Cost and Risk aware Skin lesion Classification using Bayesian Inference

Written by Daniel Blackley
Supervised by Professor Stephen Mckenna
Co-Supervised by Mr Jacob Carse

A thesis presented for the degree of
Bachelor of Computer Science

School of Science and Engineering
University of Dundee
United Kingdom
April 2021

Contents

Abstract

Skin lesions are areas of the skin that have abnormal growths and can be an indication of skin cancer, thankfully, these can usually be identified with a visual examination. Manual examination is a rather tedious and time consuming job for expert dermatologists, recent advances in deep learning has helped automate the classification of skin lesions while showing remarkable accuracy. These tools do not necessarily take into account the cost of miss-classifying a life threatening lesion with a benign one however.

We employed three machine learning algorithms: A standard softmax baseline, Monte Carlo Dropout and Bayes by Backprop in a cost-aware environment and compared the results. We found that when compared solely with accuracy MC Dropout performs best, but only slightly better than a softmax response. When we employ our model in a cost aware setting Bayes by Backprop performs best.

1 Introduction

Computer vision is the field relating to analysis and understanding of digital images, tasks can include things such as segmentation[9][1] of images or object detection[22]. Deep Learning has provided revolutionary advances in the field of computer vision and has proven to be better than many of the previous algorithms designed to handle vision-related tasks[15][1], one area of which Deep Learning is useful is in the medical imaging domain, particularly, the classification of different skin lesions[1][3][6][10][15].

Deep Learning has shown to give comparable results to expert dermatologists [10] on the task of skin lesion classification, however, it is worth noting that most discussion around the use of deep learning in the field of medical imaging is not that deep Learning models need to outperform human experts, only compliment them. The most likely outcome for deep learning is to make human-machine hybrid teams[2][3]. Given this new perspective it is important to consider not necessarily the accuracy of these models, but instead their ability to perform as probabilistic models that give well tuned distributions and some level of uncertainty regarding their classification decision. Part of having these two teams work together means that we need to consider that not all skin lesions are equal, malignant Melanoma being particularly deadly due to its ability to spread to other parts of the body[24]. If we want our model to work well in human-machine teams, we must also ensure that our model is capable of not miss-classifying a deadly skin lesion as a non-deadly skin lesion, i.e. it is important that our model to be conscious of the cost of miss-classifying[4].

Recently, the International Skin Imaging Collaboration (ISIC) Proposed a challenge called ISIC 2019[8][7][29], the challenge was to classify dermoscopic images amongst 9 different skin lesion categories, one being an unknown class. This challenge posed as one of the primary inspirations for this project, to research different methods of getting uncertainty estimations and well calibrated predictive distributions from deep learning models.

The gold standard for uncertainty estimations in deep learning is Deep Ensembles[21], this method requires the training of many networks and combines them into a single, more powerful, network, this has the downside of the being very computationally expensive to train and generate predictions with. Another method would be to use Bayesian Neural Networks, Bayesian Neural Networks put distribution of values over the weights of a network and samples multiple times from this distribution, essentially making an infinite ensemble of networks[5]. In this project, we seek to give a comparison of 3 different methods, Monte Carlo Dropout[11] (MC Dropout), a standard Softmax Response[16] (SR) and Bayes by Backprop[5] (BbB) in regards to uncertainty estimation and the cost of miss-classification.

1.1 Project aims

A formal specification of our project is somewhat inappropriate, but we can list the following aims:

1. Train a competent network that can perform adequately on the task of skin lesion classification.
2. Generate some form of uncertainty estimation for our proposed methods.
3. Produce results showing each of our methods while taking into account cost of misclassification.
4. Formulate evaluation metrics to emphasise the strengths weakness of our proposed methods, providing insight as of to which method may perform better in different situations.

We specify requirement 1 further as the definition of a competent model is somewhat vague. ISIC 2019 has a live leader-board with a submission system, we decide to compare our models competency by submitting our methods to ISIC 2019 and comparing them with similar models that have been submitted.

2 Background Research

We give a brief overview of all areas that are pivotal to the aims of the project. We talk a lot of algorithms that we implement in this section, all equations are designed to mimic, as closely as possible, the equations of their respective papers. We also talk about other algorithms we implement in later sections, the difference is that the algorithms in this section are required understand the differences between the methods we wish to compare or are strongly related to our aims, the later algorithms are just used to build a competent classifier and should be the same across all methods.

2.1 Neural Network Architectures

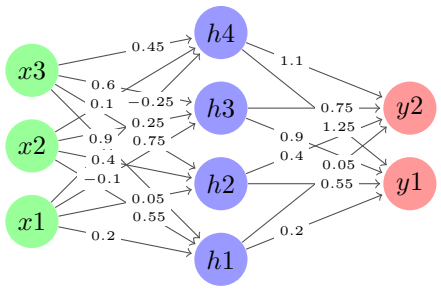


Figure 1: A simple diagram of a MLP, showing weight values and 3 input Neurons connected to a 4 Neuron hidden which then connects to 2 output Neurons.

Multilayer perceptrons (MLPs) are what most would think about when we refer to Neural Networks. Their architecture is somewhat based on the operation of Neurons in the brain[14]. We can see in Figure 2.1, a MLP consists of a input layer, a output layer and some number of hidden layers that are all connected through connections we refer to as 'weights'. In practice, a single Neuron will sum along each of the the weights and add some bias, then we pass that output through a non-linear transformation function, and then pass that on again to the next neuron, this continues until we arrive at our output layer[14]. In our case, we can use the output of the final output layer to make a classification decision.

These weights are usually initialised randomly, then we give our network data to train on, which allows us to calculate our weights values. When we try to find the values for our weights, we try to find the values that best explain the data, maximises a likelihood function or, equivalently, minimise some loss function. This is the frequentist perspective of which we refer to as the Maximum Likelihood Estimation (MLE). We note that the derivative of a function gives the direction in which the function increases, or in the case of loss functions, which tend to be negative, the direction in which it decreases. This is the essence of backpropogation and gradient descent, we calculate the derivative of our loss function then attempt to

descend down the gradient, i.e. find the values that minimise our loss.

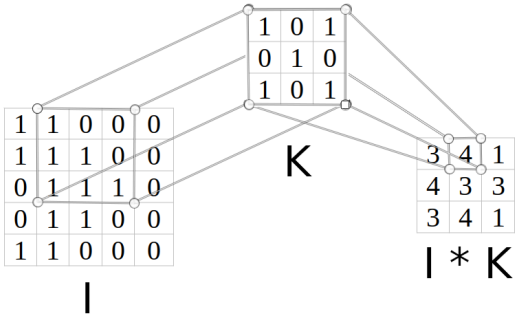


Figure 2: Example of a Convolutional layer.

A Convolutional Neural network (ConvNet) is a type of Network used particularly in image processing that is an addition to a standard NN. They have added hidden convolutional layers that help them to detect complicated patterns in images. A convolutional layer works by 'sliding' a matrix (feature detector) across the pixels of the input vector and, after each slide, outputting the dot product of that section of the image and the matrix (feature map), As seen in Figure 2. In this case the weights are not a single value, but is instead the matrix of values that we use to calculate the output. After our convolutional layer we also pool our output which helps to simplify the model by reducing the spatial dimension of our feature map, State-of-the-art Deep Learning (DL) models use many of these convolutions and dense layers together, counting millions of parameters[28].

ConvNets are usually developed using the current resources and then scaled up for better accuracy as more resources become available [28], Tan and Quoc propose a compound coefficient that can be used to scale Deep Learning models and then propose their own set of models, EfficientNets, of which are easily scalable and effective[28][13].

2.2 Softmax and probabilities

A probability distribution is some set of numbers that describes the likelihood of a event occurring. We can write $p(y)$ To denote the probabilities of each outcome, $y_1, y_2, ..., y_n$ of an event y, we also refer to this as the probability distribution over y. Two key properties of probability distributions is that all numbers must be non-negative and sum to 1. If we apply a softmax function (1) to the raw output of a network, we could think of that network as a probabilistic function, where each probability denotes a classification. Defined as $p(\hat{y}|\hat{x}, w)$, where $\hat{y}$ is our predicted classification label given our unknown input vector, $\hat{x}$, and w are the parameters of our model.

$$\frac{\exp(\hat{y})}{\sum_{n=1} \exp(\hat{y}_n)} \tag{1}$$

The equation for the Softmax Response (1) takes the output vector of a network and applies a exponentiation operation to ensure values are non-negative and normalises the values so that they can sum to 1. We note that non-negative values and summing to 1 are key properties of a probability distribution. Intuitively, if the softmaxed output of a network is meant to represent a probability distribution across possible classifications, then a measure for uncertainty could simply be 1 - the maximum softmax response. Not only is this easy and efficient to implement, but has also been shown to be a surprisingly effective baseline that we can use to compare to other methods[16].

2.3 Bayesian Inference

The intuition behind Bayes Theorum is that, when we know some probability distribution $p(C_k)$ and are given some data, x, such that we want to know the new posterior probability of $p(C_k)$ given that data, $p(C_k|x)$, we must first consider the prior probability $p(C_k)$ and then update that when given our likelihood, $p(x|C_k)$. We put this all over $P(x)$ which can also be expressed as $\sum_k p(x|C_k)p(C_k)$ to ensure that

our posterior probability distribution still sums to 1 over all values of k (2).

