the skin-lesion images. Then, we provide the details regarding how it was used within the second multi-network ensemble.

3.3.7   ResNeSt

In 2015, the residual neural network, or *ResNet*, revolutioned the world of convolutional neural networks [HZR15]. The ResNet model was designed by Kaiming He, and introduced in the paper: *Deep Residual Learning for Image Recognition*. Prior to the residual network, deep neural networks faced the "vanishing gradient" difficultly while training. The "vanishing gradient" occurs when the gradients get very close to 0. In general, as the number of layers within a network increases, the gradients approach 0. Motivated by Taylor expansion, Kaiming introduced the technique known as a "identity skip connection". An identity skip connection reduces the effect of gradient vanishing regardless of the number of layers within the network, allowing for deeper networks to be built without negatively impacting training.

In general, a residual network uses blocks of convolutional layers, in combination with an identity skip connection (more formally known as *identity mapping*). Each block of convolutional layers is referred to as a *residual block*. Suppose $\mathbf{h}^{(l)}$ is the input to a residual block and $\mathbf{s}^{(l+1)}$ is the output of the residual block. If we denote the series of applied convolutional layers (weighted sums), batch normalizations, and ReLU activations by $\mathcal{F}$, then $\mathbf{s}^{(l+1)} = \mathbf{h}^{(l)} + \mathcal{F}(\mathbf{h}^{(l)})$. Figure 3.6 illustrates this idea of a residual block with an

identity skip connection. Note that the final ReLU activation is not applied until after the addition of the residual block transformations and the identity connection such that $\mathbf{h}^{(l+1)}$ can be found by applying the ReLU activation to $\mathbf{s}^{(l+1)}$. Additionally, if the dimensions of $\mathbf{h}^{(l)}$ and $\mathcal{F}(\mathbf{h}^{(l)})$ are not the same, a linear projection is used by the shortcut connections to match the dimensions.
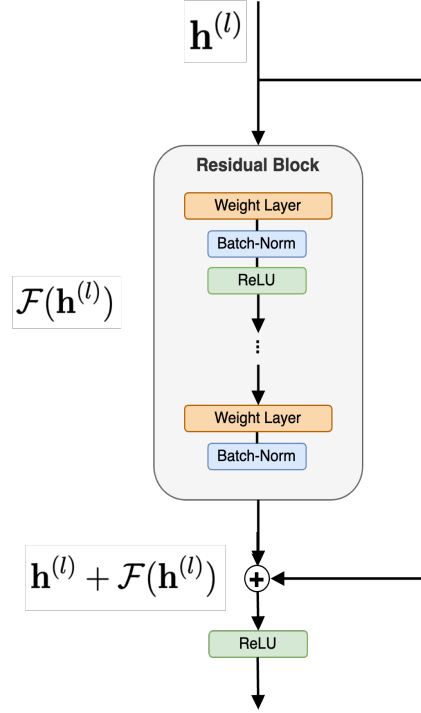

Figure 3.6: Residual Block Example

Many variants of the ResNet have emerged since its initial release in 2015. In 2020, Amazon introduced the *ResNeSt* model in the paper, *ResNeSt: Split-Attention Networks*. This model was built upon the ResNet meta-architecture, and many ResNeSt models contain the same number of layers as the original ResNet models. However, the network structure was also scaled up to provide even deeper versions, with significantly larger input crop-sizes. For instance, the ResNeSt-269 model has an input crop-size of 416×416 and has a total of 269 layers. This was particularly attractive for our application to melanoma detection since many of the original images were very large, and we wanted to limit further cropping to minimize further information loss within our training images.

24

The ResNeSt adopted both the block-style architecture and the identity-skip, or *shortcut*, connection from the ResNet model. As before, this mitigated potential vanishing gradient issues. Table 3.2 shows the basic structures of the ResNet-101 and the ResNeSt-101 networks. Since it is not shown in the table, we should note that a 2×2 average-pooling layer (stride of 2) is also applied to the identity connections in the ResNeSt's transitioning blocks. Average-pooling works very similar to max-pooling in that it simply replaces the value of each pixel by the average of the fixed-size patch surrounding the pixel, instead of the maximum.

