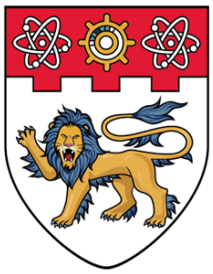

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4001 : Neural Network & Deep Learning

GROUP PROJECT

No.	Name	Matriculation Number
1	Darren Choo Jian Hao	U2121223J
2	Gan Hao Yi	U2120769F
3	Foo Jin Rui	U2120527E

Table of Contents

1. Introduction

Our group has chosen Task F - Flowers Recognition. We will be using 3 pre-trained models and experimenting with different modifications to each of them. In each of these models, we will have two baselines to compare our modifications to. The baselines include:

1) Freezing the whole architecture and training only the last layer.
2) Training the entire original architecture with pre-trained weights.

The 3 models and their modifications are as follows:

1) EfficientNet - Architecture Modification
2) Vision Transformer - Loss Function modification
3) Resnet - Architecture and Loss Function Modifications

2. EfficientNet

2.1 Introduction

This section delves into the EfficientNet family of neural networks to tackle classification tasks on the Flowers102 dataset. EfficientNet stands out as a family of convolutional neural network architectures that has demonstrated superior performance compared to other architectures like ResNet and InceptionV2 on the ImageNet dataset, while remarkably maintaining a significantly smaller number of parameters as shown in Figure 1.

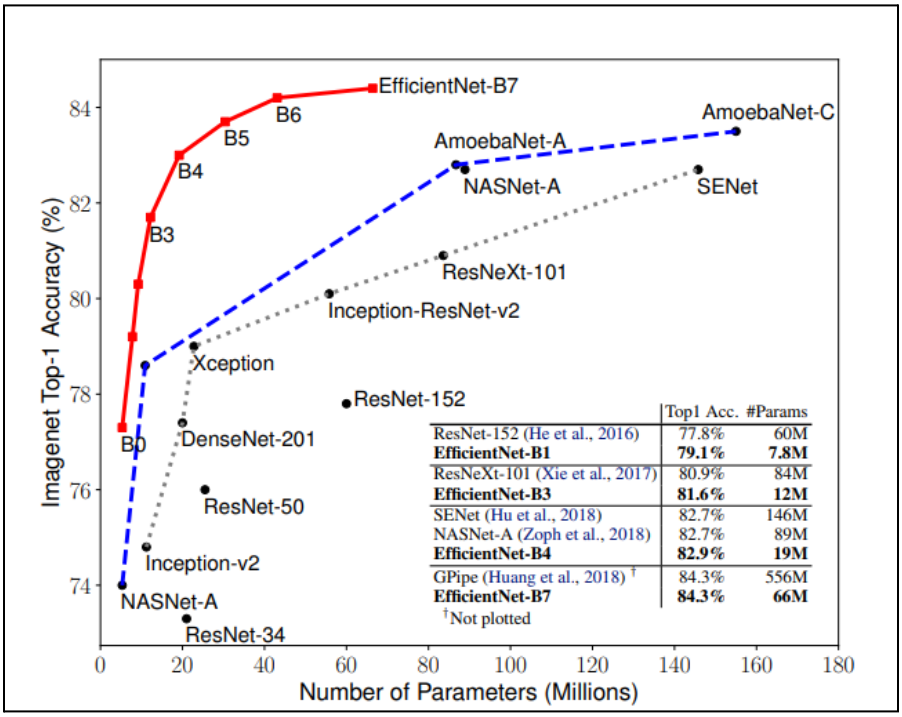

Figure 1. ImageNet Accuracy Against Number of Parameters for Different Models

The distinctive feature of EfficientNet lies in its innovative scaling method, which uniformly adjusts all dimensions of width, depth, and resolution using a compound coefficient. This approach sets it apart from other neural networks such as ResNet, where performance improvements are typically achieved by scaling only the depth of the network.

This report primarily centers on the baseline model, EfficientNetB0. EfficientNetB0 is chosen for its compatibility with 224 by 224 resolution input images, enabling direct comparisons with other models in this report, all of which also use 224 by 224 input sizes. The depth, width, and resolution of the architecture for this analysis have been intentionally refrained from modifications as these aspects have already undergone meticulous optimization through a grid search approach using the compound scaling method.