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_k p(x|C_k)p(C_k)} = \frac{p(x|C_k)p(C_k)}{p(x)} \tag{2}$$

Using our case of skin cancer diagnosis as a basic example of Bayes theorem, lets presume we have a test that has diagnosed someone with cancer, before we conducted this test, we presumed the chance of this patient having cancer is equivalent to the incidence of cancer in the population. This would be our prior belief $p(C_{cancer})$, then $p(x|C_{cancer})$ is the reliability or likelihood of our test's classification decision. From this we then calculate the actual, posterior, probability that our patient has cancer, $p(C_{cancer}|x)$, by using the reliability, $p(x|C_{cancer})$ of the test to update the prior belief $p(C_{cancer})$.

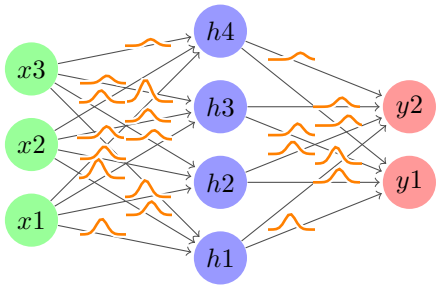


Figure 3: A Bayesian Neural Network, we use distributions of values instead of fixed values

Let us consider using Bayes theorem in a different way, we define our network as $p(w, D)$, where w is our model parameters and D is our training data, $\{x_i, y_i\}_i$. We can now recall how a frequentist Neural Network is trained through MLE, that is, we try to maximise the likelihood (or equivalently minimise some loss function), $p(D|w)$. Now we consider the Bayesian perspective, instead of training to maximise the likelihood, we train to maximise the posterior distribution $p(w|D)$ via Maximum a Posteriori (MAP). This is equivalent to MLE, except we now have an added prior belief about the weights, $p(D|w)p(w)$. Consider if our data is 4 coin flips, and every coin flip lands on heads, the aforementioned frequentist perspective would conclude that all future coin tosses would land on heads, whereas the Bayesian perspective, given a reasonable prior, would come to a less extreme conclusion. We call this overfitting, when our model performs well on the training data but fails to generalise to real world examples. We call methods to reduce overfitting regularization techniques.

$$p(y|x) = \int p(y|x, w)p(w|D)dw \tag{3}$$

Bayesian Inference, unlike a standard Neural Network which uses fixed values as weights as discussed in section 2.1, uses a distribution of values to represent our weights and biases, as seen in Figure 2.3. The primary benefit being that our model cannot over-fit to the training data. Bayesian Inference is the process of computing the posterior distribution $p(w|D)$ which allows us to make a prediction considering all possible values for w (3). Carrying out a predictions as described in equation (3) is intractable as, even though we can choose our prior distribution $p(w)$ and we can work out the likelihood $p(D|w)$ given the data, trying to calculate $p(D)$ (the denominator when trying to calculate $p(w|D)$) requires integration over the high dimensional space of weights.

2.4 Bayes by Backprop

Bayes by Backprop is a method proposed by Blundell et al[5] to train a Bayesian Neural Network while also being able to leverage our usual backpropagation methods. As mentioned earlier Inference is computationally intractable, so instead we use variational Inference. Variational Inference approximates the true posterior distribution by creating a separate, approximate distribution $q(w|\theta)$. We then calculate the parameters of θ that minimises the Kullback-Leiber Divergence (KL-Divergence) between our variational posterior and the true posterior, shown in Equation 4.

$$KL[q(w|\theta)||P(w|D)] = \int q(w|\theta) \log \frac{q(w|\theta)}{P(w|D)} dw \tag{4}$$

However, we still don't know the value of $p(w|D)$, so we can reformulate our cost function into equation 5, the negative of this is also referred to as the Evidence Lower Bound (ELBO), or we can refer to this loss as the Varational Free Energy. We then seek to minimise this loss and, as a result, we maximise the ELBO. We also mention that our likelihood, $p(D|w)$, is the cross entropy loss function, which combines the negative log likelihood with the softmax output, we discuss this loss function in section 3.5.

$$F(D, \theta) = \int q(w|\theta) \log \frac{q(w|\theta)}{P(w)} - q(w|\theta) \log P(D|w) dw \tag{5}$$

$$F(D, \theta) \approx \sum_{i}^{n} \log q(w^i|\theta) - \log P(w^i) - \log P(D|w^i) \tag{6}$$

Minimising the ELBO by this method is still costly, so Blundell recommends that we use monte carlo sampling to approximate $F(D, \theta)$, as seen in equation 6.

There is still one last thing to consider when training our Bayesian Neural Network. We cannot perform our usual backpropogation algorithms on our distribution (parameterized by θ). To deal with this we use something called the reparameterization trick[19]. The essence of the reparameterization trick is that we rewrite our expectation over q so that the distribution to which we take the gradient is independent of θ. We use the optimisation steps as described by Blundell below:

1. Sample $\epsilon \sim \mathcal{N}(0, 1)$
2. Let $w = \mu + \log(1 + \exp(\rho)) \circ \epsilon$
3. Let $\theta = (\mu, \rho)$
4. Let $f(w, \theta) = \log q(w|\theta) - \log P(w)P(D|w)$
5. Calculate the Gradient with respect to the mean $\Delta_\mu = \frac{\partial f(w,\theta)}{\partial w} + \frac{\partial f(w,\theta)}{\partial \mu}$
6. Calculate the gradient with respect to ρ: $\Delta_\rho = \frac{\partial f(w,\theta)}{\partial w} \frac{\epsilon}{1+\exp(-\rho)} + \frac{\partial f(w,\theta)}{\partial \rho}$
7. Update varational Parameters

Where $\mathcal{N}$ is a normal distribution, μ is the mean and ρ is our standard deviation, the varational posterior parameters are $\theta = (\mu, \rho)$ and $\circ$ is point-wise multiplication

Blundell points out that $\frac{\partial f(w,\theta)}{\partial w}$ is the same gradient found by our usual backpropagation methods, so we can leverage our backpropagation algorithms normally.

2.5 Monte-Carlo Dropout

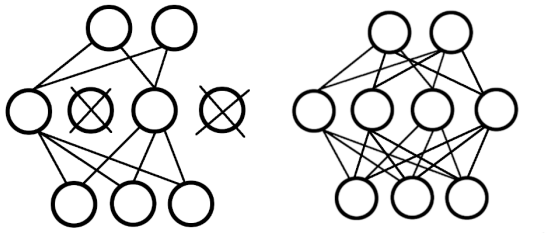


Figure 4: Dropout being applied to a neural network.

Dropout is a popular and effective regularization technique [26]. Dropout consists of 'dropping' neurons from a network (setting their activation function to 0) as seen in Figure 8. We usually use some probability value to determine whether or not a Neuron should be randomly dropped. When we evaluate the model we turn Dropout off, utilising all neurons in the network. Dropout prevents overfitting by stopping complex co-adaption of neurons. Recently Gal et al proposed a new method called MC Dropout[11]. Gal proves that running multiple forward passes through

a network with dropout on and averaging the results is actually mathematically equivalent to Bayesian Inference.

$$\mu_{pred} = \frac{1}{T}\sum_{t=1}^{T} p(y|x, \hat{w}_t) \tag{7}$$

Where $\hat{w}_t$ is the weights at t forward passes

The implementation is remarkably simple. we need only average our output over the multiple passes as defined in equation 7. The equivalence to Bayesian inference is that, in Bayesian inference we sample multiple times from our varational posterior, averaging the results, in MC Dropout, every forward pass we run is equivalent to sampling from a varational posterior[11]. This method brings many benefits, not only is it computationally cheap but models that have been trained with dropout do not need to be retrained and can use MC Dropout immediately.

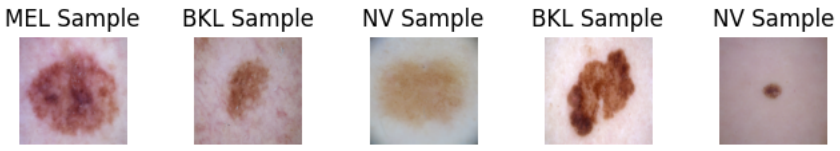


Figure 5: Different types of skin lesions

2.6 Skin lesion classification

Skin cancer is the most common type of cancer, each year we see two to three million new skin cancer cases, with 132,00 being melanoma[24]. Thankfully, skin lesions can be identified by a dermatologist with an unobtrusive visual examination. Previously, computer aided support was limited to creating algorithms that attempted to identify the same visual cues that dermatologists identified[15][7]. DL can automatically extract these higher level features, automating much of the process and showing great results[10].

Skin cancer is usually grouped into non-MEL and MEL categories, due to the fact that MEL accounts for the majority of skin cancer related deaths[24][7][15]. It is critical then, that MEL skin lesions do not get incorrectly classified as non-MEL.