Layer Name	Output Size	ResNet101	ResNeSt101
Conv1_x	112×112	7×7, 64, stride 2	$[3 \times 3]$, 64, stride 2
		3×3 max pool, stride 2	
Conv2_x	56×56	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	`[ResNeSt Block, 256]` $\times 3$
Conv3_x	28×28	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	`[ResNeSt Block, 512]` $\times 4$
Conv4_x	14×14	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	`[ResNeSt Block, 1024]` $\times 23$
Conv5_x	7×7	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	`[ResNeSt Block, 2048]` $\times 3$
Linear	1×1	3×3 average pool, 1000 (fully connected)	

Table 3.2: ResNet-101 and ResNeSt-101 Architecture

Clearly, the major difference between the ResNet and ResNeSt models lies within the design of the computational blocks. While the ResNet block contains a few convolutional layers, batch-normalizations, and ReLU activations, the ResNeSt block is much more involved. Each ResNeSt block has *Split-Attention* sub-blocks that enable channel-wise attention across

different subsets of channels.

We now describe the basic design the ResNeSt blocks, as described in the paper, *ResNeSt: Split-Attention Networks* [ZWZ20]. Suppose the incoming input to a given block has shape (H, W, C) where H and W are the spatial dimensions and C is the number of the channels. At the beginning of each ResNeSt block, the C channels are divided into K groups. Here, K is a tunable hyperparameter referred to as the cardinality, and each group is known as a cadinal group. Figure 3.7 illustrates this split to cardinal groups.
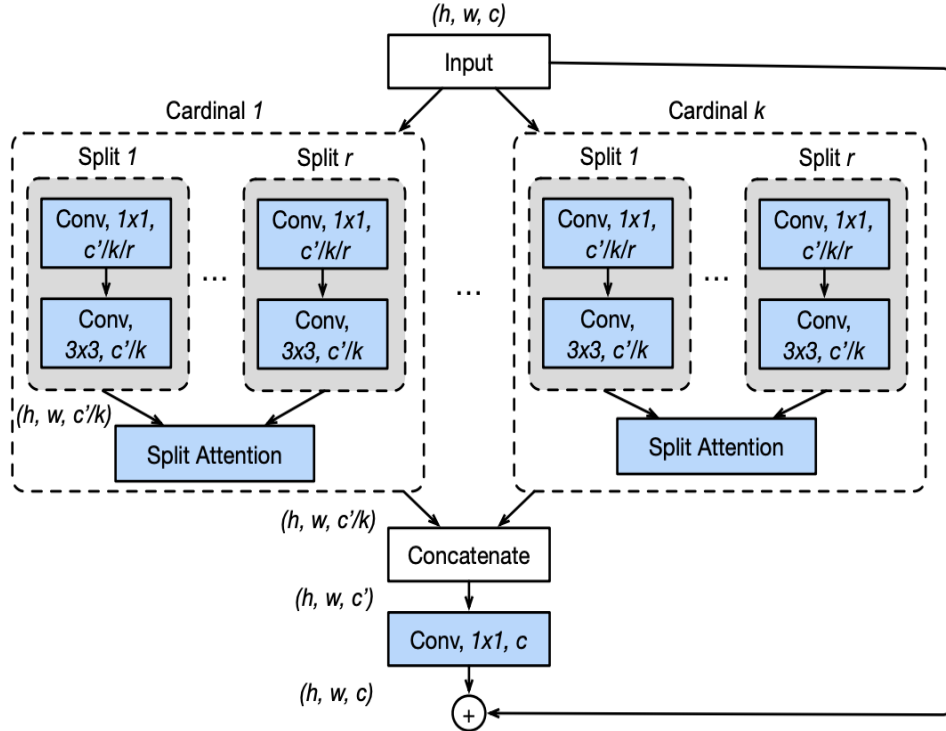

Figure 3.7: ResNeSt Block

As expected, the block has the shortcut connection on the right-hand side and creates K cardinal groups from the C input channels. With each caridnal group, the set of C/K channels is further split into R groups. Therefore, the total number of feature groups is $G = KR$. Two convolutional layers (1×1, 3×3) are then applied to each of the R groups within a a caridnal group in parallel. The R outputs within each cardinal group are then used together as input for the *Split-Attention* block as shown in Figure 3.8.