In the upcoming subsection, we will explore an architectural modification to EfficientNetB0. Its performance based on its accuracy in classifying the Flowers102 dataset will then be assessed against the baselines.

2.2 Replacing Squeeze-and-Excitation (SE) with Efficient Channel Attention (ECA)

EfficientNet employs a series of inverted residual (MBCONV) blocks, a concept first introduced in the MobileNetV2 CNN architecture. Within each MBCONV block, EfficientNet incorporates a SE module that enables the network to learn the significance of each feature map in the computation of the output. In this context, Figure 2 illustrates the architecture of a SE module. Initially, the input is 'squeezed' through global average pooling, and then 'excited' through a neural network before it is scaled and merged with the input to produce the output layer.

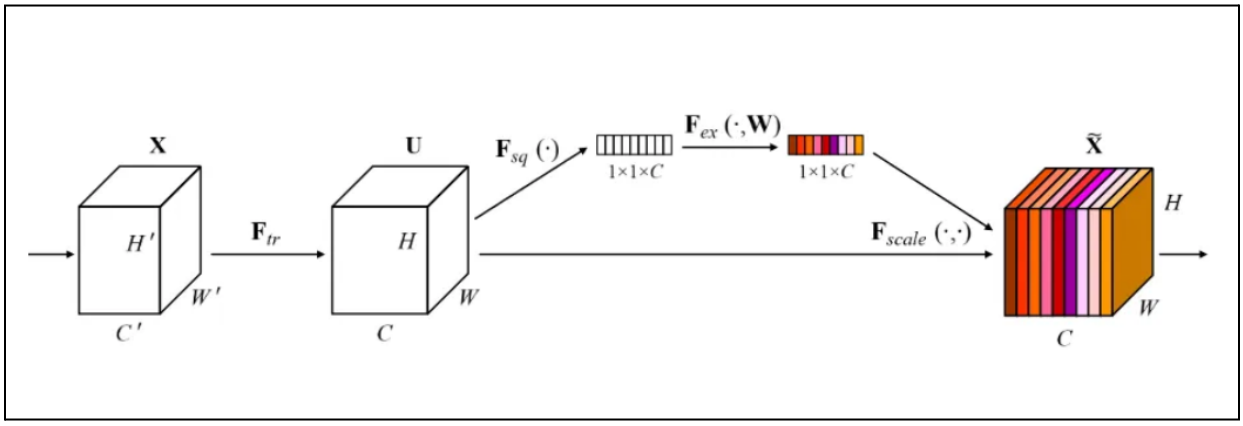

Figure 2. Architecture of the Squeeze-and-Excitation Module

While the above has demonstrated its effectiveness in enhancing the accuracy of baseline models, it is important to note that their inclusion of dimensionality reduction within the fully connected layers, aimed at simplifying model complexity, has been reported to have a potential negative impact on model performance. Additionally, it is worth considering that the fully connected layers introduce a significant number of parameters, which can lead to increased computational costs.

To address these concerns, our group has investigated the utilization of ECA as an alternative. Figure 3 provides an overview of the ECA architecture. This approach employs global average

pooling followed by 1D convolution, which is then combined with the input via element-wise multiplication to create the output layer. As a result, ECA offers the advantage of requiring fewer trainable parameters and eliminates the need for dimensionality reduction.

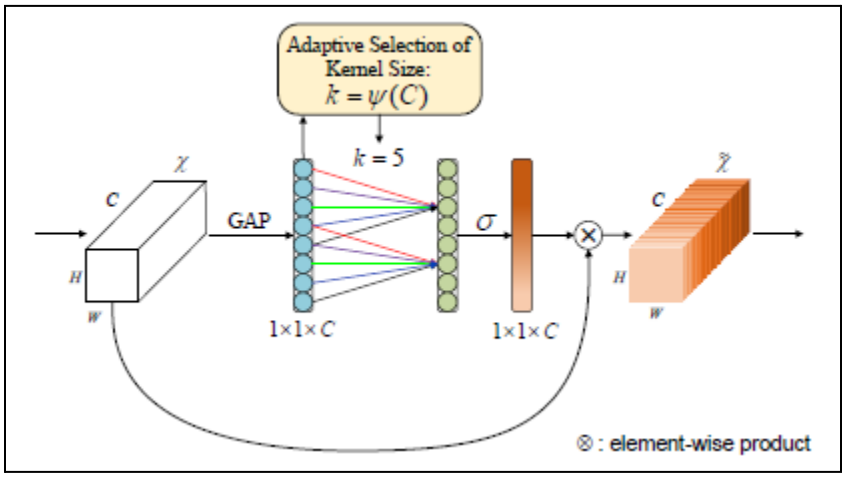

Figure 3. Architecture of the Efficient Channel Attention Module

We have replaced all the SE modules in each MBCONV block with the ECA module, and the results of these modifications can be seen in the next subsection.