2.7 Cost of miss-classification

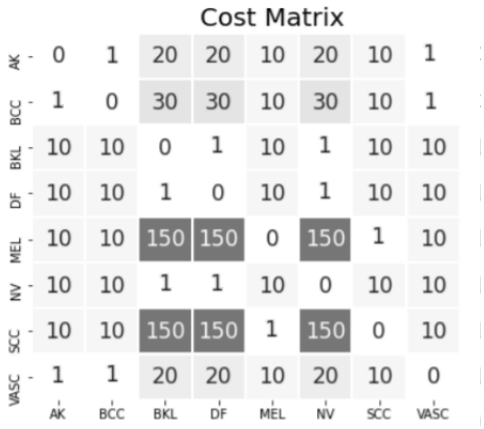


Figure 6: The cost matrix L_{kj}, The rows correspond to the true class C_k and the columns correspond to the predicted class C_j. Cost matrix was provided by a dermatologist who works with the University of Dundee

Even though it is convenient and intuitive to use ac-

curacy to compare performance of Neural Networks, using accuracy presumes that all incorrect classifications are equally costly. This is not the case, as stated in section 2.6, melanoma is the deadliest form of skin cancer. In cases like this, we don't just want our network to miss-classify, but to miss-classify in a 'correct' and non costly way. To do this we introduce a cost matrix, as seen in Figure 6. The classes appearing in Figure 6 are as follows: Melanoma (MEL), Melanocytic Nevus (NV), Basal Cell Carcinoma (BCC), Actinic Keratosis (AK), Benign Keratosis (BKL), Dermatofibroma (DF), Vascular Lesion (VASC) and Squamous Cell Carcinoma (SCC).

$$EC(C_j) = \sum_{K} L_{kj} p(C_k|x) \tag{8}$$

In this cost matrix we can find the cost of miss-classifying one class as another by using our predicted label, C_j, and our true label, C_k, and then looking up the relevant cost in the matrix L_{kj} (note that diagonal values, when k = j, the cost is always 0, as we do not incur penalties for correct predictions). In a real world setting, we do not have the ground truth, so we employ the standard method to get Expected Cost values given a probability distribution[4]. Given our softmaxed outputs of the network $p(C_k|x)$, where x is our input vector, we can calculate the expected cost by summing across all the true labels and multiplying by the probability that our classification is correct. By doing this we can calculate the Expected Cost of making a classification decision $EC(C_j)$, as shown in equation (8). When concerned with costs, we are frequently only concerned with the classification decision that minimises Expected Cost, i.e. The value of j that minimises equation (6), we refer to this as the Lowest Expected Cost (LEC).

We consider as well that, when we make a prediction using the maximum probability, we could also think about this as applying a cost matrix with a cost of 1 on everywhere except the diagonal (i.e. cost is always 1 when $j != k$), we call this a flattened matrix. Considering this and the fact that we will be using Figure 6 later in our results section, we avoid confusion by simply referring to Figure 6 as "our cost matrix" and, unless specified otherwise, we can presume that all mentions of using a cost matrix refers to Figure 6 and not our flattened matrix.

We also would like to note that we make some assumptions about Figure 6. When this cost matrix was created, it was made to be a generic cost matrix that can be applied to any skin lesion classifier. In a real world setting, we would see that the cost of miss-classifying one lesion as another is highly specific to where our model is deployed in the diagnostic process. As a result, we just assume that this cost matrix works well with our population and use case.

2.8 Comparison of methods

BbB is given the disadvantage due to having two times the number of parameters, having both a prior and posterior distribution for the weights. Blundell states that we do not train our prior distribution, as this gives detrimental results[5], instead choose our starting prior parameters carefully. Despite this, BbB still is the most computationally expensive method, taking roughly two times as long as Dropout[5].

MC Dropout also has a substantial advantage over BbB due to its ease of implementation and frequent use as a regularization technique[11][26]. Dropout comes as a built in method in a lot of machine learning frameworks, whereas we had to implement BbB using Pytorch's gaussian classes. This means that as the libraries used get updated we will still need to maintain our BbB implementation whereas dropout will be maintained by the developers of the framework. Due to dropouts prevalence as a regularization technique most models do not need to be retrained and can begin using MC Dropout with no extra cost[11], except the time take to compute the forward passes. MC Dropout then only needs to demonstrate some benefit to be considered worth implementing, it is worth noting that this isn't guaranteed as we found a study that shows MC Dropout is only comparable to SR and performed worse on the ImageNet dataset[12].

We also note that we cannot find any comparison of our methods on the specific case of using a cost matrix to correctly miss-classify skin lesions.

3 Design

The primary aim of this project was to give a comparison of different variational inference methods, the building of a good classifier occurred to serve this aim. As a result, describing official design techniques seem somewhat inappropriate, The program itself is rather small in scope, not being able to be broken down into many parts. Most of the time was used tweaking the network to find the optimal parameters to train a competent classifier. The only design methodology that was used was the Waterfall approach, perhaps if given a larger program to implement, it would be easier to break into smaller parts, and a agile style approach may be better. Instead we go over and justify some of our choices for how we designed the architecture of our Network, the primary difference between the algorithms here and in section 2 being that these algorithms work to create a competent classifier and are not pivotal to understanding the aims of the project.

3.1 ISIC 2019

Standard datasets for skin lesions tend to be small with around 200 images and contain very few classes[18]. For the past few years the International Skin Imaging Collaboration (ISIC) have been realising annual challenges with progressively larger datasets. We chose to use the ISIC 2019 dataset, which was composed of the HAM 10000[29], BCN 20000[8] and the MSK dataset[7]. ISIC 2019 has a total of 25,331 images of 8 different classes. They also released a second set without ground truth labels, which models could predict on and submit their score to the ISIC 2019 live leaderboard. This second set had 8,238 images with a 9th 'unknown' class. ISIC does not specify what the unknown class is comprised of, but it is presumably out of sample data. ISIC scores models based on their ability to classify the 8 identified classes and the models ability to measure uncertainty on out of sample data.

When we talk about uncertainty of out of sample data we refer to aleatoric uncertainty, because we do not have access to this 9th class we will not be measuring aleatoric uncertainty, but instead epistemic uncertainty, things our model could in theory learn, but doesn't due to lack of training data.

We chose the ISIC 2019 dataset because it is one of the largest publicly availible skin lesion dataset and because the live leaderboard is a convenient way to compare model competency. There are some problems with ISIC 2019 however. There is a major class imbalance we need to consider, the largest class contains 12,875 and the smallest class contains 239 samples. We detail how we overcome this problem in the coming sections.

3.2 Data

As we mentioned, one of the problems we face with ISIC 2019 is a large imbalance between the smallest and largest class, to help artificially boost the number of samples in our training data, we can Augment the data before feeding it into our network. Allowing us to see slightly changed versions of the same image allows our network to learn more robust features, as well as benefiting our sampling method described in section 3.5. Augmentations on images usually consist of things like Jittering various properties of the image (brightness, hue, saturation etc.) or altering the contents of the image via cropping or zooming. We can see the results of performing our Augmentations in Figure 7. In the interest of clarity, here is a brief list of all our transformations: horizontal and vertical flips, random rotation by between 35° and -35°, crop each side equally (center crop) by a sixth of the image, shearing by 0.05%, random resize and crop by another sixth, down to a image size of 224 pixels, saturation, hue and brightness are jittered by 20%, randomly cut out holes 4 holes with probability

0.2 per hole and we normalize the image. We apply these augmentations to only the training set, the only transform we apply to our validation and training data was a resize to and image size of 224.

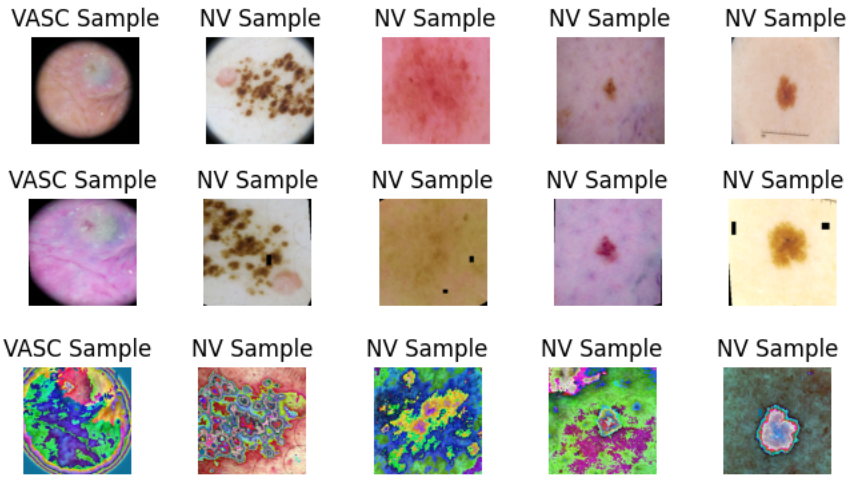


Figure 7: Figure showing our augmentations. Top row shows skin lesions with no augmentations, then we see skin lesions with every augmentation except normalization and finally we see all augmentations

To train and test our network we must also split the dataset into different sets. We use 70% of the images to train our network, then we 10% as a validation set, which we test our network on after every epoch to ascertain whether our model is generalising well to real world data or if it is over-fitting. We use the remaining 20% to test our models abilities, on totally unseen data. We also batch our images into mini-batches sizes of 16, this comes with a couple benefits, but we are particularly concerned with the faster computation times, thanks to the parallelism of modern computing platforms[17]

3.3 Network Architecture

Due to the costly nature of training a DNN and our limited resources, we chose to use EfficientNet with compound Scaling 0 (EfficientNetb0), we feed the augmented image to EfficientNetb0, then extract the output and use 2D adaptive average pooling. We pipeline that output into a single 512 neuron dense/Bayesian layer.

