Stochastic Convolutions for High Resolution Medical Image Classification

E6691.2021Spring.CKRY.report
Christopher Reekie
Columbia University

Abstract

In this project a novel stochastic convolutional layer and methodology for architectural integration of the layer into existing CNN backbones is proposed for the application of high resolution image classification where the classified objects are homogeneous in nature, and fine details drive class delineation. The dataset chosen for evaluation was the ISIC skin lesion challenge dataset for the task of classifying malignant skin lesions in high resolution medical images.

Using the stochastic convolutional layer and the proposed architectural approach, a model utilizing the EfficientNet-B0 backbone ('Full Stochastic Split') achieves significant performance improvement over baseline, exceeding the higher resolution Efficienet-B4 model performance while reducing model size by 54%.

Runtime of the approach in Pytorch remained a key technical hurdle throughout the project. Optimization of the stochastic convolution operation was achieved, through iterative runtime analysis, reducing forward propagation runtime from 2.6 seconds to 24ms, though there is still room for improvement in this area.

The results demonstrate that the stochastic convolution approach scales performance more efficiently in model size than the current state of the art EfficientNet scaling approach in the chosen application of high resolution image classification .

1. Introduction

Common classification tasks involve the identification and differentiation of non-homogeneous objects. The standard benchmark for computer vision models is the ImageNet[1] dataset, which is comprised 20,000 different classes of objects which vary significantly in shape, size and representative features (e.g. dog, strawberry, car, planes etc.).

The development of many neural network architectures focus on improving performance in object classification performance on ImageNet. However, the ImageNet classification task is not wholly representative of the characteristics of many problems where CNNs can make a tangible impact.

The field of medical imaging is an area where CNNs have shown significant promise in supporting the diagnosis process where diagnosis involves the analysis of medical images.

Unlike the ImageNet computer vision task, the objects within medical images are often very homogeneous and delineation between classes (e.g. malignant or benign) can often depend only on very subtle features in the objects being studied. Similarly, medical images tend to be much higher resolution than those images used in image classification benchmarks.

It has been shown that architectures specifically designed to perform well against common CV benchmarks such as EfficientNet[2] can perform well in analyzing the malignancy of skin lesions[9].

However, the way in which these architectures are typically employed to evaluate high resolution images is to first augment the images via a center crop and resize operation to match the image resolution to that of the input dimension of the CNN.

In this project, I investigate a new approach to the Convolutional layer, integrated into existing architectures, with the goal of enhancing classification performance on high resolution medical images with homogeneous objects.

In this report, I describe and detail a new Convolutional type layer specifically designed to extract high resolution features which are then integrated into an EfficientNet backbone architecture.

This novel approach is then compared against a standard EfficientNet architecture on the ISIC lesion diagnosis dataset for the identification of malignant skin lesions from high resolution images (see figure 1).

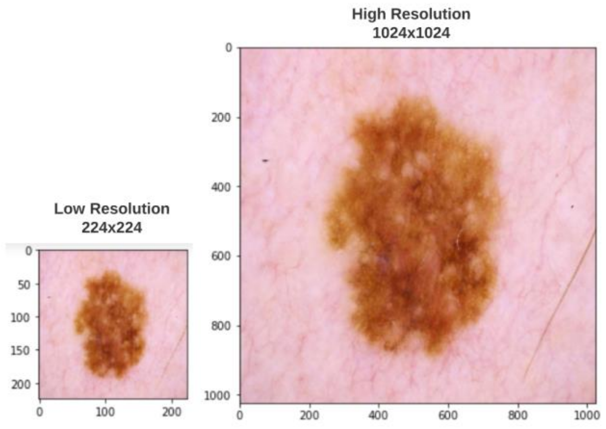


Figure 1

2. Summary of the Original Paper

The EfficientNet paper proposes a principled method for scaling CNN architectures in width, resolution and depth. The authors propose a base architecture, EfficientNet-B0, and scale this architecture up to 600x600 input resolution sizes. The largest scaled network, EfficientNet-B7, achieved state of the art performance in ImageNet classification while outperforming much larger networks.

The EfficientNet architecture serves as the backbone for the implementation of my stochastic convolution layer as such, the contributions of the original paper are detailed below.

2.1 Methodology of the Original Paper

The paper 'EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks'[2] focuses on the task of efficiently scaling CNN architectures to maximize performance whilst simultaneously minimizing the number of parameters.

The authors breakdown the scaling process of a CNN architecture into 3 dimensions: width, depth and resolution. The authors observe that scaling of CNN architectures had typically focused on only 1 of the three dimensions at a time.

The authors formulate the problem of scaling a CNN architecture as an optimization problem wherein they seek to maximize accuracy of the model by varying the 3 scaling dimensions whilst simultaneously limiting the total FLOPs and Memory of the model architecture.

The authors of the paper propose both a new baseline architecture (EfficientNet-B0) and a compound scaling approach wherein the depth, width and resolution of the base architecture are scaled in a principled manner.

The scaling approach is defined as follows; depth (α), width (β) and resolution (γ) coefficients of the baseline architecture are first determined via grid search. These coefficients remain fixed, and the baseline architecture is then scaled by increasing these dimensions by a power factor $\emptyset$ which determines resources to be allocated to each scaling dimension, where $\emptyset = 1$ is set for the baseline. The architecture dimensions are then scaled using the following:

$$depth: \ d = \alpha^{\emptyset}$$
$$width: \ w = \beta^{\emptyset}$$
$$resolution: \ r = \gamma^{\emptyset}$$

Since the FLOPS of a convolution operation is proportional to $d, w^2 \ and \ r^2$; by constraining $\alpha . \beta^2 . \gamma^2 \ \approx 2$, increasing $\emptyset$ will increase total FLOPS approximately by $(\alpha . \beta^2 . \gamma^2)^{\emptyset} \ \approx 2^{\emptyset}$. The key contribution is that this approach provides a systematic and efficient approach to increasing the respective dimensions (depth, width, height) of the model for a fixed increase in available resources $\emptyset$.

The scaling coefficients for the baseline architecture EfficientNet-B0 were determined by the authors via grid search and are provided:

$$\alpha = 1.2, \beta = 1.1, \gamma = 1.15$$

The base architecture (EffNet-B0) is composed of a stem convolution layer, followed by a series of Inverted Mobile Residual Bottleneck (MBConv) modules with Squeeze and Excitation Layers added, the activation function used is Swish (see figure 3).

The composition of an MBConv block in the EfficientNet model is as follows: The block starts with an expansion convolution, expanding the number of channels according to an expansion ratio. This is then followed by Depth Wise Convolution, Squeeze and Excitation, then finally an Output Convolution with a Residual Connection if the block is repeating and DropConnect.

Each convolution and activation operation is followed by a Batch Normalization layer. Stochastic depth is achieved via the 'Drop Connect' mechanism; in a repeating MBConv block (input channels = output channels and stride = 1) individual features in the MBConv output are randomly dropped with some probability before the residual connection. The result is stochastic depth where some features will go through fewer Inverted Bottleneck blocks than others.

A diagram of the Inverted Bottleneck block and its operations are shown below in figure 2:

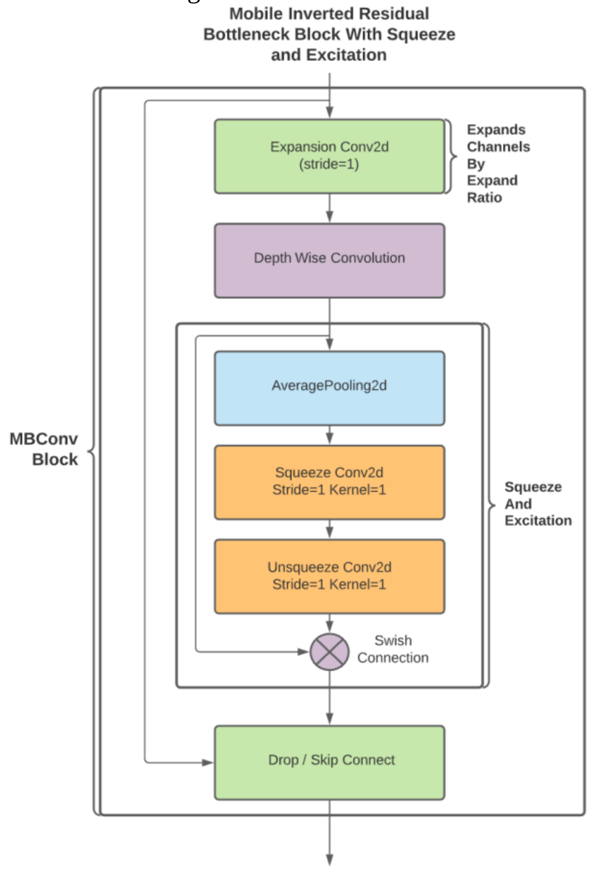

Mobile Inverted Residual Bottleneck Block With Squeeze and Excitation
Expansion Conv2d (stride=1)
Expands Channels By Expand Ratio
Depth Wise Convolution
AveragePooling2d
MBConv Block
Squeeze Conv2d Stride=1 Kernel=1
Squeeze And Excitation
Unsqueeze Conv2d Stride=1 Kernel=1
Swish Connection
Drop / Skip Connect

Figure 2

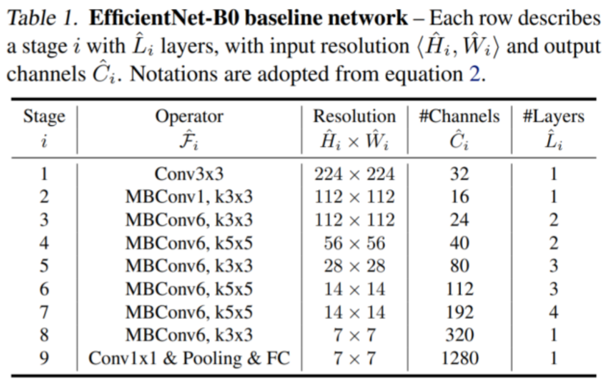


Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

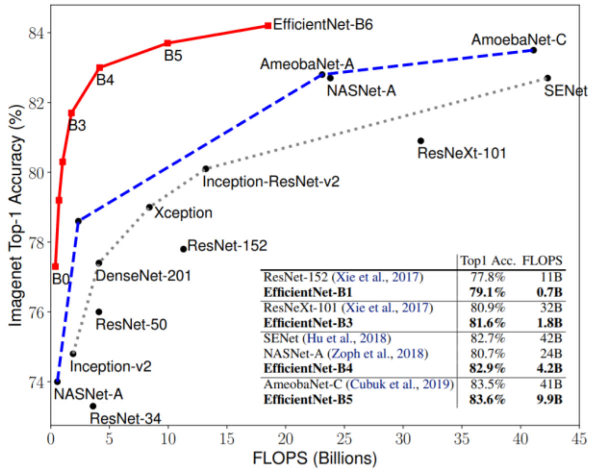
Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 3

2.2 Key Results of the Original Paper

The key contribution of the EfficientNet paper was the base architecture EfficientNet-B0 for 224x224 RGB images and architectures for higher resolutions, scaled using the B0 base architecture and the scaling approach proposed by the authors. These architectures are named B1-B7 respectively and target resolutions of 240x240 (B1) to 600x600 (B7).