Let U_i represent the output from the convolutional layers for the ith feature group, where
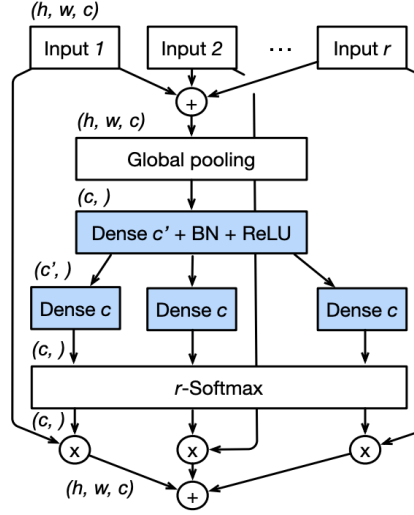
Figure 3.8: Split-Attention Block

$i \in \{1, 2, ..., G\}$. Using its R outputs, the split-attention block first calculates the combined representation for the kth cardinal group:

$$\hat{U}^k = \sum_{j=R(k-1)}^{Rk} U_j$$

where $\hat{U}^k \in \mathbb{R}^{H \times W \times C/K}$ and $k \in \{1, 2, ..., K\}$. This is step is indicated by fusing together of the R inputs at the very top of the Figure 3.8. Furthermore, each cardinal groups's associated channel-wise statistics are then gathered using the global average pooling across the spatial dimensions and represented by the $s^k \in \mathbb{R}^{C/K}$. The cth component of the global pooling vector is calculated as follows:

$$s_c^k = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \hat{U}_c^k(i, j)$$

The global pooling vector is indicated by the "Global Pooling" step in Figure 3.8.

As illustrated, the gobal pooling vector is ultimately used to compute the weighted fusion of the kth cardinal group representation, $V^k \in \mathbb{R}^{H \times W \times C/K}$. V^k is computed from channel-wise soft attention such that each channel is a weighted sum over the corresponding R splits. Formally, the cth channel is calculated as

$$V_c^k = \sum_{i=1}^{R} a_i^k(c) U_{R(k-1)+i}$$

27

where $a_i^k(c)$ represents the soft assignment weight such that:

$$a_i^k(c) = \begin{cases} \frac{exp(\mathcal{G}_i^c(s^k))}{\sum_{j=1}^{R} exp(\mathcal{G}_j^c(s^k))}, & \text{if } R > 1 \\ \frac{1}{1+exp(-\mathcal{G}_i^c(s^k))}, & \text{if} R = 1 \end{cases}$$

Here, $\mathcal{G}_i^c$ denotes a mapping from the global context respresentation, s_k, to an associated weight for the i feature group and cth channel.

Lastly, the cardinal group representations V^k for $k = 1, 2, ..., K$ are concatenated along their channel dimensions and a 1×1 convolutional layer is applied upon return to the ResNeSt block (this is generally performed to restore dimension if reduced at the beginning of the block [HZR15]). As illustrated in Figure 3.7, the shortcut connection X is then added to the concatenated output from the Split-Attention blocks V. The sum, $V + X$, serves as the final output of the ResNeSt block.

We should note that the techniques employed within split-attention block appear to draw heavily on the muliple-head attention and self-attention mechanisms from the 2017 paper, *Attention Is All You Need* [VSP17]. Although the transformer model is very different, many of the foundational ideas seem to overlap.

3.3.8   Ensemble #2

The ResNeSt model was shown to perform better than the EfficientNet in the accuracy and latency trade-off for image classification problems. Additionally, the ResNeSt model has demonstrated excellent transfer learning results on a variety of well-known public benchmarks. We now discuss the steps taken to leveraged this high performing network to pick up on the most important features within our skin-lesion images.

As mentioned in the previous section, our second ensemble uses the ResNeSt network to extract the principal features from our image data and uses a standard multi-layer perceptron to find patterns within the patient-level metadata. We used the deepest avaiable network, the *ResNeSt-269*, which uses an input crop-size of 416×416 and has a total of 269 layers. This input size is significantly larger than the maximum crop-size for widely used networks such as those from the ResNet or Inception-Net families. We used this larger network to

limit any further cropping of our skin lesion images. Since many of the images were originally very large, further cropping may lead to unnecessary information loss. In addition, a deeper networks generally have a stronger ability to learn features at multiple levels of abstraction which may lead to better a generalization. As described above, the ResNeSt-269 model stacks several computational blocks containing split-attention operations and shortcut connections, with a variety of intermediate pooling (max/average) operations throughout the network. Note that batch-normalization was also applied after each block.