We also use batch normalization, this normalises each of our batches to have a mean of 0 and standard deviation of 1, this has the effect of stabilising the learning process[17]. We use the Rectified Linear (ReLu) activation function, a standard activation function[23] that works by setting all negative input to 0 or allowing the input to pass through unchanged. We also use a drop rate of 0.5, Gal recommends a drop rate of 0.1 or 0.2 in his original paper, but he kept network architecture small to avoid overfitting, our network was too complicated to try this. Using a drop rate of 0.5 retained dropouts regularization benefit, this value of 0.5 has shown to work well with MC Dropout[3]. The last regularization technique we use is weight decay or L2 regularization, this punishes the network when it uses weights that are too large. The intuition being that smaller weight values force the network to suppress any irrelevant components by solving the problem using the smallest weights possible[20].

The primary benefit of this architecture is that, we need not run multiple passes through the entire EfficientNetb0 to obtain our predictions, simply once through EfficientNetb0, then we run multiple times through our 512 neuron Bayesian layer or dense layer with dropout turned on.

EfficientNet also has a pretrained version available that we make use of, we download weights that have been pretrained on the ImageNet dataset. This has been proven to either give a small benefit as our model already has knowledge of how to identify basic patterns[30].

3.4 Optimiser

For this network we opted to use the Stochastic Gradient Descent (SGD) Optimiser, with a momentum value of 0.9, a weight decay value of 0.00001 and a cyclic learning rate scheduler. Momentum attempts guess the size of the step we take in gradient descent based on the previous step, i.e. if a large step was taken previously, its intuitive to presume that the next step will also be large[27]. We found that using a cyclic learning rate scheduler also gave us notably better results, this scheduler functioned by changing the learning rate up to some specified maximum learning rate, then stepping down to some minimum learning rate. The theory is that, when we perform gradient descent we attempt to find the global minima but can sometimes get stuck in local minima. A cyclic scheduler can help us escape local minima[25]. We used a step up size of every 5 epochs, then a step down size of a further 5 epochs, we then decrease the maximum learning rate by half every time we reach our minimum learning rate.

We try to keep both models as similar as possible to keep a fair comparison, however, we found that the Bayesian Layers required a notably larger initial learning rate, as a result, we set the max learning rate of our Bayesian layers to be 0.08 and the learning rate of our EfficientNet layers to be 0.01. Both methods used a minimum learning rate of 0.0001.

3.5 Loss Function

Due to our need for well calibrated distributions and large class imbalance, we initially decided to go with the weighted Cross entropy loss function, defined as equation (9). This loss function has the benefit of being able to accept weights to punish more for incorrect predictions on classes with fewer samples.

$$Loss = -\sum_{c=1}^{C} w_c(y_c + \log(p_c)) \tag{9}$$

Where w_c is our assigned class weight, y_i is our ground truth label and p_c is our softmaxed predictions

The cross entropy loss attempts to combine the softmax output of a network with the negative log likelihood of the data. We used equation (10) to obtain our weights for our classes, we can see that, using this equation, if there is a higher number of samples in a class, we would obtain a larger weight. equation (10) isn't fundamentally different from simply doing $\frac{1}{N_c}$, but this way our values stay above 1, which helps with debugging.

$$w_c = \frac{N}{N_c} \tag{10}$$

Where w_c is the weight for class c, N is the total number of samples and N_c is the number of samples in our current class.

We can weight our loss function to deal with the class imbalance, but due to the extreme nature of our imbalance, with the smallest class being less than 1% of the total dataset, we found that employing stratified sampling gave us better results. We use our class weights to generate probabilities that we use to determine how often we should sample each image. This means that some images could be sampled hundreds of times more than others, which is why it was pivotal that we had such a wide variety of data augmentations. As a result, we decided to use two loss functions, one unweighted to deal with training and backpropogation, and one using the same weights we

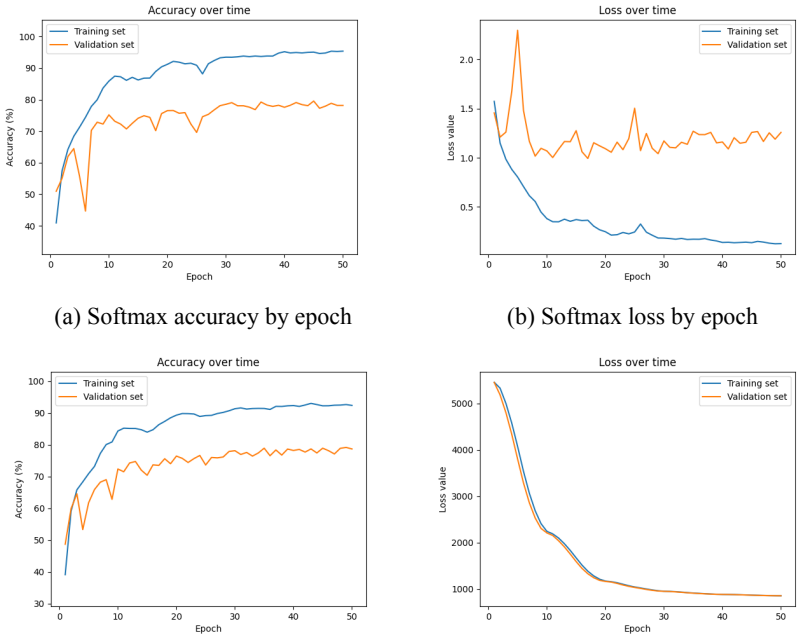
used to randomly sample to create a weighted cross entropy loss function to use for the validation set, of which we sample from normally.

3.6 Prior and Posterior Distributions

As mentioned in section 2.4, Bayes by Backprop uses both a prior and posterior distribution of which we can choose the type. For our varational posterior we use a gaussian distribution and for our prior we use a gaussian scale mixture. This is a scaled mixture of two gaussian distributions, Both have a mean value of 0 but with differing variances. When variance 1 (σ_1) is greater than variance 2 (σ_2) and variance 2 is much less than 1 ($\sigma_2 << 1$) we obtain a "spike and slab" prior that concentrates around the 0 mean. We define this distributions probability function as equation (11), where w_j is the jth weight of the network and $\mathcal{N}(x|\mu, \sigma^2)$ is the gaussian density evaluated at x.

$$\log p(w) = \sum_j \log(\pi \mathcal{N}(w_j|0, \sigma_1^2) + (1-\pi)\mathcal{N}(w_j|0, \sigma_2^2)) \tag{11}$$

We found that values of $\pi = 0.5$, $\sigma_1 = 1$ and $\sigma_2 = 0.0025$ worked best for us. We chose this prior and posterior as we wish to stay as close to Blundells original paper as possible, though we acknowledge there are alternatives[5]. When is comes to monte carlo sampling from our varational posterior during training time, we found that 3 samples gave good results.



(a) Softmax accuracy by epoch

(b) Softmax loss by epoch

(c) BbB accuracy by epoch

(d) BbB loss by epoch

Figure 8: Our training curves

3.7 Training

We feed our training data to our network and perform backpropogation and gradient descent using the loss function and optimiser as described above. We train our network on 50 epochs, that is, we give our network the number of samples in the training dataset 50 times over. We attach the curves we used to debug our model, showing our loss values over time and our accuracy over time in Figure 8. We use our training curves by tweaking the parameters of our network and observing the effect on the validation loss. Some slight overfitting does seem to occur after around 30 epochs for the softmax response. We instead consider running only 30 or 40 epochs through the network and also tried performing early stopping based on the best loss value across 100 epochs. These results ended up being unreliable or consistently worse on across all our metrics in Section 6, so we decided to use 50 epochs. We note as well that, for MC Dropout and SR we use the same model after 50 epochs of training, as this was an advantage Gal mentioned[11], that dropout can be applied to existing models. For BbB we train a separate network.

3.8 Entropy as an Uncertainty estimation

Entropy is a concept in Information Theory related to measuring the level of "Information" or uncertainty across a set of probabilities. We chose to use Entropy measured in bits ($\log_2$) to quantify our level of uncertainty across predictions as this was one method that was recommended by Gal[11] as well as being proven by other studies to be an effective uncertainty estimation[9][3]. We combine this with running multiple forward passes to get the average entropy by forward pass show by equation (12). We note that there have been some studies showing that variance across predictions actually gives better uncertainty values, but we found that variance gave us rather poor values while entropy gave us better values.

$$\mu_H = \frac{1}{T}\sum_{t=1}^{T} -\sum_{i=1}^{n} p(y_i|x_i,\hat{w}_t)\log_2(p(\hat{y}_i|\hat{x}_i,\hat{w}_t)) \tag{12}$$

$$H_{norm} = \frac{H - H_{min}}{H_{max} - H_{min}} \tag{13}$$

Where H is entropy

We also note that entropy doesn't necessarily end up within the range of 0-1, so we used normalised entropy instead, shown in equation (13).