These efficient architectures vastly outperform existing architectures when compared on the basis of FLOPS. The EfficientNet-B7 architecture achieved state of the art performance on the ImageNet benchmark. This is illustrated in the below graph taken from the original paper (figure 4):



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.1%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.6%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.9%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.6%**	**9.9B**

Figure 4

Due to the performance and small footprint of the EfficientNet models, I use them as the backbone for investigating the novel 'stochastic convolution' approach for high resolution images.

2.3 Review of Related Work

There is extensive research in the field of computer vision classification without direct focus on high resolution image classification, Touvron, Vedaldi et al. [5] demonstrate that current state of the art classification models tend toward higher image resolutions. The current state-of-the-art classification model in the ILSVRC 2012 is the 800x800 resolution 480MM parameter EfficientNet-L2 architecture [6]. However, the focus of these models to improve performance on standard CV benchmarks and high resolution images are generally downscaled to the target resolution.

The application explored within this paper is that of dermatological classification specifically, identifying benign and malignant skin lesions in high resolution images. The most relevant previous publication in this area is Estva et al, wherein a the Inception v3 model was trained using 129,450 images of skin lesions to classify malignancy. A transfer learning approach was used with the Inception v3 model having been first trained on the ImageNet dataset then fine tuned on the skin lesion dataset. The authors achieved an ROC AUC of 0.94 for melanomas classified exclusively with images.

This work demonstrated that the application is promising but no architectural modifications were explored for the task.

3. Methodology

The core thesis of this project is to determine if extracting fine features from different regions of high resolution, uncompressed images will provide an increase in classification performance without disproportionately increasing parameter count and FLOPs of the model.

In this report I evaluate an architectural variation on the convolutional layer against the task of medical image classification. The medical images are homogeneous and high resolution.

The ISIC melanoma dataset [10 – 17] was chosen due to the high resolution of base images, the homogeneity in captured objects and the nature of the classification task wherein fine features (e.g. edges of moles) may drive class differentiation.

To evaluate the efficacy of this approach, I reconstruct the EfficientNet architectures (B0 and B4). The base models, without modification, are used as benchmarks with which to evaluate the performance of the augmented approaches.

The 'stochastic convolution layer' is then integrated into the base architecture and then the performance of the augmented models are compared against the base model to determine if the layer provides an uplift in performance.

3.1. Objectives and Technical Challenges

3.1.1 Objectives

The project was broken down into the following steps:

Data Gathering & Preparation

High resolution images of benign and malignant skin lesions were gathered from the ISIC 2016, 2017, 2018, 2019 & 2020 Challenge competitions. Preprocessing steps are outlined in the implementation step.

Reproduction of the EfficientNet-B0 and B4 Architectures

The EfficientNet-B0 and EfficientNet-B4 architectures were recreated using the Pytorch API to serve as a backbone for experimentation.

Development of the Stochastic Convolution Layer and Optimization

The custom 'stochastic convolution' layer was programmed using the Pytorch API. Optimization remained a large technical challenge, many implementations were tested to improve efficiency.

Integration and Evaluation of Stochastic Layer Embedded Within the EfficientNet backbones

Three new architectures are proposed which integrate the stochastic convolutional layer into the existing EfficientNet backbone. The new architectures were evaluated against the baseline model approach.

3.1.1 Technical Challenges

The key challenge in the implementation and evaluation of the novel stochastic convolution layer type was in optimization using the Pytorch framework.

Given the unique nature of the operation relative to existing functions in the high level Pytorch API, the runtime overhead of the layer was significant relative to a standard convolutional layer. This necessarily inhibited the number of experiments that could be performed and the amount of analysis.

Multiple implementations and variations on the approach were used in order to optimize the approach. A detailed discussion of the implementation challenge and constraints are discussed in the implementation section.

3.2. Problem Formulation and Design Description

In this section the 'stochastic' convolution layer is defined and the method for its application is defined.

3.2.1 Stochastic Convolutional Layer

In a typical 2d convolutional layer, the layer's kernels are convolved over the input image. The stride of the convolution defines the steps with which the convolutional kernel is passed over the input image to produce an output with a given resolution. The number of kernels dictate the number of output channels.

The below figure illustrates this process for 3 output channels from a convolution layer. The colored boxes illustrate the region over which the kernels for each output channel in the convolution layer are applied to a low-resolution downscaled image of a malignant lesion (224x224). The overlapping regions cover the whole image (see figure 5):

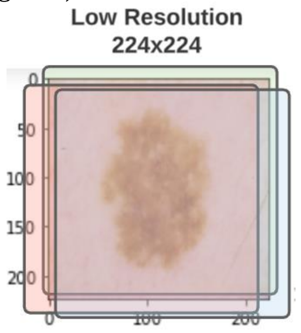


Figure 5

The key differentiating component between the stochastic convolution, as implemented in this paper, and the standard convolution is that in the stochastic convolution, the filters are convolved over only a random portion of a high-resolution, uncropped image to produce an output of fixed dimension. This process is illustrated visually in figure 6.

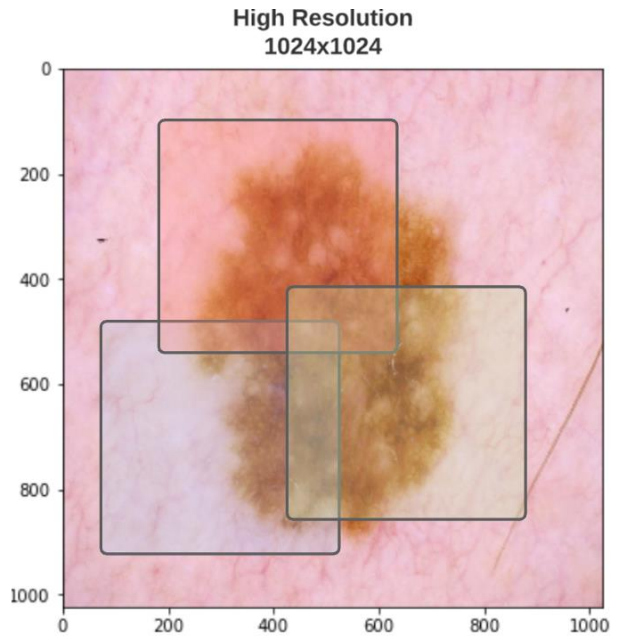


Figure 6

The dimensions of the regions over which the 'stochastic' convolution is applied are chosen to produce a specific output resolution.

Formulaically. defining the convolution function (over a single channel image) is given by:

$$F * I(x, y) = \sum_{j=-N}^{N} \sum_{i=-N}^{N} F(i, j) I(x - i, y - j)$$

In strided convolution, the above formula is applied to the whole input image. The filter is moved over the input image according with pixel steps equal to the stride.

In the stochastic convolutional layer, for each output channel, a random region of the image is selected, then strided convolution is applied to that random region without padding using the 3 RGB filters corresponding to each output channel.

The dimensions of the region of the high resolution image over which the convolution is applied is calculated with the following formulas:

$$Slice\ Height = (Output\ Height - 1) * stride + kernel\ height$$
$$Slice\ Width = (Output\ Width - 1) * stride + kernel\ width$$

Regions of the high resolution image were randomly extracted by sampling from a random uniform distribution over the selectable region according to the input dimensions and slice dimensions according to the following formulae:

$$HeightEnd \sim Uniform(Slice\ Height, Image\ Height)$$
$$WidthEnd \sim Uniform(Slice\ Width, Image\ width)$$

$$HeightStart = HeightEnd + Slice\ Height$$
$$WidthStart\ = WidthEnd + Slice\ Width$$

$$Slice = Img[:, HeightStart: HeightEnd, WidthStart: WidthEnd]$$

Sampling using a uniform distribution starting from the required slice dimension ensured that any random region of the image could be randomly extracted.

Stochastic Convolution Pseudo Code:

The pseudo code for the stochastic convolution operation as defined is given below along with the final implementation used within this project to address optimization issues:

Stochastic Convolution by Image:

```
For each image in the minibatch $image \in MB$:
    For each 3 dimensional $kernel \in output\ channels$:
        Randomly subset image $random\ image\ slice \in image$:
            Output[img,k] = $Conv2d(kernel,\ random\ image\ slice)$
```

The above implementation was the intended approach for the project; a random slice of the high image is sampled using a uniform distribution for **each set of RGB kernels** and **each image**.

Due to issues with the lack of high level Pytorch optimization for the Conv2d API, the approach was adjusted to randomly slice the same area of **all images** in the batch for **the 3-dim kernel corresponding to each output channel**.

This approach is outlined in the pseudocode below:

Stochastic Convolution by Mini-Batch:

```
For each 3 dimensional $kernel \in output\ channels$:
    Randomly slice full image batch $random\ slice \in MB$:
        Output[:,k] = $Conv2d(kernel,\ random\ batch\ slice)$
```

The stochastic convolution by batch differs from the convolution by image in that it randomly slices the same region of all images in the minibatch over which the kernel is convolved. Due to the data augmentation approach where images were randomly rotated and scaled, this ensured that the kernel was still applied to different regions per image. This approach was substantially faster (by a factor of over 100x) and did not materially affect the performance of the trained models.

4. Implementation

Within this section, I describe the base model architecture and the variations of the base model architecture with integrated stochastic convolutions. I then describe the data sourcing and preprocessing process and finally the evaluation process for the new architecture.

4.1. Deep Learning Network

4.1.1 Base EfficientNet-B0 & B4 Architecture

The EfficientNet-B0 architecture is a convolutional neural network. It is comprised of a convolution stem which processes the input image to produce feature maps of resolution 112x112x32 from a 224x224 RGB image. These feature maps are then processed by a series of Mobile Residual Inverted Bottleneck blocks (as described in section x). The output features from these blocks are then flattened in the head, dropout is applied which is then followed by a fully connected layer to the output.

This architecture is illustrated in figure 7 below:

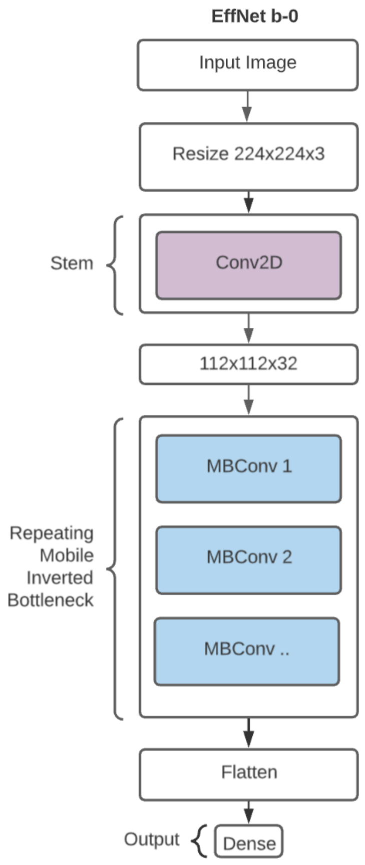


Figure 7

The EfficientNet-B4 is of similar design, with the resolution, width and number of blocks increased according to the following scaling parameters xx. The target resolution of EfficientNet-B4 is 380x380.