2.3 Experiments & Results

Figure 4 shows the outcomes of the aforementioned modification in comparison to the baseline models. As illustrated in Figure 4, when considering both training and validation accuracies, the EfficientNetB0 model with ECA implementation outperformed the original EfficientNetB0 model with only the last layer trained while keeping the rest of the architecture frozen with a maximum validation accuracy of 0.885 compared to 0.867. However, it should be noted that it did not surpass the performance of the EfficientNetB0 model with no layer freezing with a maximum validation accuracy of 0.910.

In terms of training duration, the model with freezing completed its training in half the time compared to the other two models, which had similar training times.

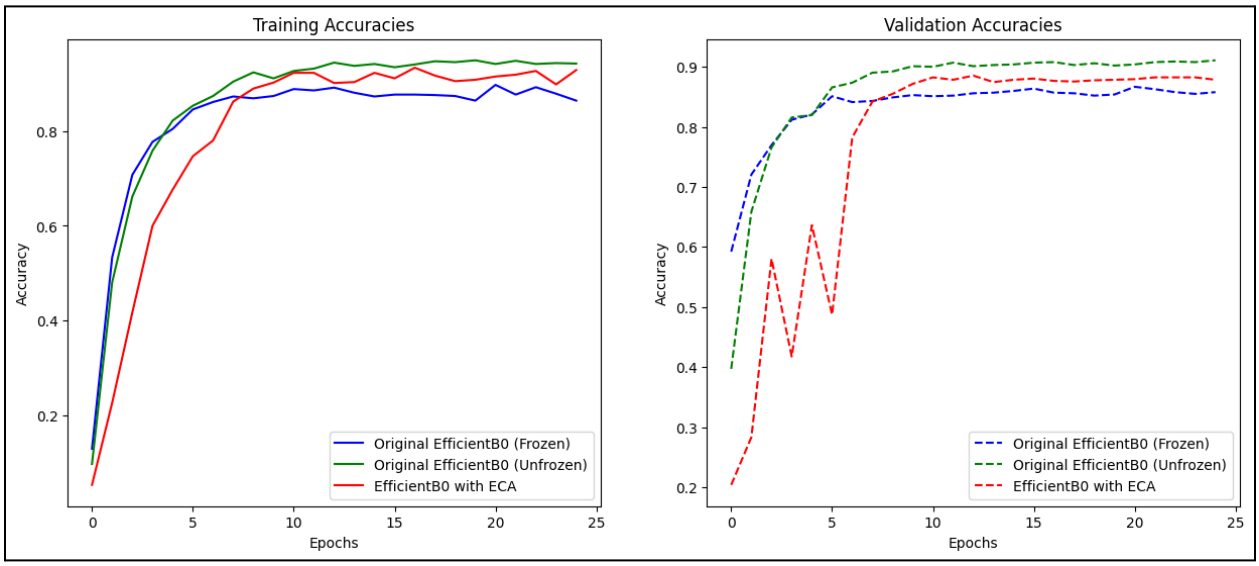

Figure 4. EfficientNet Train and Validation Accuracies for Different Models against Epochs

In terms of test accuracy, the model with freezing performed the worst, achieving an accuracy of 0.836. On the other hand, both the model without freezing and the model with ECA implementation yielded the same result, with test accuracies of 0.879.

2.4 Conclusion & Future Improvements

The test results reveal that despite the efficient channel attention using fewer parameters in each MBCONV block compared to the squeeze-and-excitation module, it is able to attain a comparable level of accuracy. Despite its parameter efficiency, the training time is similar to that of the original EfficientNetB0 model with unfrozen layers. This similarity may be attributed to the fact that the reduced parameters in the ECA modules contribute only a small proportion to the total trainable parameters, and thus did not significantly impact the training time. To enhance the model's robustness in future iterations, a potential avenue could involve complementing the channel attention module in EfficientNetB0 with a spatial attention module.