After feeding our 3-channel skin-lesion image through the ReSneSt-269 model, we obtain a linear layer with 1000 nodes as output. This output represents the extracted features found from the image. Ideally, the channel-wise attention mechanisms were successful in capturing the cross-feature interactions that may have not been found from the standard convolutional network within the first ensemble. Since the ResNeSt-269 already contains a deep set of layers, we chose to not connect any additional linear layers to the 1000-node output.

As in Ensemble #1, a multi-layer perceptron was also built and trained in parellel to the ResNeSt-269 model. This network also uses the 7 patient-level features as input, then sequentially applies a 256-node linear layer with batch-normalization, dropout regularization (with $p = 0.2$), and ReLU activation.

Figure 3.9 illustrates how the ResNeSt-269 output was fused together with the multi-layer perceptron output. The concatenated output effectively served as a $1000 + 256 = 1256$ node linear layer. This layer was then connected to a single output node. In order to estimate the probability that the skin-lesion is maligant, we applied the sigmoid function to the output node's final score.

One of the drawbacks of using such a deep network is that it requires substantial computing resources to train the large number of network parameters. However, we may be able to leverage pre-trained models that were trained using computing resources that are much more powerful than what is available to us. We can use the weights from a network that has *already been trained* for a large-scale, generic, image classification task. This process is known as *transfer learning*. The idea is that if a network has been trained on a large enough,
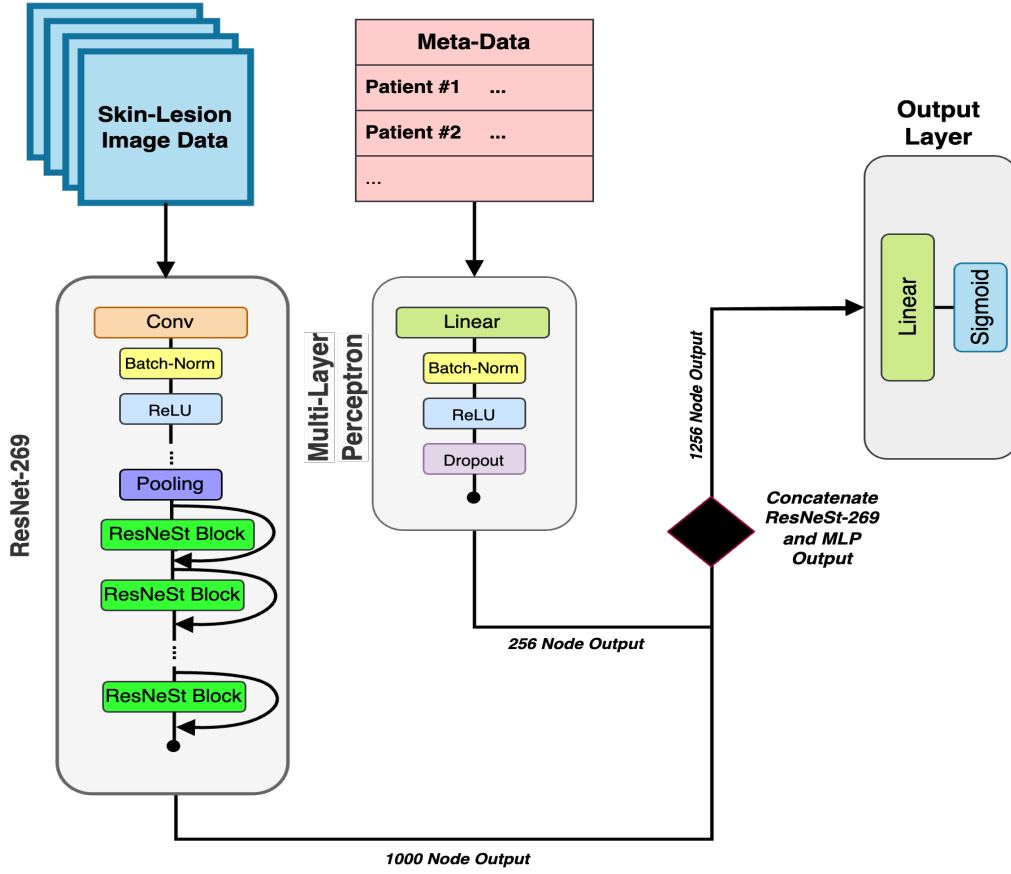
Figure 3.9: Multi-Network Ensemble #2

and general enough, dataset, we may take advantage of the learned weights by initializing our network with these pre-trained weights instead of starting from scratch with randomly initiallized weights.