4 Implementation

In this section we talk less about the algorithms we used to create a competent classifier, but instead the Technology we use and the metrics we think will be useful to evaluate this classifier.

4.1 Python and Pytorch

Everything was implemented using the Python Programming Language. Python was the most obvious choice, not only is its focus on simple and "Pythonic" approach to programming allow us to rapidly prototype, but Python also has a wide range of available Machine Learning Libraries and utility libraries based around Machine Learning. There were a few different options to consider with regards to relevant Frameworks.

Pytorch is a widely used, open source platform for machine learning. It provides high-level machine learning tools including things like automatic backpropogation. Keras was another good option, it is the most widely used platform for Machine Learning and is notable for being beginner friendly, allowing rapid and easy deployment. We decided on Pytorch however, as Keras ended up being too beginner

friendly, deliberately hiding a lot of its more intricate functions. Some of our methods required substantial modification to the training and testing procedures, Keras doesn't allow you access to these innately, whereas Pytorch does, making Pytorch easier to work with.

4.2 Miscellaneous libraries

The majority of the project was made possible with Python and Pytorch, but in the interest of clarity, we quickly summarise and justify our choices for some of the other software that played a large role in this project:

EfficientNet-Pytorch: The EfficieNet model we used had been created and maintained by github user lukemelas. We install and use his implementation designed for pytorch, which follows the implimentation as defined in the original paper.

matplotlib/seaborn: An industry standard, both matplotlib and seaborn are easy to use graphing libraries that we use to plot all graphs seen in this project.

sklearn: A helpful machine learning library that we used for calculating the area under our curves.

Pandas: Pandas is a flexible and open source data manipulation library, we use this to load in, manipulate and handle our large volumes of data.

PIL: Python Imaging Library (PIL) has built in support with pytorch and has fast computation times, which was helpful for the large number of images we rapidly loaded.

numpy: Numpy is a fundamental package for scientific programming, allowing high level manipulation of multidimensional arrays, we used this to manipulate the output of our model and generate the results that we plot.

Github: Github was used for source control, allowing us to roll back any mistakes during implementation and create different branches for our different methods.

4.3 Evaluation

We used a variety of techniques to visualise the performance of each of our proposed methods. To generate our predictions we run 100 forward passes (Or sample 100 times from the variational posterior) and average them, as described in Section 2.

First we compare our different methods by the simplest approach, we compare accuracy and the number of correct and incorrect classifications per sample. These plots are fairly simple and will be easier to explain in the results section, alongside their relevant figures. The only technique used that requires some more in depth explanation would be the risk coverage curve. For our risk coverage curve we plot accuracy by coverage, coverage being the percentage of the data-set that is being tested. To decide which percentage to test and which percentage to reject, we use a standard reject option[4].

We use some parameter θ as a threshold, all probabilities below that threshold will be rejected and removed. To get our value of θ,we retreive all the classification decisions and their relevant probability, then find the lowest probability in that list and use that value as a value for θ, rejecting that sample. We then re-plot accuracy and find the second lowest probability in our list, using that for θ, then we re-plot accuracy again... and continue until the entire dataset has been rejected by the threshold, and we end up with a coverage of 0 and accuracy of 100 (We decide to make no classifications 100% accurate so that the graph is easier to read).

We should also note that, when we consider coverage curves, we don't usually comment on the values on the far left side of the graph, as it can be noisy at low values of coverage (Caused by a overconfident prediction being incorrect). We choose risk coverage curves as they give a nice overview of how the model performs across the entire dataset.

We recall that, as we stated earlier, accuracy is not a good comparison of how well our model will perform in a human-machine team. We can then look at

how well calibrated our machines predicted distributions are, The idea behind a well calibrated probability distribution is that, if we have 100 samples making the same classification decision with 60% certainty, we would expect 60 of those predictions will be the correct classification label. To create a visual comparison we use reliability diagrams of each of our proposed methods. We group all of our predictions by probability range, i.e. probabilities in the range 0.0 to 0.2, 0.4 to 0.6 ... 0.8 to 1.0, Then we count and the frequency of the relative classification occurring.

We also wish to measure how costly each of our methods are using our previously mentioned cost matrix (Figure 6 in section 2.7), to do this we use a risk coverage curve and a confusion matrix. A confusion matrix is a easy way to visualise the incorrect classifications of our model. Each row of the matrix represents the instances in the true class, while each column represents the instances in a predicted class, if our model makes a classification decision, we check the true label add 1 into the relevant point in our matrix.

For our second set of risk coverage curves, we employ a similar method as for accuracy, but instead we use our LEC to make a classification decision and we measure Test-Cost instead of accuracy. Test cost is the average actual cost incurred for a classification decision on the test set. We also use a similar method for finding θ as we did for accuracy, except we now search for the highest LEC in our list of LEC classification values, and use that as θ.

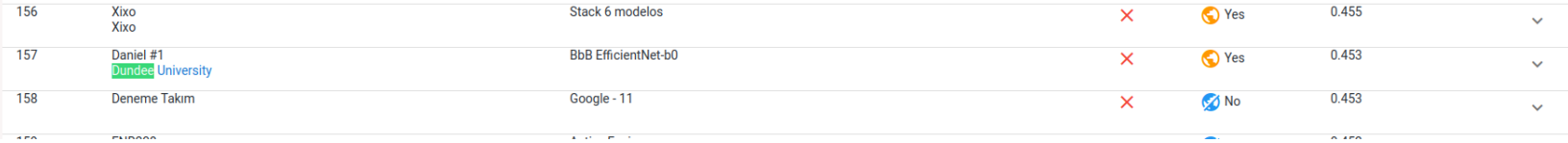


Figure 9: Figure showing our position on the ISIC leaderboard

5 Results

5.1 ISIC Submission

We quickly take the introduction of our results section to observe our submission to ISIC, Figure 9. The most interesting thing to note is that, as far as we can tell, we have the best EfficientNetb0 model on the leaderboard, with the next best EfficientNetb0 taking place 172, 15 places below us. ISIC only allows one spot on the leaderboard for all your submissions, and we found BbB performed particularly well on their unknown set. We don't want to make much comment on BbB's ISIC performance, as ISIC doesn't give us much information about what their test set contains. We can say that this model performs competently, compared to other models. MC Dropout scored similarly, getting a score of 0.445 and SR obtained 0.444, still better than the next best EfficientNetb0 model.

5.2 Accuracy

We observe Figure 10, which shows the accuracy at each forwards pass and shows entropy as a shaded background. From this we see that after around 30 forward passes our accuracy becomes stable and doesn't require many more forward passes. It is interesting to observe that MC Dropout does not outperform the Softmax Response but performs comparably. We hoped to see entropy values growing as forward passes continued, but this also seemed somewhat stable after the first few passes. BbB seems to

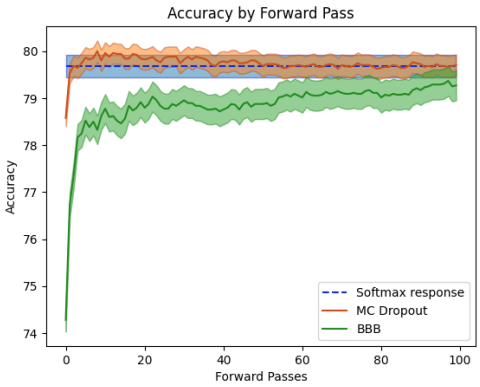


Figure 10: Accuracy by Forward pass

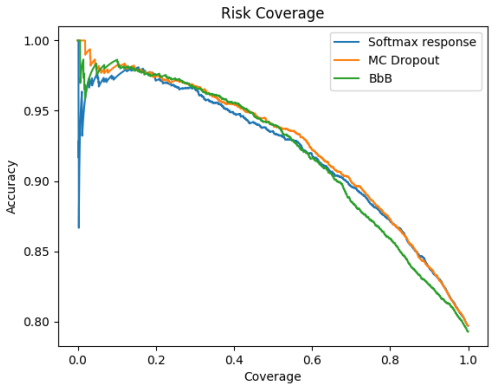
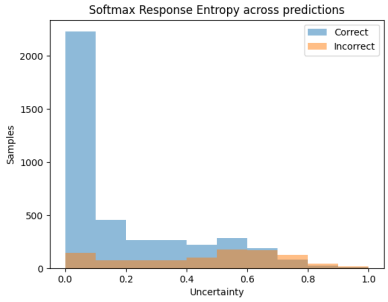
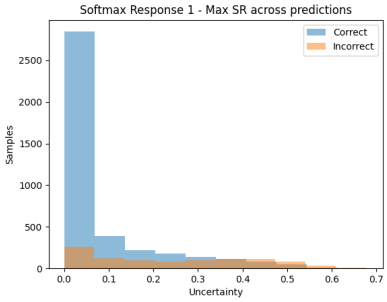