3. Vision Transformers

3.1 Introduction

Vision transformers (VIT), instead of relying on convolutional layers, leverage on self-attention mechanisms, allowing them to capture intricate relationships and dependencies within the data.

In this section, we want to compare the effects of triplet loss (TL) on the task of image classification using VIT. This is different from the regular loss function used for image classification which is cross-entropy loss (CEL).

For triplet loss, it requires our data to be in the form of triplets (anchor, positive, negative). Anchors are our base images, positives are images with the same class, and negatives are images with a different class. Triplet loss encourages the model to learn feature representations such that it minimizes the distance between anchor and positive, while maximizing the distance between anchor and negative.

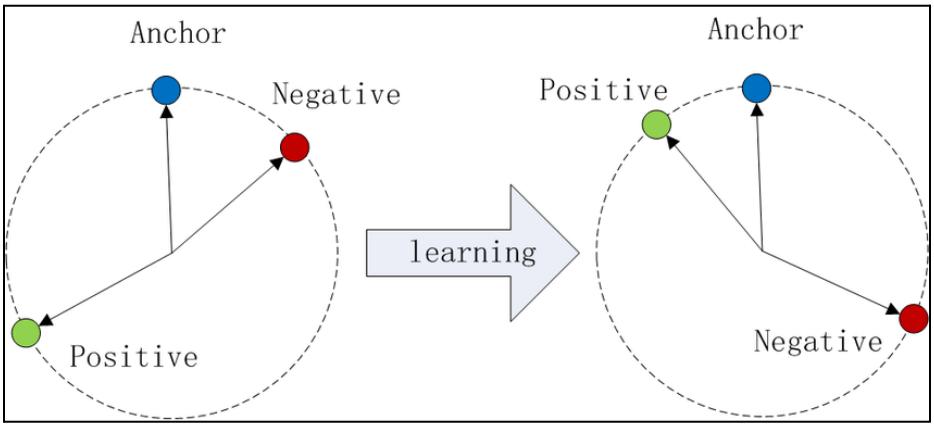

Figure 5. Visual Representation of Triplet Loss Learning

To prepare for the use of triplet loss, we created a custom dataset in order to obtain our triplets. Each image will be randomly paired with an image of the same class, and another of a different class.

We also chose the pretrained model “vit_b_16” with pretrained weights as our image classification model.

3.2 Triplet Loss with Triplet Dataset

We first tested the use of TL alone, but we were greeted with horrible validation accuracy during training, as low as 0.03. Upon further research, we found out that this was due to TL itself not considering the ground-truth class labels. This means that the output channel does not have any semantic meaning. As a result, when we backpropagate the loss, the model is not learning anything meaningful to help in the image classification task

Therefore, in order to incorporate such semantic meaning with TL, we hypothesized the summation of TL together with CEL to obtain a new total loss. This way, we get the benefits of both improved feature representations from TL, and semantic meaning from CEL. We will be referring to this summation of loss as Triplet Cross-Entropy Loss (TCEL)

3.3 Experiments & Results

Figure 5 shows the outcomes of the aforementioned modification in comparison to the baseline vit_b_16 (Frozen and Unfrozen). As illustrated in Figure 5, when considering both training and validation accuracies, the VIT with TCEL outperformed both the original frozen and unfrozen VIT with only CEL, with a maximum validation accuracy of 0.892, 0.872 and 0.691 respectively.