In our case, we initiallized our ResNeSt-269 model with the pre-trained parameters from the ImageNet dataset. Throughout training, we *fine-tuned* these parameters to our skin-lesion images by updating the parameters with a very small learning rate throughout training. To prevent extremely large modifications of the pretrained weights within our ResNeSt network, we usually began training with very small learning rates when tuning the learning rate hyperparameter. For the ResNeSt portion of the second ensemble, we tried a variety of learning rates between $\eta_{ResNeSt} \in (10^{-6}, 10^{-5})$ in search for an optimal inditial learning rate. However, since the multi-layer perceptron and final connections to the single-node

30

output layer were randomly initiallized, we searched for an optimal learning rate within $\eta_{MLP+Final} \in (10^{-4}, 10^{-2})$, as we did in the first ensemble.

3.3.9 Evaluation Methods

To examine and compare predictive performance of each ensemble, we used our (completely unseen) test data. For each model, we generated predicted probabilities for the test set to obtain visuals such as ROC Curves, accuracy plots, sensitivity plots, specificity plots, and false-positive plots. In addition to the visuals, we also computed quantitative measures such as the area under the ROC curves, confusion matrices, and a variety of other performance metrics.

Although visuals and metrics such as the ROC curve and ROC-AUC consider *all* thresholds, most of the other measures mentioned above require a specified probability threshold to generate class predictions. Given that our ensembles attempt to predict the presence of melanoma, the most aggressive form of skin cancer, we aimed to identify a threshold that prioritizes true-positives while not overly accepting too many false-positives. Accepting too many false-positives would reduce the trust in our model's final predictions. In addition, we prefered to optimize our threshold in a way that does not unfairly suffer from the large class imbalance in our data. Therefore, we used the geometric mean of the *Sensitivity* (or true-positive rate, $\frac{TP}{TP+FN}$) and the *Specificity* (or true-negative rate, $\frac{TN}{TN+FP}$),

$$G_{mean}(t) = \sqrt{\text{Sensitivity} \cdot \text{Specificity}} = \sqrt{\frac{TP \cdot TN}{(TP + FN)(TN + FP)}}$$

to determine an optimal threshold for each model. That is, for each model, we found the threshold $t^* \in (0,1)$ that maximizes the geometric mean, $G_{mean}(t)$. Note that only the training and validation sets were used to find t^* so that we can retain a completely unseen testing set when evaluating and comparing model performances.

Recall that the primary metrics of interest are true-positive rate and true-negative rate as they are not influenced by the large class imbalance within our test data. As an example, consider a measure such as accuracy. Assessing performance with accuracy would not be

appropriate in our case because a model that always classifies a skin-lesion as beniegn would achieve $> 98\%$ accuracy on our test set. Obviously, this model would be useless, and the measurement would decieving. Therefore, using our confusion matrices, we compute and output the corresponding sensitivities, specificities, balanced-accuracies, and F2-scores. Balanced-accuracy simply represents the average of the true-positive and true-negative rates, while the F2-score corresponds to the weighted harmonic mean of the precision ($\frac{TP}{TP+FP}$) and sensitivity such that:

$$F2_{score} = \frac{5 \cdot \text{Precision} \cdot \text{Sensitivity}}{4 \cdot \text{Precision} + \text{Sensitivity}}$$

The F2-score represents a single measure of both precision and recall in one metric, with slightly more weight on sensitivity than on precision.

CHAPTER 4

Results

4.1   Tuning Ensemble #1

As we described in Section 3.3.6, ensemble #1 uses a standard CNN to process the skin-lesion images and a multi-layer perceptron to process the associated patient-level data. Throughout training, we tuned a variety hyperparameters such as the L2-regularization hyperparameter, λ, the initial learning-rate η with its multiplicative factor γ_{LR} used to decay the learning rate after each epoch, and the total number of training epochs, M.