4.1.2 Stochastic Stem Architecture

The first architectural variation of the EfficientNet architecture, integrates the stochastic convolutional layer into the stem of the model. The image input process is bifurcated in the following manner: the input image is downsized to the target size of the model (224x224x3), this is passed directly to the stem of the model to produce 32 'global image' feature maps of dimension 112x112. The stochastic convolutional layer also acts like a stem to the architecture; the stochastic convolution is applied over the original input image without resize operation to produce 32 'high resolution' feature maps of size 112x112.

The 'high resolution' and 'global' feature maps from both standard and stochastic stems are then concatenated and passed on to the rest of the architecture. The same kernel size was used for the stochastic stem as the base stem (3x3, stride = 2 for B0 and B4).

This is illustrated in figure 8 below:

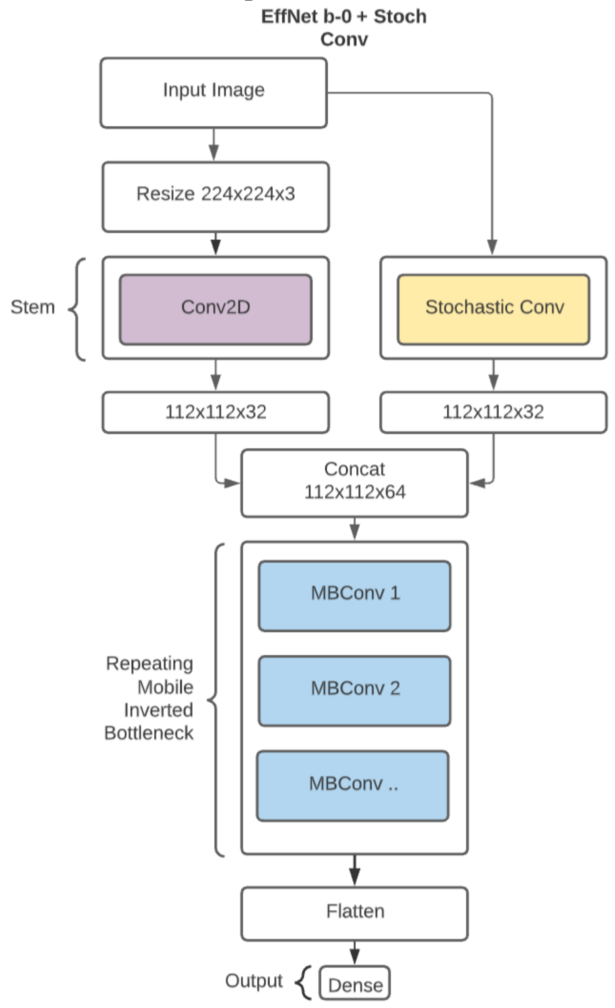


Figure 8

The idea here is that the model may be able to make better predictions by combining high resolution random features from the image with the low resolution features extracted from applying the base convolution to the full downsized image.

4.1.3 Stochastic Skip Connect Architecture

The second architectural variation on the EfficientNet backbone explored in this project integrates the stochastic convolution stem by bifurcating the first few Mobile Inverted Bottleneck blocks between the standard stem and the stochastic stem. Instead of concatenating the output of the stochastic and standard stem, instead both are passed through a series of independent but identical blocks before being concatenated at a higher level in the architecture, this is illustrated in figure 9 below:

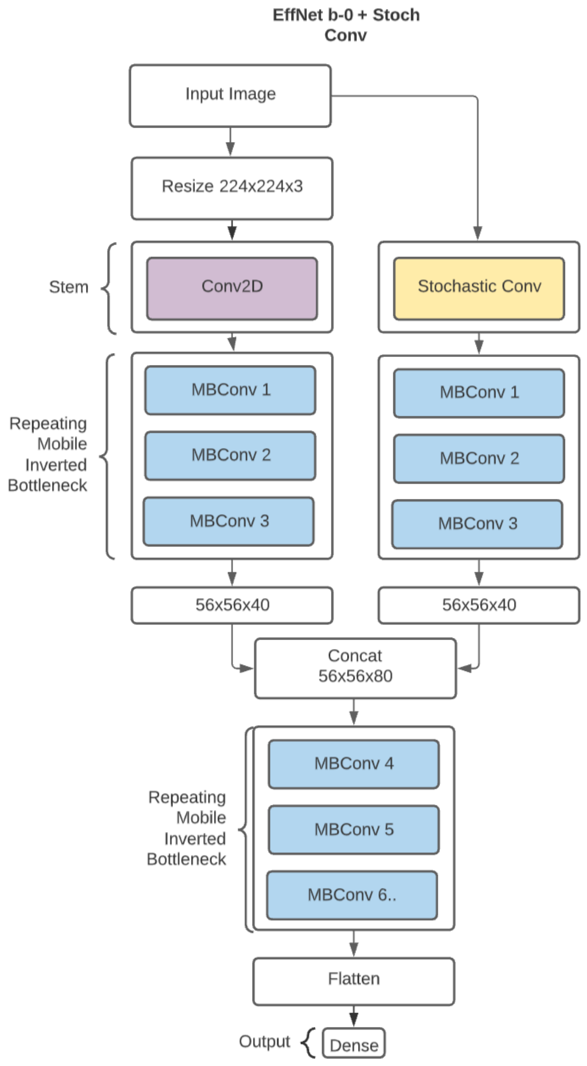


Figure 9

The thought process behind this design is that high resolution features may require some layers of processing before their representations can usefully be concatenated. By bifurcating the model in the first few layers, the model may learn some specialization in feature extraction from the low-resolution features and the high resolution local features.

4.1.4 Full Split Architecture

Following experiments from the stochastic skip and stochastic stem architecture. It was theorized that combining the high resolution randomly extracted feature maps with the lower resolution feature maps would be better achieved using a fully connected layer, rather than a convolutional layer. Consequently, the architecture shown in figure 10 is suggested.

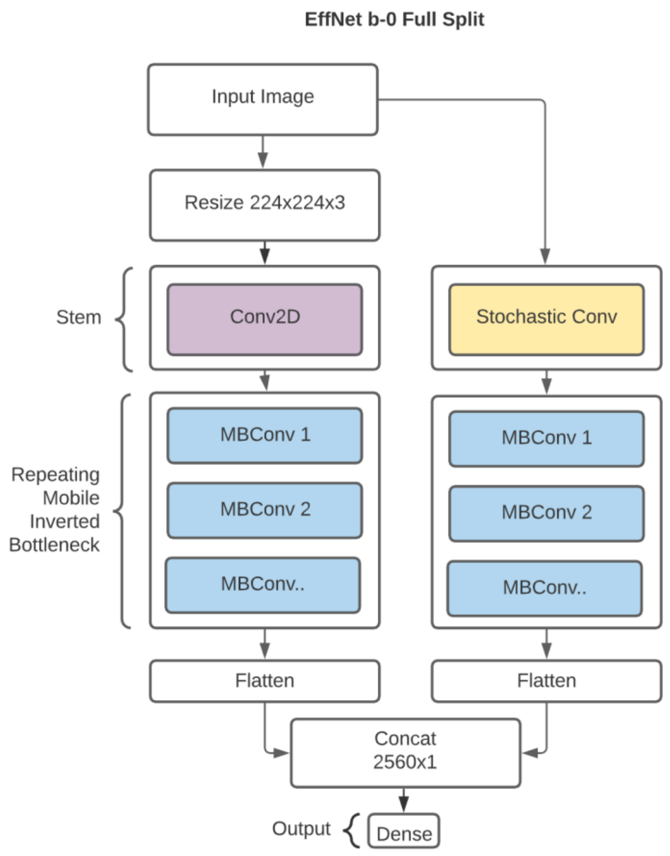


Figure 10

Unlike the stochastic skip and stochastic stem architectures, the full split architecture has a dedicated backbone for both the stochastic high resolution features and the global low resolution features. Both features go through a series of mobile inverted bottleneck blocks (modelling the B0 backbone) and are combined at the head in a fully connected layer.

The general thought process here is that by using a fully connected layer instead of convolution to integrate the high resolution feature maps, that model will be able to decide which of the high resolution features it thinks is useful and discard information that it is not. This process is better achieved in a fully connected layer than it would be in convolution.

4.1.5 Data

The data used for the project was sourced from the ISIC 2016, 2017, 2018, 2019 & 2020 [10][11][12][13][14][15][16][17] malignant lesion classification competition datasets. These datasets contain high resolution images of benign and malignant skin lesions.

In total, 72,000 images were gathered. All images in the dataset that were resolution lower than 1000x1000 were dropped from the training data, resulting in a training set of size 47,557 high resolution images of which 12,069 contained malignant lesions.

The high-resolution images were preprocessed in the following manner; if the resolution of the image was greater than 2048x2048, the image was center cropped to 1024x1024.

All images (including center cropped) images were then resized to the target high resolution image size to be used by the model. In the investigative phase of the project, 16,000 images were used for training and 4,000 images were used for validation.

To evaluate the final models, an 80%/20% split of the full 47,557 image dataset was used.

4.1.6 Data Augmentation

Identical image augmentations were used to train and evaluate all models. The images were randomly rotated through 180 degrees, randomly scaled between 60% and 100% and randomly flipped on the vertical and horizontal axis.

Augmentation was randomly applied to both the training and validation data in the evaluation process.

The custom dataset class and augmentation routine can be found in the 'utils.py' file. The data preprocessing steps and code can be found within the 'Data Preparation' and 'Resize Images' jupyter notebooks.

It is important that identical augmentation be applied to both the high and the low resolution image. Failure to do so results in the model not achieving good performance. An illustration of the augmented high and low resolution images produced by the custom data loader is shown in figure 11 below:

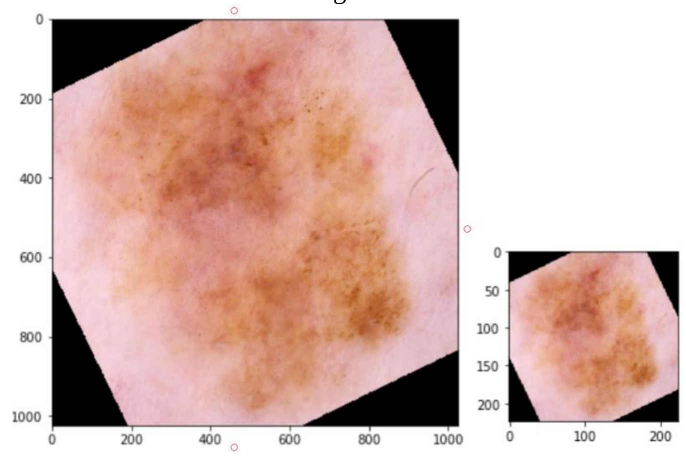


Figure 11

4.1.7 Model Evaluation Metric

The metric chosen for evaluation of the base and proposed model was the Area Under the Receiver Operation Characteristic Curve (ROC AUC). The ROC curve charts the True Positive Rate (TPR) against the False Positive Rate (FPR). This is a common scoring metric for medical classification tasks as it allows the practitioner to balance the false positive rate of the model vs the true positive rate. This is particularly important in medical applications where a false negative prediction may have significantly worse outcome than a false positive prediction.