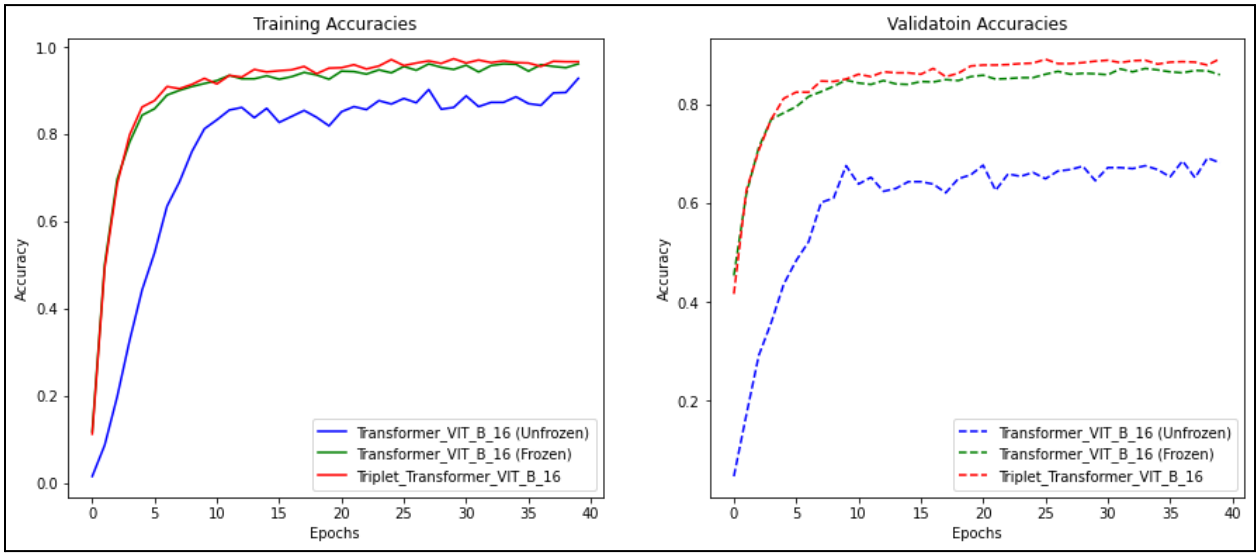

Figure 6. VIT Train and Validation Accuracies for Different Models against Epochs

In terms of test accuracy, the VIT with TCEL performed the best again, achieving an accuracy of 0.876. On the other hand, the models that were frozen and unfrozen with CEL achieved an accuracy of 0.855 and 0.646 respectively.

One interesting thing that we noted was the drastic drop in performance between the frozen and unfrozen original VIT. We initially expected the unfrozen VIT to achieve a better performance since the model is now able to learn on the flowers dataset and update the parameters for all of its layers. The Flowers102 dataset had only 8189 images, and we are aware that VIT's require large amounts of training data in order to match the performance of CNNs.

Upon further research, we found out that the unfreezing and fine-tuning of the VIT on a smaller dataset such as Flowers102 might have caused the VIT to forget the knowledge acquired during pre-training, resulting in the poor performance.

3.4 Conclusion & Future Improvements

From the tests, VIT with TCEL performed the best, though it only beat the frozen VIT by a small margin.

One way we can improve the model is by improving the choice of our negatives in triplet loss. Currently, we have been selecting random negatives for our anchor. We can explore the idea of selecting harder triplets, whereby the negative sample is close to the anchor sample, or even closer than the positive sample itself. Easy triplets whereby the negatives are very different from the anchor may not provide much useful training information and the loss obtained could be 0. With harder triplets, the model will have to focus more on learning from challenging examples, and this could help it better separate the classes in the embedding space.