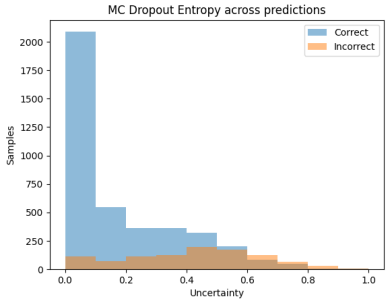
Figure 11: Accuracy by Coverage



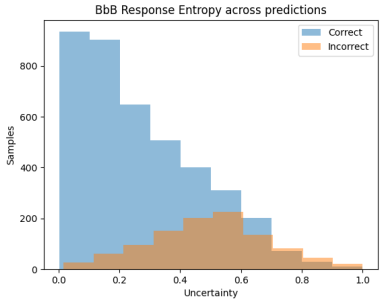
(a) Softmax Entropy



(b) Softmax Baseline

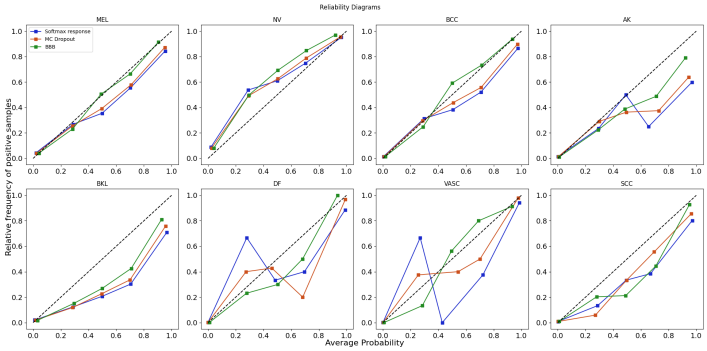


(c) MC Dropout Entropy

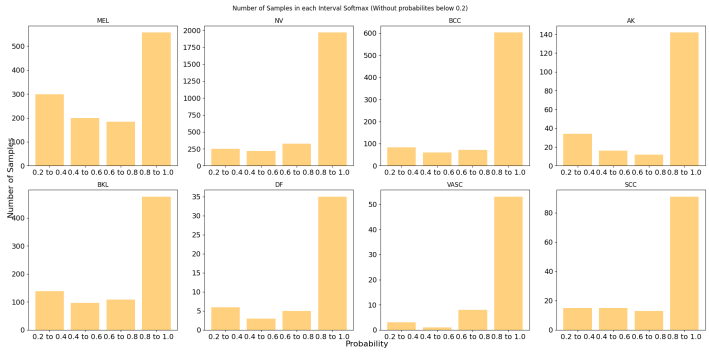


(d) BbB Entropy

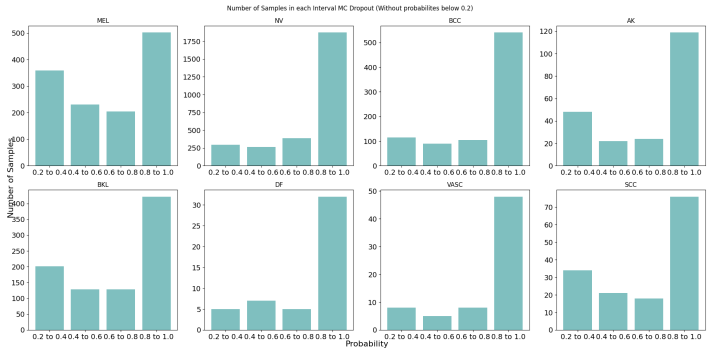
Figure 12: Various histograms showing Entropy across correct and incorrect predictions



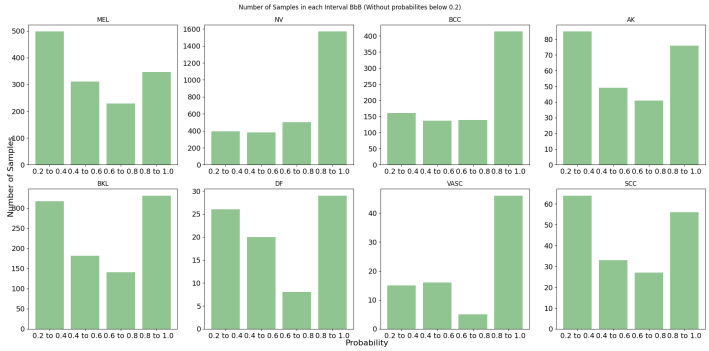
(a) Reliability diagram showing the relative frequency of positive samples in each probability interval



(b) Number of Softmax samples in each interval



(c) Number of MC Dropout samples in each interval



(d) Number of BbB samples in each interval

Figure 13: Histograms that show the number of samples that fall within each probability range, alongside our Reliability Diagram

be still slightly increasing in accuracy, implying that perhaps more than 100 passes could give further improvements.

We now turn our attention to Figure 11, our risk coverage curve. Once again, we see that our results are similar enough that we cannot draw any definitive conclusions pertaining to each method, these results are somewhat expected after observing Figure 10. It is Interesting to note that, if concerned solely with the binary metric of accuracy, then MC Dropout might be worth implementing, even if the increase in accuracy is very slight.

Giving some form of uncertainty estimation for model predictions would help with human-machine teams working together. For this to be viable however, we need to ensure that more incorrect classifications have a higher uncertainty metric. We can look to Figure 12(a, c and d), were we have plotted the number of correct and incorrect samples, with our uncertainty metric, normalized entropy. From these Figure we can see that, for MC Dropout and BbB, incorrect classifications have a higher normalized entropy. We can conclude that BbB and MC Dropout do perform better in this regard, however it is still somewhat unsatisfactory as the incorrect predictions peak at around 0.5 entropy, and even then, there is more correct classifications than incorrect classifications, making it impossible to give intuitive threshold of which all values afterwards should be rejected. We also mentioned that an intuitive baseline for uncertainty is simply 1 - the max classification probability, to be fair to the softmax response, we plotted both 1 - max probability in Figure 12(b). We see that neither performs particularly well, but using entropy as a uncertainty metric was better than this simple baseline. We also show a full breakdown of these histograms by class in Appendix E

5.3 Calibration

Figure 13a shows our reliability diagrams and Figure 13(b c and d) shows Histograms with the number of samples that fall within each probability interval for each method. We include histograms showing samples in probability intervals between 0.0-0.2 in Appendix E, we do not include them here as they can make the Figure 13 difficult to read. Observing Figure 13(b and c), We decide to be somewhat uncertain of our VASC and DF results for MC Dropout and SR, as we can sometimes see less than 20 samples in our probability intervals. We plot a dashed black line on our reliability diagrams for reference to what a perfectly calibrated model would achieve.

Interestingly, we see that BbB seems to be consistently better calibrated than SR and MC dropout on some classes, and those that it isn't better calibrated to, such as VASC, SCC, the results are close enough that it is fair to conclude that it performs comparably and not worse. We also note that, in Figure 13(b c and d), the majority of samples in the each interval for MC Dropout and the SR primarily lie within the 0.8-1.0 range, whereas BbB is less confident in its correct predictions, increasing the viability of the BbB reliability plots on the smaller classes. This result is likely due to the fact that Bayesian networks avoid making extreme conclusions when observing the training data, as discussed in Section 2.3.

5.4 Cost

We apply our cost matrix as described in Section 2.7 and now choose our predictions based on the lowest expected cost (LEC). We normalize our confusion matrices (Figure 15) by dividing across the rows by the total number of samples in the respective class. We pay particular attention to the MEL and SCC miss-classifications, as these are the most costly miss-classifications as defined in our cost matrix. We see our intended results for MEL, noting that there are roughly 17% more correct MEL classifications. More importantly the miss-classification of MEL as NV (Which carries a cost of 150) goes from roughly 17% to below 0.05%. SCC Classifications change very little, however we also note that costly SCC miss-classifications are BKL, DF and NV, of which were already less than 10% of the miss classifications, and do occur less after the cost matrix has been applied, but not on as much of a scale as MEL miss-classifications.

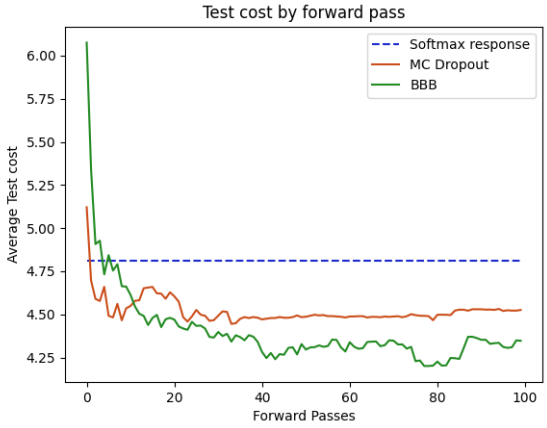


Figure 14: Figure showing Average Test cost at every Forward Pass

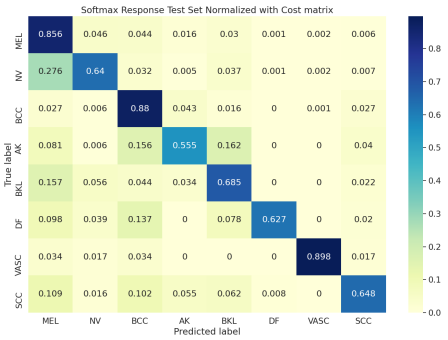
In our previous results we used accuracy as it is an intuitive and easy to understand metric. In Figure 14 we plot a similar cost by forward pass as seen earlier in Figure 10, except now we use our cost matrix to get the actual cost of our predictions. Notably, we now evaluate a range of values, instead of getting the binary classification of correct or incorrect to calculate accuracy. This brings with it some confusion, whereas with accuracy it is easy to understand how much better one method can be, the idea that one method might save 1 cost can be a difficult benefit to evaluate. Before our discussion we acknowledge that a difference of 1 cost can only be determined to be significant by the creator of the cost matrix, however due to the nature of our cost matrix being general and unrealistic, as discussed in Section 2.7, we shall presume that any small increase in cost is significant.