The ROC curve facilitates the evaluation of models at different thresholds of TPR and FPR. The area under the ROC curve provides a single quantitative metric that captures the desired quality of the model (high TPR and low FP rate). Therefore, a high ROC AUC curve is desirable and is the metric used to compare models. The formula and a visual illustration [4] is provided in figure 13.

$$True\ Positive\ Rate\ (TPR) = \frac{True\ Positive}{True\ Positive\ + False\ Negative}$$

$$False\ Positive\ Rate\ (TPR) = \frac{False\ Positive}{False\ Positive\ + True\ Negative}$$

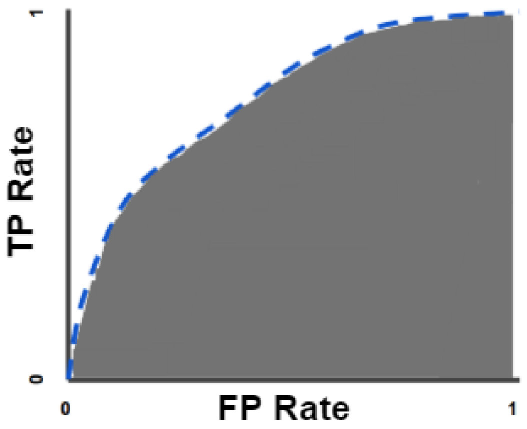


Figure 12

4.1.8 Model Evaluation Process

All models were trained with the same optimizer, using the same training and validation data with the same batch size. Due to the stochastic nature of model training and convergence, all models were trained a minimum of 3 times over which the performance was evaluated to ensure that the results of the experiment were valid and one-offs didn't overly influence final scores.

To the extent possible, and within compute constraints, models were trained for a target of 5 times each, for 60 epochs each time.

4.2. Software Design

In the section below, I describe the structure of the EfficientNet Pytorch implementation and the components of the Mobile Inverted Residual blocks. I then describe the various implementations of the stochastic convolutional layer that were tested to find the most optimal solution. Finally, I cover the integration of the EfficientNet model into the backbone.

4.2.1 EfficientNet Base Architecture

The EfficientNet backbone model used as the basis for experimentation was constructed in Pytorch. EfficientNet-B0 and B4 backbones were explicitly defined rather than building out a scaling architectural approach.

Separate classes for each model architecture were constructed as derived classes of Pytorch's 'Module' class. These can be found within the 'efficientnet_model.py' file.

Different models were written for experimentation, however the key models for the EfficientNet base architectures and the experimental stochastic architectures. The suffix 'Stochastic' indicates that the stochastic convolution was integrated into the stem of the architecture and the suffix 'SkipStochastic' indicates that the stochastic layer was implemented in the manner described in section 4.1.3. The suffix 'StochasticSplit' corresponds to the architecture described in 4.1.4.

The **EffNet** class takes the input image resolution for the low resolution image over which the 'global features' are extracted, and resolution for the high resolution image over which the stochastic convolutional layer is applied. The user can also specify the dropout rate for the model top to the constructor.

Each of the model classes is comprised of a model stem, repeating inverted mobile residual blocks and a model top. The architectures of the various model backbones are defined in dictionaries in the 'model_architectures.py' file.

The backbone architecture of the model is constructed dynamically by creating instances of the InvertedMobileResidualBlock class using the specification in the corresponding backbone dictionary. The top of the architecture is a convolutional layer, followed by average pooling, dropout and a dense layer to output.

4.2.2 Custom Layers and Inverted Mobile Residual Block

The EfficientNet architecture is comprised of a stem, a head and a series of Inverted Mobile Residual Bottleneck blocks. The residual bottleneck blocks are all comprised of an optional expansion layer, followed by a depthwise separable convolution layer then a squeeze and excitation layer followed by a standard convolutional layer for the output (see figure x). Skip and drop connect is then applied in cases where there are repeating blocks.

A dedicated class was written for the residual blocks 'InvertedMobileResidualBlock' as a derived class of the Pytorch module class.

The individual layers that comprise the residual blocks were themselves written as independent derived classes. These classes can be found in the 'layers.py' file.

The 'DropConnect' class handles stochastic depth through repeating blocks. The 'ExpandLayer' class handles convolutional expansion at the beginning of the block. The 'SqueezeExcitationLayer' class provides squeeze convolution and un-squeezed convolution functionality for the squeeze and excitation layer. The 'DepthWiseConv' class is used for depth wise separable convolution within the block.

Due to the lack of 'same' padding for convolutions in the Pytorch framework, a dedicated class was created for same padding 'Convolution2dSamePadding', the constructor for this class requires the input dimension size to determine the padding required for 'same' padding.

Detailed description of each class and the operations within each function can be found within the code.

An example of construction and training of the models is provided in TrainScript.py.

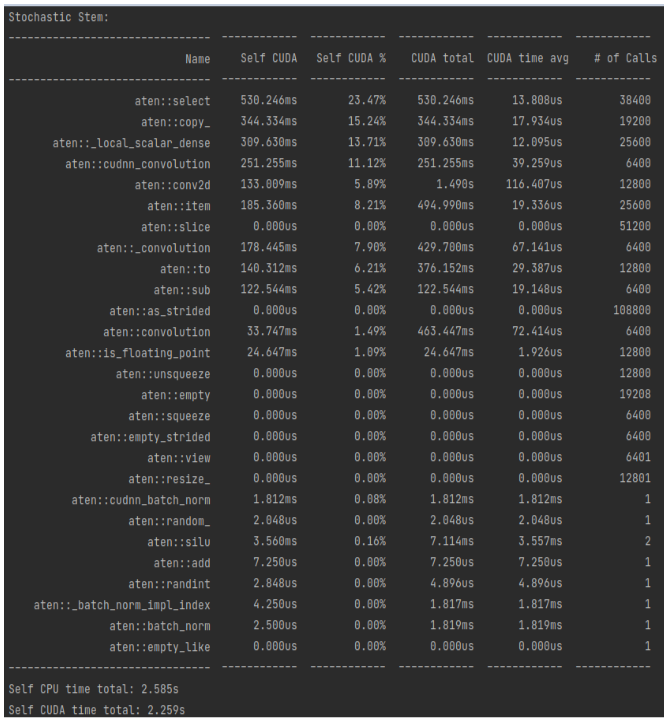
4.2.3 Stochastic Convolution & Optimization

Implementing the stochastic convolution in the Pytorch framework in an efficient manner was the most challenging aspect of the project. Standard deep learning functions like Convolution and dense operations have been optimized in low level cuda C++ kernels to avoid any copy operations which generally take the majority of overhead of compute time.

Since the stochastic convolution applies a 2d convolution over only a subset of the high resolution image, and not the whole image, a custom implementation was written to support this functionality.

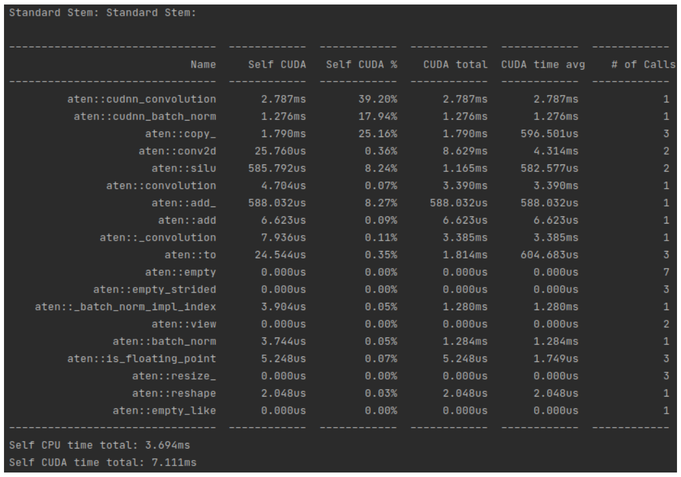
The main challenge developing the stochastic convolution function using the Pytorch API was in reducing the total on device copy operations.

The initial naïve approach of applying manual convolution to random slices of every image in the whole batch for each kernel in the layer (see ‘StochasticConv1’ in ‘layers.py’) resulted in forward runtime on the EffNetB0 architecture of 2.6 seconds using a batch of 200 images see figure 13. A standard convolutional stem took 8ms on the same hardware (see figure 14).



```
Stochastic Stem:
-------------------------------  ------------  ------------  ------------  ------------  ------------
                           Name     Self CUDA   Self CUDA %    CUDA total  CUDA time avg    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------
                   aten::select     530.246ms        23.47%     530.246ms      13.808us         38400
                    aten::copy_     344.334ms        15.24%     344.334ms      17.934us         19200
      aten::_local_scalar_dense     309.630ms        13.71%     309.630ms      12.095us         25600
        aten::cudnn_convolution     251.255ms        11.12%     251.255ms      39.259us          6400
                   aten::conv2d     133.009ms         5.89%        1.490s     116.407us         12800
                     aten::item     185.360ms         8.21%     494.990ms      19.336us         25600
                    aten::slice       0.000us         0.00%       0.000us       0.000us         51200
             aten::_convolution     178.445ms         7.90%     429.700ms      67.141us          6400
                       aten::to     140.312ms         6.21%     376.152ms      29.387us         12800
                      aten::sub     122.544ms         5.42%     122.544ms      19.148us          6400
               aten::as_strided       0.000us         0.00%       0.000us       0.000us        108800
              aten::convolution      33.747ms         1.49%     463.447ms      72.414us          6400
        aten::is_floating_point      24.647ms         1.09%      24.647ms       1.926us         12800
                aten::unsqueeze       0.000us         0.00%       0.000us       0.000us         12800
                    aten::empty       0.000us         0.00%       0.000us       0.000us         19208
                  aten::squeeze       0.000us         0.00%       0.000us       0.000us          6400
            aten::empty_strided       0.000us         0.00%       0.000us       0.000us          6400
                     aten::view       0.000us         0.00%       0.000us       0.000us          6401
                  aten::resize_       0.000us         0.00%       0.000us       0.000us         12801
        aten::cudnn_batch_norm        1.812ms         0.08%       1.812ms       1.812ms             1
                  aten::random_       2.048us         0.00%       2.048us       2.048us             1
                     aten::silu       3.560ms         0.16%       7.114ms       3.557ms             2
                      aten::add       7.250us         0.00%       7.250us       7.250us             1
                  aten::randint       2.848us         0.00%       4.896us       4.896us             1
    aten::_batch_norm_impl_index       4.250us         0.00%       1.817ms       1.817ms             1
               aten::batch_norm       2.500us         0.00%       1.819ms       1.819ms             1
               aten::empty_like       0.000us         0.00%       0.000us       0.000us             1
-------------------------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 2.585s
Self CUDA time total: 2.259s
```