We performed a grid-search to find the optimal set of hyperparameters. That is, we trained the ensemble with a distinct set of hyperparameters $S^{(k)} = \{\lambda^{(k)}, \eta^{(k)}, \gamma_{LR}^{(k)}, M^{(k)}\}$ and obtained the ROC-AUC score on the validation data. We trained and validated our ensemble with a total of 10 unique sets of hyparmeters, $S^{(k)}$ for $k = 1, 2, ..., 10$, and took the set corresponding to the maximum ROC-AUC, $S^{(\hat{k})}$ as the optimal set of parameters. Given our limited computing resources, this repeated process of training and validating was very time consuming. In fact, the process took more than a week to complete (with occasional periods of inactivity). The optimal set of hyperparameters $S^{(\hat{k})}$ contained:

- L2-regularization hyperparameter of $\hat{\lambda} = 0.01$

- Initial learning-rate $\hat{\eta} = 0.01$

- Multiplicative Decay Factor $\hat{\gamma}_{LR} = 0.5$ (η_t is cut in half after each epoch)

- Number of Epochs $\hat{M} = 10$

4.2   Tuning Ensemble #2

As we described in Section 3.3.8, ensemble #2 uses a ResNeSt-269 network to process the skin-lesion images and a multi-layer perceptron to process the associated patient-level data. We tuned a set of hyperparameters similar to Ensemble #1. However, as mentioned in the previous chapter, we tuned the learning rate associated with the ResNeSt network $\eta_{ResNeSt}$ seperately from the learning rate associated with the rest of the ensemble, $\eta_{MLP+Final}$, because the weight's in the ResNeSt network are pre-trained and we would only like to fine-tune them for our application in melanoma detection.

Again, we performed a grid-search to find the optimal set of hyperparameters $S^{(\hat{k})} = \{\hat{\lambda}, \hat{\eta}_{ResNeSt}, \hat{\eta}_{MLP+Final}, \hat{\gamma}_{LR}, M^{(k)}\}$ using the ROC-AUC scores on the validation data. We trained and validated this ensemble with a total of 15 unique sets of hyparmeters, $S^{(k)} = \{\lambda^{(k)}, \eta_{ResNeSt}^{(k)}, \eta_{MLP+Final}^{(k)}, \gamma_{LR}^{(k)}, M^{(k)}\}$ for $k = 1, 2, ..., 15$. Since the ResNeSt-269 model contains over 100 million parameters alone, the training the ensemble required even more time than the first ensemble. In total, the entire process of training required about 2 weeks, incluing occasional periods of inactivity. In this case, the optimal set of hyperparameters $S^{(\hat{k})}$ contained:

- L2-regularization hyperparameter of $\hat{\lambda} = 0$ (No L2-regularization applied)

- Initial learning-rate for ResNeSt-269 parameters $\hat{\eta}_{ResNeSt-269} = 3 \times 10^{-5}$

- Initial learning-rate for all other parameters $\hat{\eta}_{MLP+Final} = 1 \times 10^{-3}$

- Multiplicative Decay Factor $\hat{\gamma}_{LR} = 0.1$

- Number of Epochs $\hat{M} = 10$

4.3   Evaluation and Comparison

In this final results section, we use the dedicated test data to assess and compare the overall predictive performance of each multi-network ensemble. We started by examining the metrics

of interest, true-positive rate and true-negative rate, computed at all possible thresholds. Figure 4.1 displays the specificity and specificity scores across the range of thresholds, $t \in (0, 1)$.
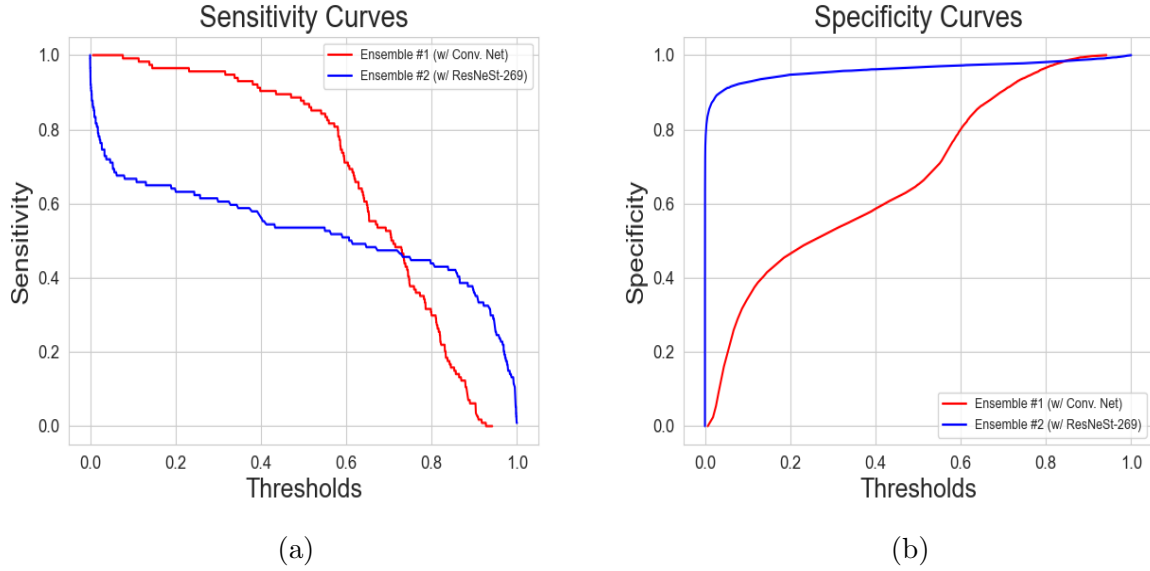