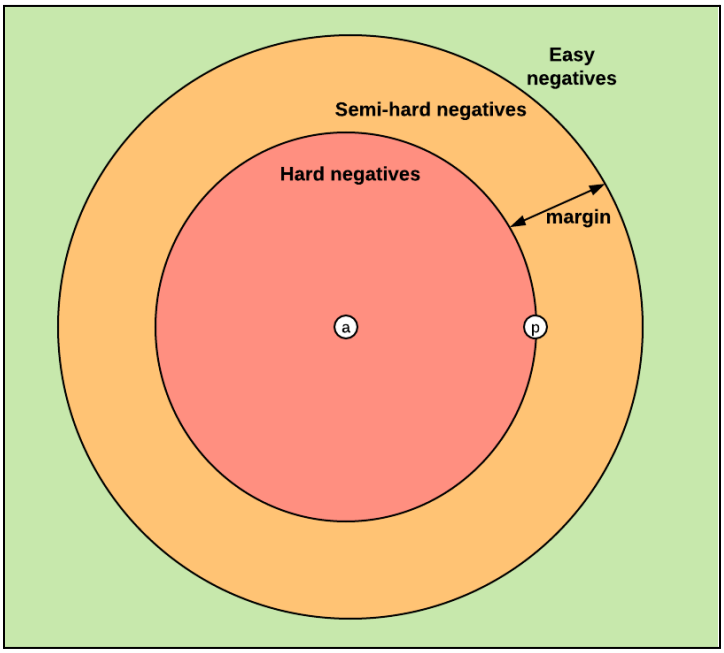

Figure 7. Visual Representation on Different Types of Negatives in Triplet Loss

4. ResNet

4.1 Introduction

This section delves into the ResNet family of neural networks to tackle image classification tasks. We mainly used ResNet18 as it is a deep convolutional neural network with a simple structure, while incorporating skip connections to counter the risk of vanishing gradients. The architecture of the network is in the figure below, with a convolution layer, 4 layers of different depths of filters, and a fully connected layer.

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

Figure 8. ResNet18 Architecture

4.2 ResNet with Deformed Convolution

Things we tried:

- Original ResNet
- ResNet with deformed convolution after conv1 of ResNet18
- ResNet with deformed convolution + conv1 of ResNet18

Deformable convolution allows the filters to change their shape to adapt to the changes in features. Initially, the filters have a fixed shape, but the filters are allowed to learn to adjust their shape to deal with unknown complex transformations in the input.

We used the deform_conv2d function from torchvision to perform the deformed convolutions for our model.

In this modification of the ResNet18 model, we are attempting 2 different methods of implementing deformable convolution.

- X (input) -> conv1 -> deformed_Conv2d -> … (rest of network)
- X (input) -> (conv1 + deformed_Conv2d) -> … (rest of network)

We initially experimented with adding a deformed convolution after each layer (layer 1 - layer 4) in ResNet18, but it resulted in very poor accuracies, from 10%-30% accuracy. We decided to try adding 1 deformable convolution after the “conv1” filter in the ResNet architecture so that the layer can take in an input that still has most of its original features after the first convolution, as shown in the left image in Figure 9.

As we experimented more, we realized that since deformable convolution allows the filter to change its shape to learn more about the surrounding features. We tried to combine the “conv1” layer together with a deformable convolution by adding them together so that the model can learn more of the input features, as shown in the right image in Figure 9.

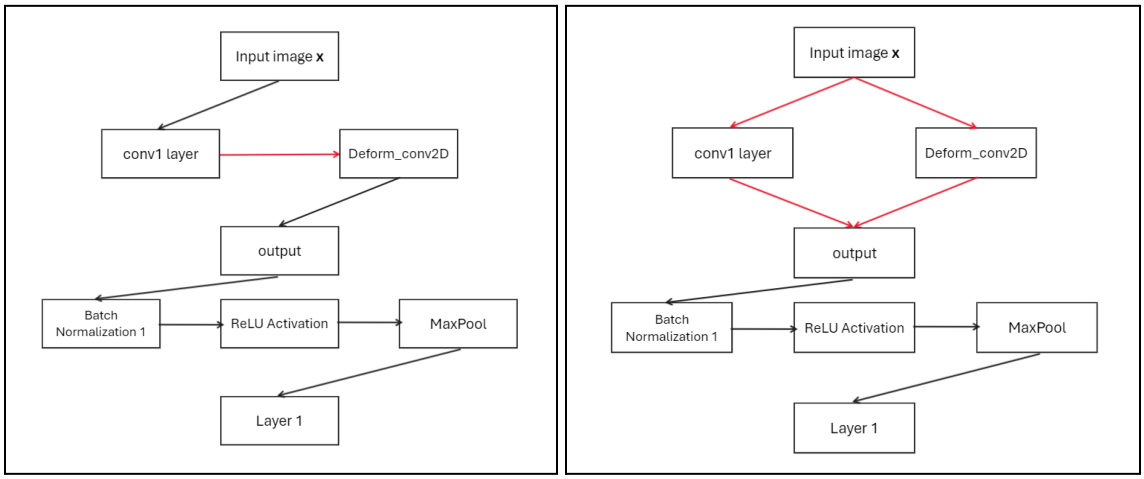

Figure 9. New structure of modified ResNet18 Model (conv1 -> deformed, conv1 + deformed)

4.3 Experiments & Results

Figure 10 shows the outcomes of the modifications in comparison to the baseline ResNet18 model (Frozen and Unfrozen). The ResNet18 model with unfrozen parameters performed the best with a training accuracy of 0.874, and a validation accuracy of 0.903. The ResNet18 with frozen parameters had a training accuracy of 0.787, and a validation accuracy of 0.836.