We look at Figure 14, showing that our average test cost seems consistent after 100 forward passes, giving some viability to our results. We acknowledge that BbB doesn't fully flatten out, and both methods require more passes to stabilise when compared with Figure 10, we concluded that more than 40 passes is adequate for both methods.

Figure 16 shows our coverages by average test cost. Figure 16(b) gives some interesting results, showing that BbB consistently performs better across all values of coverage. This result is rather intuitive, given the knowledge that BbB is better calibrated we could presume that it would be able to make more cost-aware decisions. It is also not surprising then, considering MC Dropout and SR performed comparably in our Reliability diagram, that they repeat that pattern again, being very similar, with MC Dropout performing slightly better. By breaking Figure 16(b) down by class, Figure 16(a), we can gain some further insight to why BbB performs better. We note that BbB performs remarkably well on MEL, nearly halving the test cost across all coverage values when compared to the SR or MC Dropout, BbB performs worse on less costly classes, such as NV or BKL, likely due to potential costly MEL miss-classifications instead becoming less costly BKL or NV classifications.

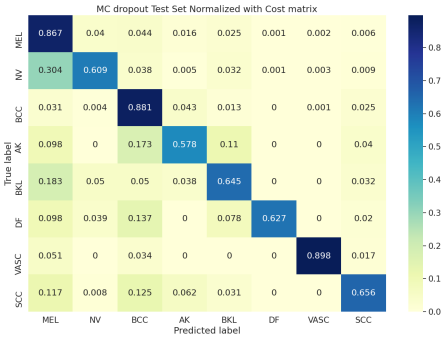
5.5 Discussion

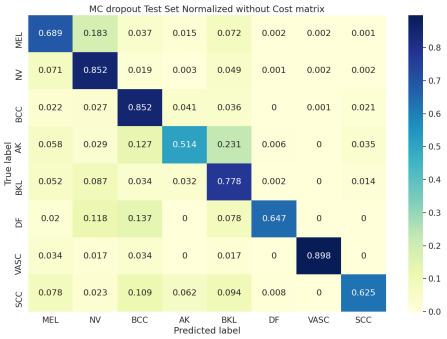
Table 1 aims to give a general overview of our results, as well as showing the time taken to run our forward passes. AUC scores give us a easy single metric for to represent our accuracy or cost over all the values of coverage, we computed these from Figure 11 and Figure 16 respectively. When concerned with only accuracy we can say that all methods are similar enough that we do not see a notable improve-



(a) Confusion matrix showing the Softmax response's Normalized predictions with a cost matrix applied

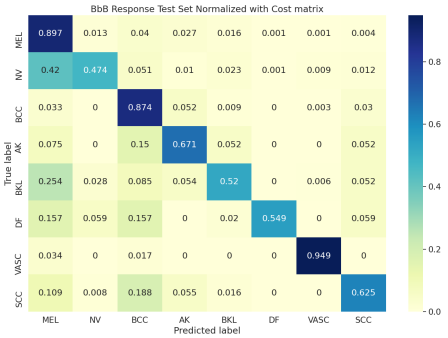
(b) Confusion matrix showing the Softmax response's Normalized predictions without a cost matrix applied

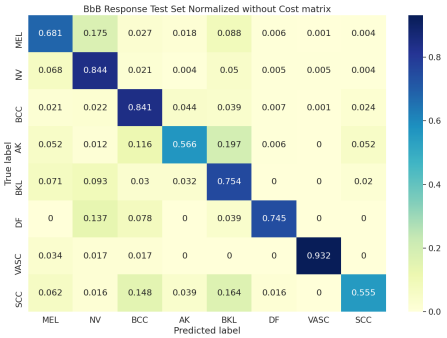




(c) Confusion matrix showing MC Dropout's Normalized predictions with a cost matrix applied

(d) Confusion matrix showing MC Dropout's Normalized predictions without a cost matrix applied

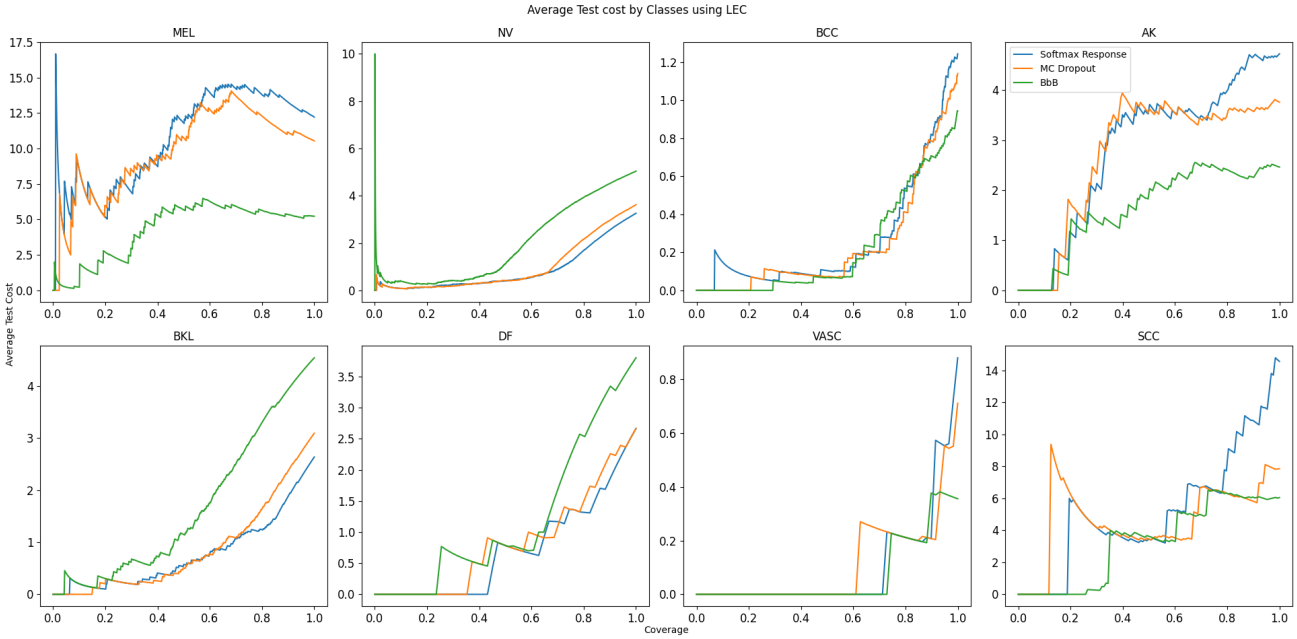




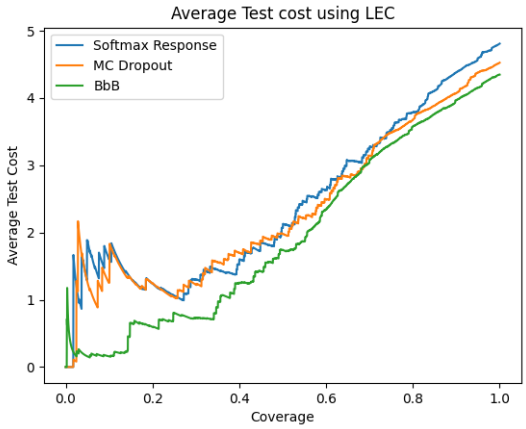
(e) Confusion matrix showing BbB's Normalized predictions with a cost matrix applied

(f) Confusion matrix showing BbB's Normalized predictions without a cost matrix applied

Figure 15: Confusion Matrices showing the use of a Cost matrix on the miss-classification of the Softmax Response



(a)



(b)

Figure 16: These risk coverage curves show how the average test cost changes as more of the data is rejected.

ment, given that MC Dropout does have a higher AUC value and it's relatively easy to implement nature, we would conclude that this method would be adequate.

Larger computational time was demonstrated by the use of running multiple forward passes, we display our results of this in minutes and seconds in table 1, We measure computational time as the time taken to produce predictions on our 5066 test samples. We note that, as expected, MC Dropout takes roughly double the time of a standard SR and BbB takes roughly 2 times longer than MC Dropout. We use rough ratios as our times are dependant on hardware used, but we presume that the ratio between time taken will remain the same. We consider the time taken to compute our results when compared to a average human classifying skin lesions, it is obvious that all methods will be able to outperform manual in terms of speed, but how fast these method need to be is too reliant on the hardware and use case for us to make a comment. We do not mention training time here, as all models took roughly 8-9 hours to train, training time would've been more important if we considered using a deep ensemble of Neural Networks.