Figure 13



```
Standard Stem: Standard Stem:

-------------------------------  ------------  ------------  ------------  ------------  ------------
                           Name     Self CUDA   Self CUDA %    CUDA total  CUDA time avg    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------
        aten::cudnn_convolution       2.787ms        39.20%       2.787ms       2.787ms             1
         aten::cudnn_batch_norm       1.276ms        17.94%       1.276ms       1.276ms             1
                    aten::copy_       1.790ms        25.16%       1.790ms     596.501us             3
                   aten::conv2d      25.760us         0.36%       8.629ms       4.314ms             2
                     aten::silu     585.792us         8.24%       1.165ms     582.577us             2
              aten::convolution       4.704us         0.07%       3.390ms       3.390ms             1
                     aten::add_     588.032us         8.27%     588.032us     588.032us             1
                      aten::add       6.623us         0.09%       6.623us       6.623us             1
             aten::_convolution       7.936us         0.11%       3.385ms       3.385ms             1
                       aten::to      24.544us         0.35%       1.814ms     604.683us             3
                    aten::empty       0.000us         0.00%       0.000us       0.000us             7
            aten::empty_strided       0.000us         0.00%       0.000us       0.000us             3
    aten::_batch_norm_impl_index       3.904us         0.05%       1.280ms       1.280ms             1
                     aten::view       0.000us         0.00%       0.000us       0.000us             2
               aten::batch_norm       3.744us         0.05%       1.284ms       1.284ms             1
        aten::is_floating_point       5.248us         0.07%       5.248us       1.749us             3
                  aten::resize_       0.000us         0.00%       0.000us       0.000us             3
                  aten::reshape       2.048us         0.03%       2.048us       2.048us             1
               aten::empty_like       0.000us         0.00%       0.000us       0.000us             1
-------------------------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 3.694ms
Self CUDA time total: 7.111ms
```

Figure 14

In the standard convolutional stem the majority of compute time is spent on the convolutional operation, in the naïve approach the majority of time is spent on copy operations.

This is because the standard ‘Conv2D’ operation in Pytorch is written in a C++ cuda kernel where in pointers are used to avoid copy ops in the implementation.

Since the stochastic convolution applies the convolution operation to different slices of the image for each kernel in the layer, slicing the image in Pytorch and passing the slice & kernel weight to the Conv2D method results in unnecessary copy operations (including a copy operation to the output tensor).

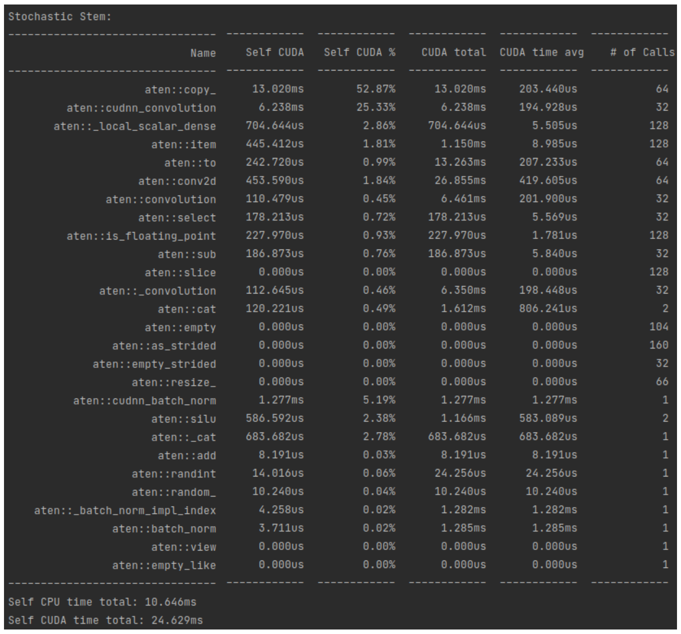
A cuda kernel level optimized approach to the stochastic convolution operation to run in equivalent time to the standard convolution using pointers and references was investigated, but optimization at this level was not possible in the time constraints of this project and the nature of the automatic gradient calculation embedded in the Pytorch framework.

Through an iterative process of cuda runtime analysis, 4 different stochastic convolution implementations were tested (StochasticConv1 through 4) and the most optimized implementation was used in experiments (StochasticConv4).

The most optimized approach (‘StochasticConv4’) implementation slices the same region of all images in the batch per kernel, rather than randomly sampling regions for each image to avoid copies. Random augmentation to the underlying images were performed prior to inference to ensure that each kernel was applied to different regions of the image.

Additionally, the implementation avoids GPU copy operations to the output by instead concatenating output from multiple Pytorch nn.Conv2D modules, making use of some embedded optimizations within those modules and constructs all tensors on the GPU.

The result of the aforementioned optimizations was an improvement in forward prop runtime from 2.6 seconds in the naïve approach to 24ms in the ‘StochasticConv4’ approach. On average, the layer is 3x slower than a standard convolution operation. This is illustrated in the figure 15.



```
Stochastic Stem:
-------------------------------  ------------  ------------  ------------  ------------  ------------
                           Name     Self CUDA   Self CUDA %    CUDA total  CUDA time avg    # of Calls
-------------------------------  ------------  ------------  ------------  ------------  ------------
                  aten::copy_      13.020ms        52.87%      13.020ms     203.440us            64
      aten::cudnn_convolution       6.238ms        25.33%       6.238ms     194.928us            32
    aten::_local_scalar_dense     704.644us         2.86%     704.644us       5.505us           128
                   aten::item     445.412us         1.81%       1.150ms       8.985us           128
                     aten::to     242.720us         0.99%      13.263ms     207.233us            64
                 aten::conv2d     453.590us         1.84%      26.855ms     419.605us            64
            aten::convolution     110.479us         0.45%       6.461ms     201.900us            32
                 aten::select     178.213us         0.72%     178.213us       5.569us            32
      aten::is_floating_point     227.970us         0.93%     227.970us       1.781us           128
                    aten::sub     186.873us         0.76%     186.873us       5.840us            32
                  aten::slice       0.000us         0.00%       0.000us       0.000us           128
           aten::_convolution     112.645us         0.46%       6.350ms     198.448us            32
                    aten::cat     120.221us         0.49%       1.612ms     806.241us             2
                  aten::empty       0.000us         0.00%       0.000us       0.000us           104
             aten::as_strided       0.000us         0.00%       0.000us       0.000us           160
          aten::empty_strided       0.000us         0.00%       0.000us       0.000us            32
                aten::resize_       0.000us         0.00%       0.000us       0.000us            66
       aten::cudnn_batch_norm       1.277ms         5.19%       1.277ms       1.277ms             1
                   aten::silu     586.592us         2.38%       1.166ms     583.089us             2
                   aten::_cat     683.682us         2.78%     683.682us     683.682us             1
                    aten::add       8.191us         0.03%       8.191us       8.191us             1
                aten::randint      14.016us         0.06%      24.256us      24.256us             1
                aten::random_      10.240us         0.04%      10.240us      10.240us             1
  aten::_batch_norm_impl_index       4.258us         0.02%       1.282ms       1.282ms             1
             aten::batch_norm       3.711us         0.02%       1.285ms       1.285ms             1
                   aten::view       0.000us         0.00%       0.000us       0.000us             1
             aten::empty_like       0.000us         0.00%       0.000us       0.000us             1
-------------------------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 10.646ms
Self CUDA time total: 24.629ms
```

Figure 15

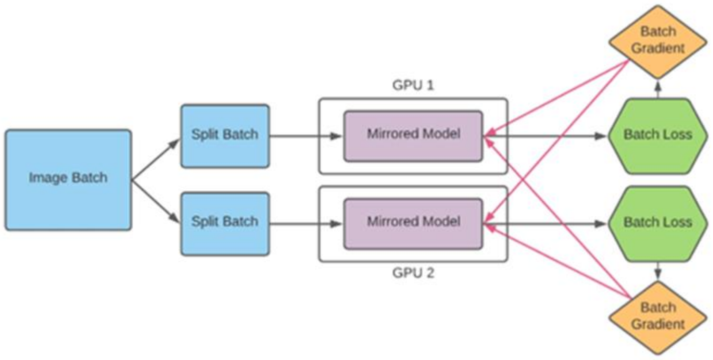


Figure 16

4.2.4 Training Routine

The training routine can be found within the 'TrainScript.py'. To ensure the validity of experiments, random seeds are set to ensure that the training datasets remain identical between model evaluation runs.

To evaluate the performance of the models and rule out 'one off' good initializations from affecting the objective evaluation of the architecture, each model was trained multiple times using different random initializations and the same optimizer. The results of each training run were then averaged and used for comparison. Due to the long training times sample sizes were small, with 5 runs performed for each model.

The training routine was written to allow both full (float32) and mixed (float16) precision using Pytorch's amp.autocast() functionality which automatically converts the model and backprop to mixed precision.

Evaluation of both methods were performed and no significant difference in performance was identified between the two methods so mixed precision was used for training and evaluating all models due to the improved runtime and lower on device memory requirements.

Distributed training using Pytorch's distributed data parallel was used to distribute training. In the distributed training routine, each device maintains an identical copy of the full model. The total batch is split between the two devices, forward inference and loss calculation is performed independently by both devices then aggregated and identical weight updates are made to both copies of the model on each device. An illustration of this process is provided in figure 16. All models were trained using 2 RTX Titans.

5. Results

5.1. Project Results

In this section, I provide an analysis of the results obtained throughout the course of the project. All of the models were evaluated using the same melanoma dataset to ensure a comparison on an 'apples to apples' basis. The following section contains the following analysis:

- Validation of the project implementation of the EfficientNet B0 and B4 architectures using the official Tensorflow implementation[3].
- Evaluation of the stochastic stem architecture using both the EfficientNet B0 and B4 backbones.
- Evaluation of the stochastic skip connect architecture using the EfficientNet B0 and B4 backbones.

The melanoma dataset was used as benchmark to compare the EfficientNet backbone written in this project and the official implementation due to the time and compute requirement of training on the ImageNet dataset. The official Tensorflow implementation of the EfficientNet architecture was used only to validate the Pytorch implementation. All subsequent architectures were evaluated to the Pytorch project baseline B0 and B4 models.

All performance graphs demonstrate the maximum, minimum and average (solid line) ROC AUC of each model architecture after multiple training passes (3-5).

5.1.1 Recreating EfficientNet Architecture

To validate the implementation of EfficientNet, both the Pytorch implementation created in this project and the official implementation via the Keras API [3] was trained using the aggregated melanoma dataset and the same optimizer (RSMProp learning rate = 0.001, alpha = 0.99, weight decay = 0 and momentum = 0). Exponential learning rate reduction was applied with a decay rate of 0.9 with warmup of 15 epochs. No pretrained weights were used.

Both models were trained 5 times each using identical batch sizes and the same training data. The average results between the project implementation and the official were then compared for parity, the average loss per batch is shown in figure 17, the training and validation set ROC AUC is shown in figure 18.

The training time for the base EfficientNet-B0 model was 90 seconds per epoch and 240 seconds for the Stochastic implementation for a total training time of 1.5 hours for the base model and 4 hours for the stochastic model for each training run.