Figure 4.1: True-Positive Rates and True-Negative Rates for Ensembles #1 & #2

As mentioned in the Section 3.3.9, since our ensembles are ultimately predicting the presence of a life-threatening condition, a false-negative is far more dangerous than a false-positive. Ideally, we want our ensembles to have the greatest possible true-positive rate without compromising the overall "trust" in its classifications. That is, our goal was to maximize the sensitivity without dropping the specificity too much. Figure 4.1 indicates that there may exist a distinct range of thresholds for each model that could accomplish this goal. From Figure 4.1(a) and Figure 4.1(b) we can see that the optimal threshold lies within $t \in (0.45, 0.75)$ for Ensemble #1 since their appears to be the best balance of sensitivity and specificity in this range. Notice that for Ensemble #2, Figure 4.1(a) indicates a sharp and sudden drop in sensitivity within the range of smaller thresholds. However, Figure 4.1(b) shows an equally sharp and sudden *increase* in specificity within the same range of smaller thresholds. Therefore, we can see that the optimal threshold lies within $t \in (0, 0.05)$ for Ensemble #2.

The geometric mean of the sensitivity and the specificity was then used to find the optimal threshold, t^*, for each model. For both models, we evaluated the $G_{mean}(t)$ at a variety of different thresholds between $t \in (0, 1)$, and stored the threshold that maximized the $G_{mean}(t)$. Using the optimal thresholds for each model, we created the following performance metrics and confusion matrices from our test set.

Model	t^*	$G_{\mathbf{mean}}$	Sensitivity	Specificity	Balanced Accuracy	$\mathbf{F2}_{\mathbf{Score}}$
Ensemble #1 (w/ Conv. Net)	0.580	0.781	0.798	0.765	0.782	0.105
Ensemble #2 (w/ ResNeSt-269)	0.003	0.853	0.895	0.813	0.853	0.287

Table 4.1: Performance Metrics Using the $G_{mean}(t)$ Maximizer, t^*

Predicted

		Beniegn	Malignant	Total
Actual	**Beniegn**	4981	1531	6512
	Malignant	23	91	114
	Total	5004	1622	

(a) Ensemble #1 (w/ Conv. Net)

Predicted

		Beniegn	Malignant	Total
Actual	**Beniegn**	5293	1219	6512
	Malignant	12	102	114
	Total	5305	1321	

(b) Ensemble #2 (w/ ResNeSt-269)

Table 4.2: Confusion Matrices For Both Multi-Network Ensembles

From the performance metrics and confusion matrices, we can see that both model's demonstrate an excellent classification performance. However, the improvement from the ResNeSt-269 in Ensemble #2 is very apparent. Using the deep split-attention network to process the skin-lesion images rather than a generic convolutional neural network has shown to drastically improve every metric listed in Table 4.1. Most notably, we can see that the balanced accuracy increases by over 7% when using Ensemble #2! This indicates that the Ensemble #2 model was able to successfully leverage ResNeSt-269 architecture to extract the most important features from the skin-leasion images. Table 4.1 tells us that the balanced

accuracy increases

For a more general assessment and comparison across all possible thresholds, we also used the testing data to compute the ROC curves for each model (shown below) and their associated ROC-AUCs.