Method	**Accuracy**	**Acc. Covg. AUC**	**Avg. Test Cost**	**Cost Covg. AUC**	**Comp. Time**
Softmax Response	79.69%	0.92	4.81	2.451	3:27
MC Dropout	79.73%	0.925	4.53	2.362	7:36
Bayes by Backprop	78.96%	0.919	4.35	1.963	12:50

Table 1: Table showing an overview of useful results, including the Area Under Curve (AUC) values for our risk coverage curves. Acc. is shorthand for Accuracy, Covg. for Coverage and Comp. for Computation.

An important factor to consider when discussing the cost of our methods would be the prior distributions of skin lesions in our population. Our model was trained using the ISIC 2019 dataset, this dataset had a major class imbalance. By training and testing our model on this dataset we acknowledge our assumption that the number of samples of each classification in the dataset are representative of the general population. We also note that our cost coverage curves are dependant upon this population, SCC was a particularly costly miss-classification and yet was less than 3% of the total data-set, any of our methods could've made some terrible classification decisions in regards to SCC, but the overall cost would be low because of a large volume of correct NV classifications. This should be considered when evaluating our model, BbB predicts costly on NV and remarkably well on MEL, despite this, our test cost AUC is still somewhat comparable to MC Dropout and the SR due to their less costly predictions on NV. If in a population in which 70% of skin lesion images were MEL, then, using our currently trained model, BbB test cost AUC would be notably lower and if it was 70% NV then it would be notably higher. It is remarkable then that, even though BbB performed more costly on NV, which was over 50% of the dataset, it still manages to maintain a lower test cost over all values of coverage, from this we conclude that, if the cost of miss-classification is a concern in a this specific population, with our specific cost matrix, then BbB would perform best.

6 Conclusion

The primary aim of this project, to give a comparison between MC Dropout, SR and BbB in a different environments, was a success through our through our variety of experiments that we ran. We found no notable difference in the accuracy of the proposed methods, though MC Dropout did seem to perform

slightly better in this regard and its ease of implementation means that we could potentially conclude its superiority over SR. Though BbB still performed comparably on this metric, the complexity required to implement and maintain as well as the extra time taken to run the forward passes is not worth the result. When we consider a cost-aware environment however, where the calibration of our methods as well as the cost effectiveness is concerned, we find that BbB, despite the unbalanced testing data, still outperforms MC Dropout and the SR. We are still concerned about whether or not BbB would help in a human-machine team, we would require more research into real world results, but that is somewhat outside of the scope of this project. We can conclude that, of these chosen methods, BbB would most likely perform best given our observations.

7 Appraisal

There were multiple learning curves to overcome when this project was taken, a particularly difficult area was the reading of academic literature, most papers presume a baseline of machine learning knowledge and jargon, which can make them seem somewhat impenetrable to someone who has no prior knowledge of the area. Alongside this, the papers the student looked at also introduced somewhat abstract concepts, such as Bayesian Inference, of which there was also no prior experience. A particular area of note was the implementation of BbB, though Dropout was easy to implement, due it being a built in method, BbB was implemented by following the defined formulas and using the Pytorch Gaussian classes, this was the first time such abstract math was implemented by the student and it proved challenging, but surprisingly understanding of the concepts became easier as implementation occurred, as the abstract concepts stopped being abstract.

We recognise that many improvements could be made to how our current network operates, our model is not necessarily representative of how a model would be implemented in a real world setting, usually we can combine our classifier with a network designed to segment skin lesions, which usually gives better results[1], but we did not have the time to also train a segmentation model for this set of experiments. We also could have used a larger model of EfficientNet instead of the model which used a compound scaling of 0, but given time constraints, we stuck with our current model. It would also have been nice to compare our methods to the golden standard, Deep Ensembles[21], but again, this would've taken too much time to train.

Given more knowledge of the area, there are some changes to the design that we would've made, particularly, with plottting our graphs. We primarily used matplotlib, but in reality there were some other better candidates for plotting confusion matrices, such as pycm, and for plotting reliability diagrams, such as netcal.

We show our Gantt chart in Appendix C, this was created at the start of the project, creating the Gantt chart so soon was somewhat difficult considering a lack of knowledge of the area. Given this lack of knowledge however, any immediate questions became obvious after this chart was made. We deviated a lot from this chart, as expected given how little the student knew of the area, particularly, the student underestimated the difficulty of tweaking parameters of the model and how long that would take. Besides this we performed weekly meetings to ensure that progress was on track as well as creating a TODO list at the end of every meeting to ensure we were all in agreement of what was to be done. Minutes are attached in Appendix D

8 Future Work

We have discussed some ideas for future work previously, but we now give a short recap of some areas we particularly think should be explored further.

Some form of validation of our results should be done, perhaps training multiple models and generat-

ing error bars on our coverage curves. As our results are currently, we have only trained one model, and cannot be certain whether these results will repeat as more models are trained.

We have already acknowledged why our model is highly specific to the ISIC 2019 Dataset, given different prior distribution of samples we would find different cost of results. More investigation could be done into using different population of skin lesion, as well as perhaps giving a more realistic position for this model in the diagnostic pipeline, with a more realistic cost matrix. As it stands, these results serve more as a proof of concept and we require a more specific problem, as well as some results of its performance, not on its own, but as a diagnostic tool alongside expert dermatologists.

One other area that we neglected was our ability to measure aleatoric uncertainty. Currently our model performs well on our 8 specific classes, but given out-of-sample data we could use our uncertainty metric to decide to classify as unknown. We would then also need to incorporate some cost for miss-classifying a sample as unknown.

9 Acknowledgements

This project would have not been achieved without the help of Professor Stephen Mckenna and Mr. Jacob Carse, both of which put up with months of silly questions and my boring meetings. Without their knowledge and help I would not have been able to make even a small dent into the world of AI. I would also like to thank the University of Dundee as a whole, for allowing me to make use of their GPUs which sped up the training and testing of my models significantly.

10 Appendices

10.1 Appendix A

Source code

10.2 Appendix B

Technical manual

10.3 Appendix C

Gantt chart

10.4 Appendix D

Meeting minutes

10.5 Appendix E

Result images

10.6 Appendix E

Ethics assessment

10.7 Appendix F

Mid term progress report

11 References

References

[1] Adekanmi Adegun and Serestina Viriri. Deep learning techniques for skin lesion analysis and melanoma cancer detection: a survey of state-of-the-art. *Artificial Intelligence Review*, 54(2):811–841, Feb 2021.

[2] A. S. Ahuja. The impact of artificial intelligence in medicine on the future role of the physician. *PeerJ*, 7:e7702, 2019.

[3] Aditi Singh Aryan Mobiny and Hien Van Nguyen. Risk-aware machine learning classifier for skin lesion diagnosis. *Journal of Clinical Medicine*, 8(8), 2019.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[5] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks, 2015.

[6] "Leibig C., Allken V., Ayhan M. S., Berens P., and Wahl S.". Leveraging uncertainty information from deep neural networks for disease detection. Sci Rep 7(17816), 2017.

[7] Noel Codella, David Gutman, M. Emre Celebi, Brian Helba, Michael Marchetti, Stephen Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Kumar Mishra, Harald Kittler, and Allan Halpern. Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic). 10 2017.

[8] Marc Combalia, Noel C. F. Codella, Veronica Rotemberg, Brian Helba, Veronica Vilaplana, Ofer Reiter, Cristina Carrera, Alicia Barreiro, Allan C. Halpern, Susana Puig, and Josep Malvehy. Bcn20000: Dermoscopic lesions in the wild, 2019.

[9] Terrance DeVries and Graham W. Taylor. Leveraging uncertainty estimates for predicting segmentation quality, 2018.

[10] Andre Estava, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebeastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[11] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR.

[12] Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[13] Nils Gessert, Maximilian Nielsen, Mohsin Shaikh, René Werner, and Alexander Schlaefer. Skin lesion classification using ensembles of multi-resolution efficientnets with meta data. *MethodsX*, 7:100864, Jan 2020.

[14] Kevin Gurney. *An introduction to neural networks*. UCL Press, 11 New Fetter Lane London EC4P 4EE, 1 edition, 1995.

[15] Balazs Harangi. Skin lesion classification with ensembles of deep convolutional neural networks. *Journal of Biomedical Informatics*, 86:25–32, 2018.

[16] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *CoRR*, abs/1610.02136, 2016.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[18] Mohamed A. Kassem, Khalid M. Hosny, and Mohamed M. Fouad. Skin lesions classification into eight classes for isic 2019 using deep convolutional neural network and transfer learning. *IEEE Access*, 8:114822–114832, 2020.

[19] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[20] Anders Krogh and John Hertz. A simple weight decay can improve generalization. In J. Moody, S. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1992.

[21] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles, 2017.

[22] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128(2):261–318, Feb 2020.

[23] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[24] Erdem Okur and Mehmet Turkan. A survey on automated melanoma detection. *Engineering Applications of Artificial Intelligence*, 73:50–67, 2018.

[25] Leslie N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[27] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[28] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[29] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific Data*, 5(1):180161, Aug 2018.

[30] Philipp Tschandl, Christoph Sinz, and Harald Kittler. Domain-specific classification-pretrained fully convolutional network encoders for skin lesion segmentation. *Computers in Biology and Medicine*, 104:111–116, 2019.