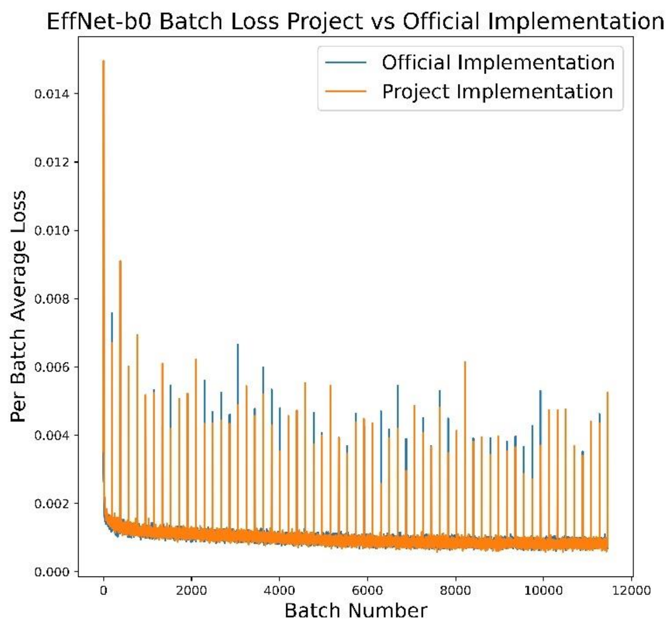


Figure 17

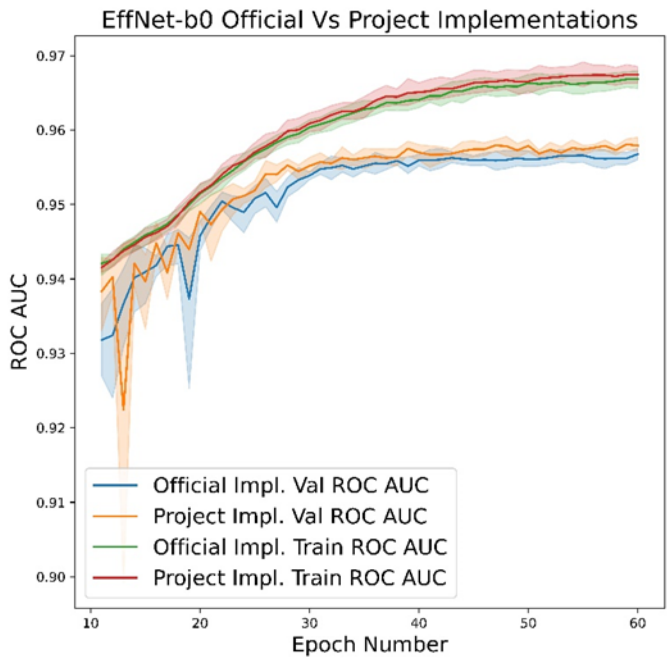


Figure 18

Observing the above graphs, it is clear that after averaging the results over 5 separate training runs for each model, that there is no discernable difference in the EfficientNet model implemented in this project and in the official implementation. Early batch loss is somewhat divergent for the project implementation, though this is very likely due to the difference in intialization scheme between the two methods.

The EfficientNet-B4 architecture was also constructed to test the convolutional layer in a higher capacity model. The B4 model contains around ~17.5MM parameters, in contrast to 4.5MM of the B0 model. Validation and training ROC AUC graphs were also compared to the official implementation and are shown in figure 19 below.

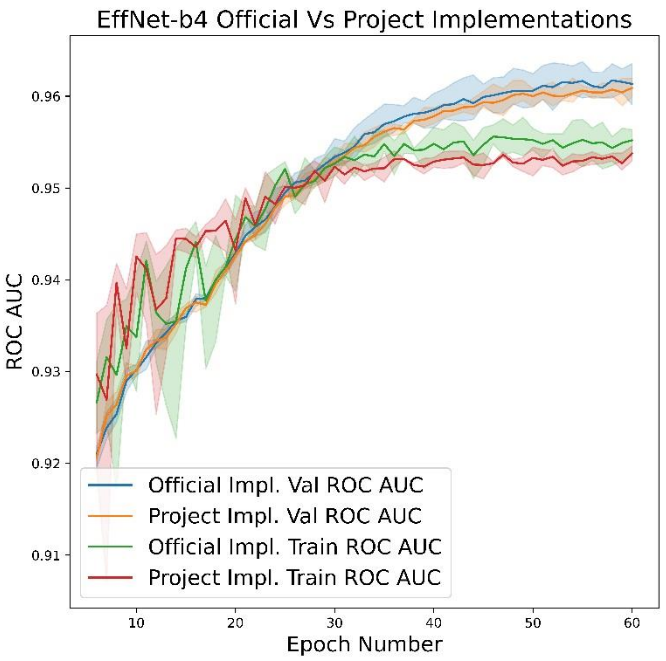


Figure 19

In a similar vein to the B0 implementation, on average the implementation of the B4 architecture in this project and that of the official tensorflow implementation are very similar, with differences likely driven different random initialization schemes for the starting weights.

5.1.2 Stochastic Stem Architecture Model

5.1.2.1 Stochastic Stem EfficientNet-B0

The model implementation with the stochastic convolution integrated into the stem of the architecture (as detailed in section 4.1.2) was evaluated on the same sample of training data, using the same optimizer and parameters as with the evaluation of the base models.

The stochastic stem was applied to high resolution images of 1024x1024 and the base stem was applied to the same downsized image at a resolution of 224x224. The stochastic stem produced 32 output channels that were concatenated with the 32 output channels from the base for a total of 64 output features from the combined stems. This model was compared against the base EffNet-B0 architecture with 32 output channels from the base stem. The training and validation set RoC AUC were averaged over 5 runs and are displayed in figure 20 below.

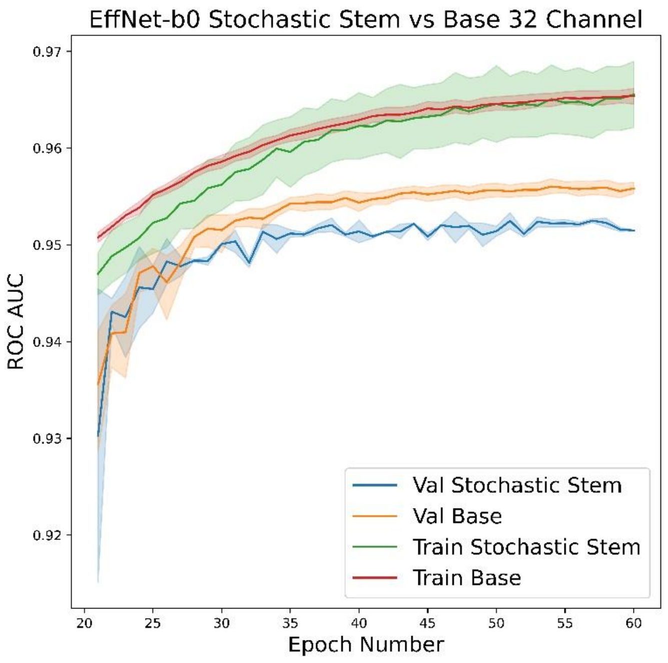


Figure 20

Comparing the stochastic stem approach with base model, the ROC AUC score for the stochastic approach on the training dataset achieves equal performance to the base model in later epochs (30+). However, the ROC AUC score for the stochastic stem approach falls short of the base model performance on the unseen validation data.

The stochastic approach was compared against a base EffNet-B0 model with increased number of output channels from the stem (64), to compare models on an equal FLOP basis, the results are shown in the figure 21 below:

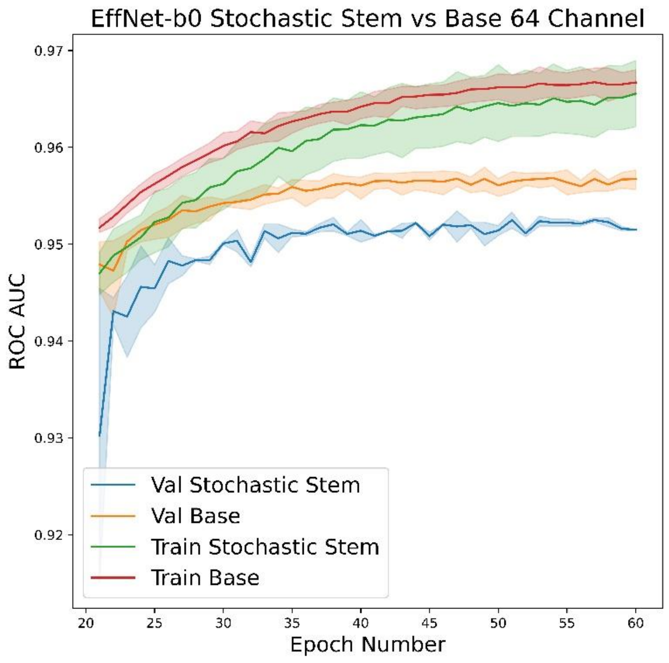


Figure 21

The results mirror those of those when compared against the 32 channel input model; the stochastic approach achieves almost parity performance with the base architecture, but falls short on unseen data.

The architectural design of the B0 architecture reduces the number of feature maps from the stem from 32 to 16 channels after the first mobile inverted bottleneck block. It's possible that by adding more feature maps to the stem, that the architecture is impeded because it needs to incorporate both the stochastic feature maps and the global feature maps into 16 output feature maps and the high resolution feature maps may not always be predictive.

5.1.2.2 Stochastic Stem EfficientNet-B4

To evaluate whether the capacity of the base model (B0) is a limiting factor in incorporating the stochastic feature maps, I attempt the same approach with the EfficientNet B4 architecture. In contrast to EffNet-B0's ~4.5MM parameters, EffNet-B4 has ~17.5MM parameters.

EffNet-B4's stem produces 48 different feature maps from the input image, I take an identical approach to the B0 stochastic stem and add an additional 48 feature maps from the stochastic stem (for a total of 96 feature maps) which is then passed to the backbone of repeating inverted mobile residual blocks. The first block produces 24 feature maps from the 48 (96 with stochastic) feature maps.

The results are illustrated in figure 22:

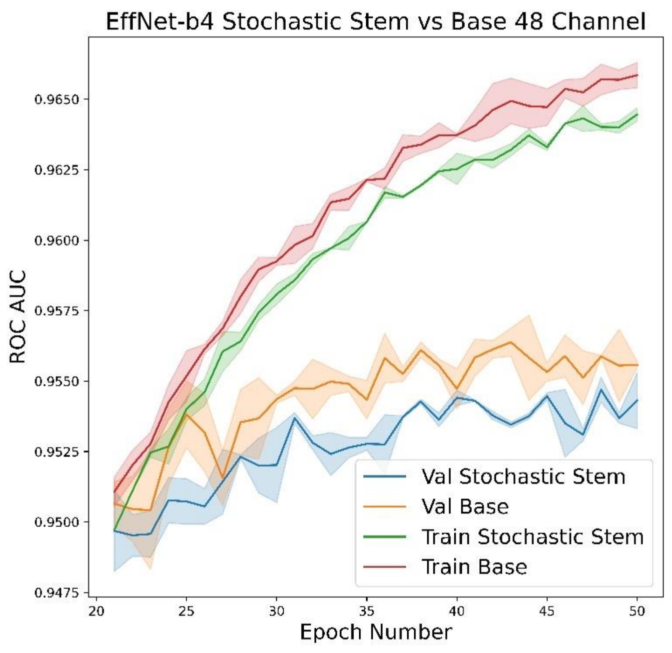


Figure 22

Similar results are observed for the B4 backbone as the B0 backbone. Both training and validation ROC AUC are lower in the stochastic stem architecture relative to the base model. This implies that the capacity of the backbone model is not the limiting factor in incorporating the stochastic high resolution feature maps with the global low resolution feature maps.