Model	ROC AUC
Ensemble #1 (w/ Conv. Net)	0.852
Ensemble #2 (w/ ResNeSt-269)	0.929
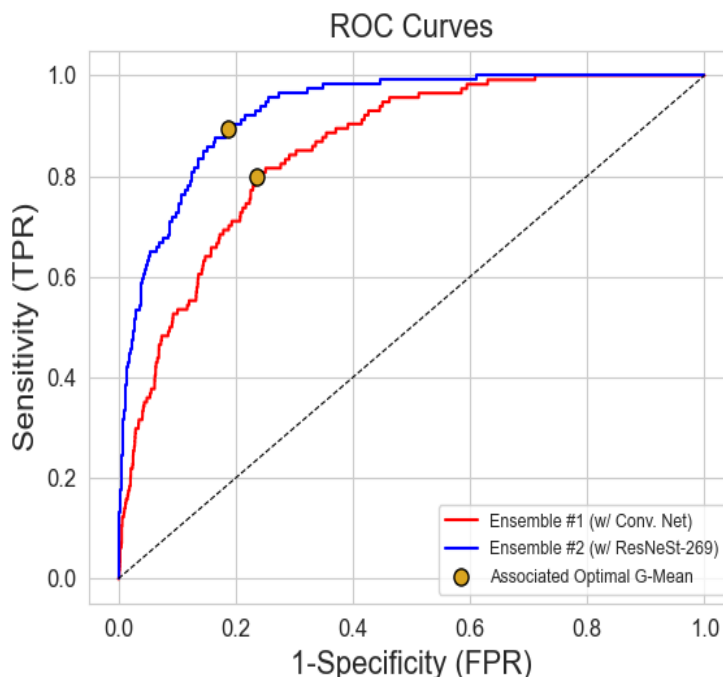
(a)   (b)

Figure 4.2: ROC-AUC and ROC Curves for Ensemble's #1 & #2

Although it is difficult to compare the models visually from Figure 4.2(b), it's clear from both the plot and the associated table of ROC-AUC scores that both ensembles demonstrated outstanding predictive performance across the entire range of thresholds. They each exhibit an ability to successfully classify whether or not the skin-lesion is beniegn or malignant, and they appear to produce predictive results significantly better than a model that purely guesses (which is displayed by the black dotted line).

The associated ROC-AUCs from Figure 4.2(a) above also tell us that the ensemble using the ResNeSt-269 network to extract features from the skin-lesion images was able to detect the presence of melanoma better than the ensemble using the generic CNN, across the whole

raneg of thresholds. That is, at each threshold $t \in (0, 1)$, the aggregate score of sensitivity and specificity was the same or better for Ensemble #2 than it was for Ensemble #1. Using the split-attention network increased the total area under the ROC curve by over 9%. This provides further evidence suggesting that the second ensemble model was able to successfully leverage the channel-wise attention design within the ResNeSt network to extract the most important features and most relevant cross-feature interactions from the skin-leasion images. Table 4.1 tells us that the balanced accuracy increases

CHAPTER 5

Conclusion and Future Work

Start with idea of retraining on the entire training + Validation sets.

Write here. Maybe discuss the work that has been done in the past [VPZ19], and maybe point out the efficient net results from here [LQW21] (note that others have been able to leverage efficient net model to get even better results, but our ROC-AUC is comparable amongst the best available options)

REFERENCES

[AI23]   Albumentations AI. "Transforms.", 2023. Last Accessed: 2023-03-20.

[Fou23]   Skin Cancer Foundation. "Melanoma Warning Signs.", 2023. Last Accessed: 2023-03-20.

[HTF01]   Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[HZR15]   Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition.", 2015.

[Ins23]   National Cancer Institute. "Cancer Stat Facts: Melanoma of the Skin.", 2023. Last Accessed: 2023-03-20.

[IS15]   Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *CoRR*, **abs/1502.03167**, 2015.

[ISI20]   "The ISIC 2020 Challenge Dataset.", 2020. Last Accessed: 2023-02-13.

[LQW21]   Chaoyi Li, Zihan Qiao, Kehan Wang, and Jiang Hongxing. "Improved EfficientNet-B4 for Melanoma Detection." In *2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, pp. 127–130, 2021.

[Soc23]   American Cancer Society. "Key Statistics for Melanoma Skin Cancer.", 2023. Last Accessed: 2023-03-20.

[VPZ19]   Eugenio Vocaturo, Diego Perna, and Ester Zumpano. "Machine Learning Techniques for Automated Melanoma Detection." In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 2310–2317, 2019.

[VSP17]   Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need.", 2017.

[ZWZ20]   Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander Smola. "ResNeSt: Split-Attention Networks.", 2020.