It was theorized that due to the difference in the global and high resolution feature maps, that the first layer of the CNN architecture was unable to combine the low level feature maps.

This lead to the inspiration for the stochastic skip architecture where in the stochastically extracted feature maps and the global low resolution feature maps go through separate levels of non-linear transformation (separate inverted bottleneck blocks) before being concatenated and combined at a higher level in the architecture.

5.1.3 Stochastic Skip Architecture Model

The stochastic skip architecture as described in section 4.1.3 was constructed and evaluated using the same training routine and optimizer as with the stochastic stem approach. The base architecture is bifurcated through the first 5 inverted bottleneck blocks; the 32 low resolution 'global' stem and 32 stochastic stem feature maps are independently processed through 5 dedicated blocks to produce 40 56x56 feature maps each which are then combined into 80 56x56 feature maps via concatenation. The combined feature maps are then fed through the remainder of the architecture.

The model was evaluated in a similar fashion to the Stochastic Stem approach. The validation and training ROC AUC as compared to the base and base model with 64 channel stem are compared from the 20th epoch onward (to highlight differences).

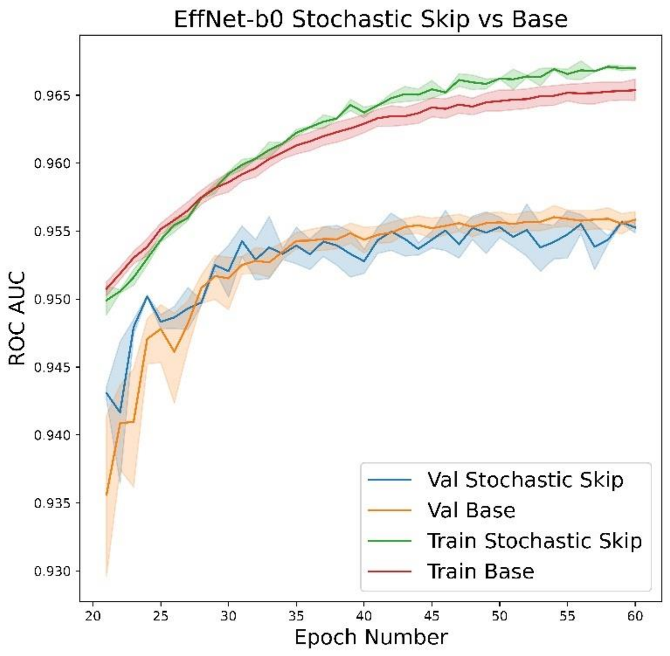


Figure 23

Observing the above graphs, it can be inferred that the skip stochastic approach appears to have improved performance over the stochastic stem approach relative to the baseline B0 model in both 32 and 64 channel configurations. This infers that the model has benefited from learning inverted bottleneck blocks dedicated to processing the feature maps generated from the stochastic convolutional stem. These dedicated inverted residual blocks have learned to combine features generated only from the stochastic stem layers.

However, while the stochastic skip model outperforms the base B0 model in training performance, it still fails to outperform the base architecture in validation performance, though it has improved performance over the stochastic stem model.

It was theorized the model may be lagging in performance relative to the baseline because the stochastic and global features were combined at a convolutional layer rather than fully connected layer.

Since not all regions of the image may contain features that drive prediction, even after multiple layers of non-linear transformation the stochastic features may not all contain predictive features and a stochastic layer may not have the same capacity as a fully connected layer to ignore these features.

This theory lead to the inspiration for the full stochastic split model, where separate B0 backbones are dedicated to the standard stem and the stochastic stem. The stochastic and standard features are then combined at the head of the architecture in the fully connected layer.

5.1.4 Full Stochastic Split Architecture Model

Observing that processing the stochastically extracted high resolution features through separate mobile inverted bottleneck blocks before combining them with the global low resolution feature maps improved performance, I theorized that instead of combining the features at a convolutional layer, that they might be better combined at a fully connected layer.

The model described in 4.1.4 was then constructed. The model is effectively 2 EfficientNet-B0 backbones, one for the stochastic stem and one for the low resolution stem. The output feature maps of the backbone are combined in a fully connected head with dropout.

The training and validation results of the model, relative to a single baseline B0 model is shown in figure 24 below:

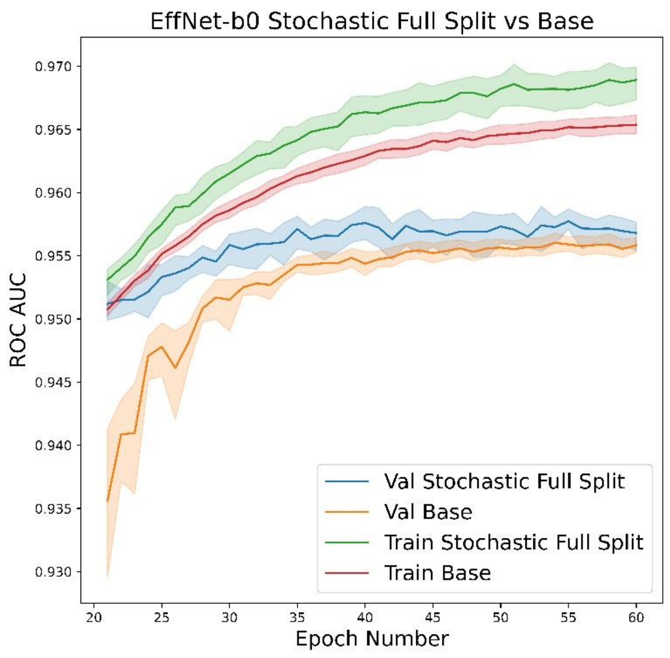


Figure 24

The full split model consistently outperforms the base model in training and in model valudation performance and achieves quality in fewer epochs than the baseline B0. This result validates the theory that the two feature maps are best combined at a fully connected layer in the network.

The full split model contains around 9MM parameters where in contrast, the basline B0 contains 4.5MM parameters. To evaluate the improvement in performance relative to parameter count, the full split model was compared against the B4 base model which contains 17.5MM parameters. The validation performance for the B0, B4 and full split model are shown in the figure 25 below:

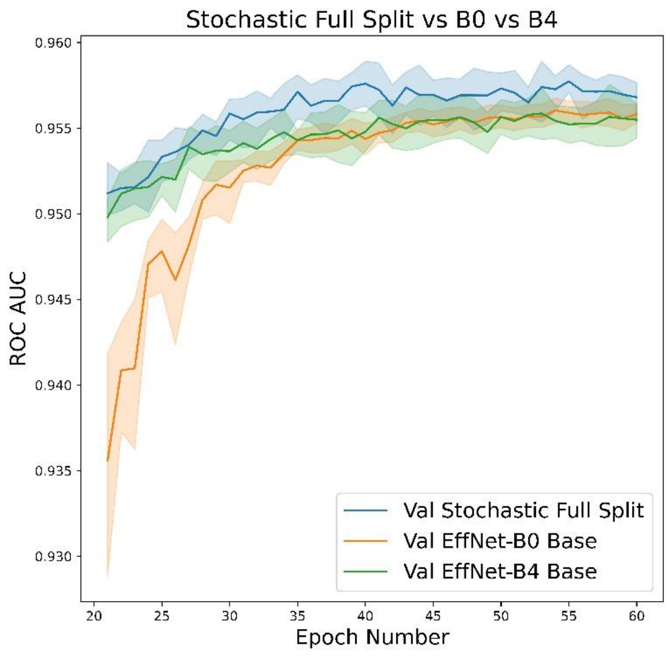


Figure 25

The results illustrate that the full stochastic split model outperforms both the B0 model and the B4 model while containing around ½ the parameter count in validation performance.

5.1.5 Full Stochastic Split TTA

An embedded advantage that the stochastic models have over base models is that due to the stochastic nature of the stochastic stem, averaging over multiple inference passes will give different results and may be equated to TTA but does not require reaugmentation of the input images.

The full split model was then retrained, but the model was run 3 times on each validation example and the output was averaged. The validation results of the full split model using single inference is compared against 3 rounds of averaged inference are is shown in figure 26:

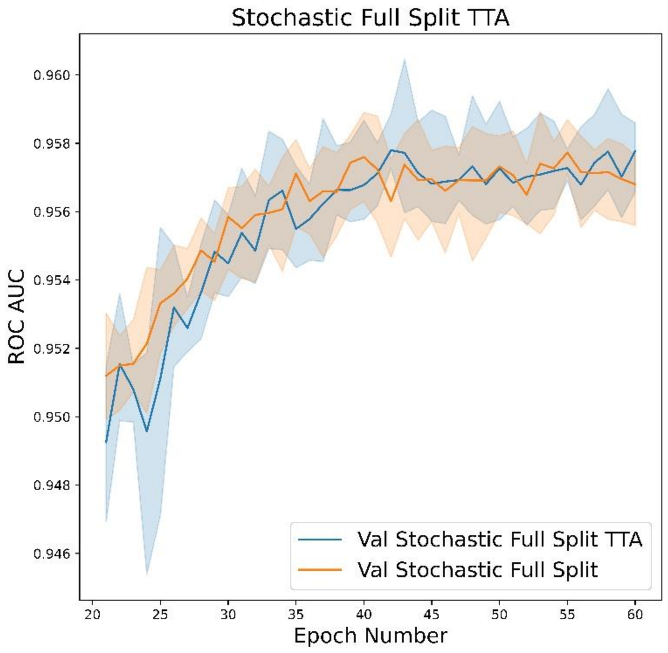


Figure 26

The results demonstrate that averaging inference over multiple runs (3 in this example) lifts performance after the model reaches quality. The high and lows demonstrate that averaging inference runs will either meet or exceed a single inference run.

5.1.6 Collected Results

The below table shows the collected results of the maximum achieved validation ROC AUC of each model types across all runs.

Model	Max Validation ROC AUC	Parameter Count
B0	0.9584	4MM
B4	0.9588	17.5MM
Stochastic Stem	0.9534	4.2MM
Stochastic Skip	0.9571	6MM
Full Stochastic Split	0.9594	8MM
Full Stochastic Split TTA	**0.9605**	8MM

Figure 27

The full split and full split model with 3 rounds of averaged inference outperform all evaluated models. The highest performing model by max validation ROC AUC score was the full split model with 3 rounds of averaged inference.

In this application, the stochastic split approach achieves significant improvements in performance at ~1/2 the model size of the EfficientNet-B4 model which is characterized by an increase in model input resolution from 224x224 to 380x380 and an increase in depth.

The stochastic split model maintains the B0 input resolution of 224x224 and depth, but increases the width by adding an additional B0 backbone dedicated to the non-linear transformation of the stochastic stem.

5.1.7 Runtime Analysis

The collective runtime of each of the model approaches are aggregated in figure 27. The Stochastic Stem, Skip and Full Stochastic Split runtimes include the optimized approach to the stochastic convolutional layer.

The total number of training iterations was limited due to the total runtime of a training loop for the the B4 and Full Stochastic Split Models.

With low level cuda kernel optimization of the forward and backward gradient propagation, the training time of the stochastic apporaches could be brought down further.

Model	Training Time Per Epoch	Total Training Time
B0	1.5 mins/epoch	1.5 Hours
B4	4.67 mins/epoch	4.7 Hours
Stochastic Stem	2 mins/epoch	2 Hours
Stochastic Skip	2.5 mins/epoch	2.5 Hours
Full Stochastic Split	4 mins/epoch	4 Hours

Figure 27

5.2. Discussion of Insights Gained

The initial results demonstrate that, under the specific training routine configuration and optimizer choice that the “stochastic stem” architectural approach where in the features extracted via stochatic convolution are combined with global features at the stem fail to provide any uplift and instead drop performance over the base EfficientNet B0 and B4 architectures.

Additionally, the base model capacity was found not to be a limiting factor. Moving from the 4MM parameter B0 backbone to the 17.5MM parameter B4 backbone did not improve the performance of the stochastic stem architecture. This clearly indicates that increasing capacity and depth of the stochastic stem based architectures didn’t extract more meaningful predictions from the random high resolution features.

However, the ‘stochastic skip’ architecture’s improvement in performance over the ‘stochastic stem’ architecture, and improvement over the baseline in training indicated that the stochastically extracted features could be meaningfully used after some layers of non-linear transformation.

The stochastic convolution extracts features from random regions of the high resolution image and since not all regions of the image may contain class identifying information, some of these feature maps may not be useful for prediction. Incorporating these feature maps with the global resolution feature maps at a convolutional layer forces the model to integrate the perhaps not useful feature maps into the output. This may have been the reason why the stochastic stem, and to a lesser extent the stochastic skip architecture performed worse than the baseline architecture in validation.

These findings lead to the inspiration of the full stochastic split architecture, which is effectively composed of two EfficientNet B0 architectures, one with the stochastic stem and one with the low resolution stem. The features were then combined at the head of the architecture using a dense connection in an effort to give the model the capacity to ignore some of the high resolution features that may have been extracted from regions of the image that didn’t contain predictive information in addition to allowing for more layers of dedicated non-linear transformation.

This approach was the most effective integration of the high resolution stochastic convolutions. The model outperformed both the baseline B0 and B4 architectures measured using validation ROC AUC.

The key factors in the success of this stochastic convolution approach are the combinination of separate dedicated non-linear residual type transformations of extracted feature maps, where in the features are combined at the head of the architecture in a fully connected rather than at convolutional layer.

A key technical hurdle in the project was in optimizing the stochastic convolution implementation. The initial naïve implementation increased runtime in the base architecture B0 by a factor of over 15x. Using cuda runtime analysis, optimizations were made that improved runtime to 4x over the base architecture. However, this remained a challenge over the course of the project since only a limited number of experiments could be performed due to runtime constraints.

6. Conclusion

The goal of this project was to evaluate the hypothesis that extracting high resolution features from random regions of high resolution images in combination with features extracted from the same images after resize compression may provide an uplift in CNN performance in image classification tasks where objects were homogeneous and class delineation may be driven by fine differences between objects.

The ISIC melanoma dataset was chosen to evaluate this approach due to the high resolution of the images, the homogeneity of the objects being classified (skin lesions), and the fine nature of the class delinating features (edges of moles).

To test this hypothesis the EfficientNet backbone architectures were chosen due to their highly optimized performance relative to FLOPs and size.

Various approaches to integrating the 'stochastic convolution' layer into the backbones were explored and all approaches were evaluated relative to baseline, unaugmented EfficientNet performance measured using the common reciever operator curve area under the curve measure (ROC AUC).

All approaches integrated the 'stochastic convolution' into the first convolutional layer of the model (otherwise named the 'stem') alongside the base stem applied to the full resized image. However, multiple architectural approaches to integrate the randomly extracted high resolution features into the model backbones were investigated.

The first architectural approach evaluated was the 'Stochastic Stem' architecture, wherein the features were randomly extracted from the high resolution image and were combined with globally extracted features from the base stem of the architecture after non-linearities were applied via concatenation in channels. This extended featuremap was then passed through the B0 and B4 backbones.

This initial approach consistently underperformed the base model architecture in validation performance. Observing the same results in the larger B4 model confirmed that capacity was not the limiting factor. Instead this appeared to be due to the model lacking the ability to ignore redundant information in the higher resolution features using convolutional filters.

The second architecture evaluated coined 'Stochastic Skip' went some way to validating the findings from the initial approach. In this architecture, the stochastically extracted high resolution feature maps were passed through dedicated Mobile Inverted Bottleneck blocks before being combined with the existing global features. This approach closed the gap in performance between the base and stochastic approach, but still slightly underperformed the base architecture.

Theorizing that simply combining the stochastic feature maps with the global feature maps via concatenation at a convolutional block would not give the model the capacity to extract only predictive information from the random regions, the 'Full Stochastic Split' architecture was devised. This architecture dedicated independent B0 backbone architectures to both the base global stem and the stochastic stem, the features were then combined at a dense layer in the top of the model.

This architectural approach proved to be the most successful implementation of the stochastic convolution, exceeding performance of the baseline B0 model and the much larger B4 base model in both training and validation performance.

The 'Full Stochastic Split' architecture consistently outperformed the B4 model at less than 50% of the size. This maximum performance was increased by averaging inference predictions of the stochastic full split model in an approach that is similar to TTA but doesn't require reaugmentation of the input images.

The lack of optimization of prerequisite functions in the Pytorch API to apply a stochastic convolution remained a challenge in model exploration and testing. Pytorch level optimizations reduced runtime significantly. However cuda kernel level optimization of stochastic convolution using pointers and references rather than copy operations would have the potential to bring runtime overhead down further.

The optimizer and learing approach suggested in the original EfficientNet paper was used to evaluate the models, and may not have been as successful as some other configuration in training the experimental stochastic models evaluated in this project. With lower level optimziation and more compute for hyperparameter and optimizer tuning further performance improvement could potentially be achieved with the proposed approach.

This project successfully lays out a guideline approach to integrating the proposed stochastic convolutional layer into lightweight CNN architectures to significantly improve performance in high resolution image classification on the task analyzed without scaling the parameter count in proportion with the target resolution.

6. Acknowledgement

All the code in this project was my own product, however I found the following resources useful in implementation. The official tensorflow implementation of the EfficientNet architectures [18] and discussions [20] [21] were helpful in understanding the specification of dropconnect rate in the backbone architecture and Borna Ahmadzadeh's [19] explanation of squeeze and excitation was helpful in understanding how it could be implemented without using a dense layer in Pytorch.

The official Pytorch documentation was useful in providing methods for code profiling.

7. References

[1] ImageNet Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image

database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248–255).
[2] Mingxing Tan, Quoc V. Le “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”
[2] H. Li, “Author Guidelines for CMPE 146/242 Project Report”, Lecture Notes of CMPE 146/242, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.
[3] Official Tensorflow EfficientNet Implementation
[4] ROC AUC Curve https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
[5] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, Herve J ´ egou “FIXING THE TRAIN-TEST RESOLUTION DISCREPANCY: FIXEFFICIENTNET” https://arxiv.org/pdf/2003.08237.pdf
[6] Qizhe Xie, Eduard H. Hovy, Minh-Thang Luong,
and Quoc V. Le, “Self-training with noisy student improves imagenet classification,” arXiv preprint
arXiv:1911.04252, 2019.
[7] Christos Kyrkou, Theocharis Theocharides ‘EmergencyNet: Efficient Aerial Image Classification for Drone-Based Emergency Monitoring Using Atrous Convolutional Feature Fusion’
[8] https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet
[9] https://www.kaggle.com/c/siim-isic-melanoma-classification/discussion/175412
[10] Gutman, David; Codella, Noel C. F.; Celebi, Emre; Helba, Brian; Marchetti, Michael; Mishra, Nabin; Halpern, Allan. "Skin Lesion Analysis toward Melanoma Detection: A Challenge at the International Symposium on Biomedical Imaging (ISBI) 2016, hosted by the International Skin Imaging Collaboration (ISIC)". eprint arXiv:1605.01397. 2016.
[11] Codella N, Gutman D, Celebi ME, Helba B, Marchetti MA, Dusza S, Kalloo A, Liopyris K, Mishra N, Kittler H, Halpern A. "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)". arXiv: 1710.05006 [cs.CV]
[12] Noel Codella, Veronica Rotemberg, Philipp Tschandl, M. Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, Harald Kittler, Allan Halpern: “Skin Lesion Analysis Toward Melanoma Detection 2018: A Challenge Hosted by the International Skin Imaging Collaboration (ISIC)”, 2018; https://arxiv.org/abs/1902.03368
[13] Tschandl, P., Rosendahl, C. & Kittler, H. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5, 180161 doi:10.1038/sdata.2018.161 (2018).
[14] Tschandl P., Rosendahl C. & Kittler H. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5, 180161 doi.10.1038/sdata.2018.161 (2018)
[15] Noel C. F. Codella, David Gutman, M. Emre Celebi, Brian Helba, Michael A. Marchetti, Stephen W. Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, 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)”, 2017; arXiv:1710.05006.
[16] Marc Combalia, Noel C. F. Codella, Veronica Rotemberg, Brian Helba, Veronica Vilaplana, Ofer Reiter, Allan C. Halpern, Susana Puig, Josep Malvehy: “BCN20000: Dermoscopic Lesions in the Wild”, 2019; arXiv:1908.02288.
[17] Rotemberg, V., Kurtansky, N., Betz-Stablein, B., Caffery, L., Chousakos, E., Codella, N., Combalia, M., Dusza, S., Guitera, P., Gutman, D., Halpern, A., Helba, B., Kittler, H., Kose, K., Langer, S., Lioprys, K., Malvehy, J., Musthaq, S., Nanda, J., Reiter, O., Shih, G., Stratigos, A., Tschandl, P., Weber, J. & Soyer, P. A patient-centric dataset of images and metadata for identifying melanomas using clinical context. Sci Data 8, 34 (2021). https://doi.org/10.1038/s41597-021-00815-z
[18] Tensorflow EfficientNet Official Github https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet
[19] Borna Ahmadzadeh https://python.plainenglish.io/implementing-efficientnet-in-pytorch-part-3-mbconv-squeeze-and-excitation-and-more-4ca9fd62d302
[20] Discussion: https://github.com/tensorflow/tpu/issues/3810.2
[21] Discussion 2: https://forums.fast.ai/t/efficientnet/46978/76

8. Appendix

8.1 Individual Student Contributions in Fractions

This project was my own work.

8.2 If/as needed: additional diagrams, source code listing, circuit schematics, relevant datasheets etc.