As illustrated in Figure 10, the model with a deformed_conv2d layer **after** conv1 layer (Green) resulted in the lowest accuracies for both training and validation, with 0.659 and 0.625 respectively. However, this is not nearly as good as the original ResNet model at 0.874.

The model with deformed_conv2d layer **+** conv1 layer still performed below the baseline ResNet18 model (Unfrozen), with training accuracy at 0.8529, and validation accuracy at 0.788. However, we can see that it performed pretty closely to the ResNet18 model (Frozen parameters), with its training accuracy higher at 0.853 to frozen ResNet18’s 0.787. However, the validation accuracy was still slightly below the ResNet18 model (Frozen parameters), at 0.788 to frozen ResNet18’s 0.836 validation accuracy.

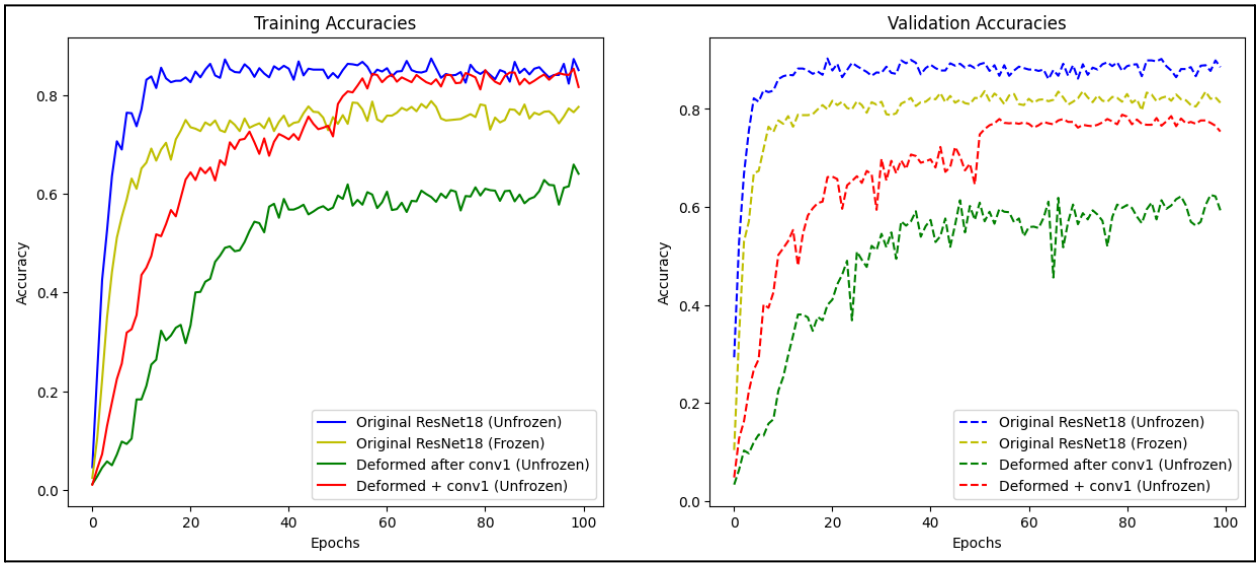

Figure 10. ResNet Train and Validation Accuracies for Different Models against Epochs

In terms of test accuracy, the original ResNet18 model still performed the best. It had a test accuracy of 0.873. The second best result was from the ResNet18 model with frozen parameters, at 0.801. Next, we have the conv1 + deformed_2d model with a test accuracy of 0.731. Lastly, we have the model with a deformed convolution filter after the conv1 layer, with a test accuracy of 0.565.

4.4 Conclusion & Future Improvements

From the tests, we can see that deformed convolution + conv1 together resulted in a far better classification accuracy than deformed convolution after conv1. The model could learn the features a lot better if we add the layers together instead of passing them through from conv1 to deformed layer. However, when more deformed layers are introduced, the model could not learn the features of the image properly, with accuracies dropping below 0.5. Some suggestions to how one might improve this is to have pre-trained weights to use for the deformable convolution layers that are integrated into the ResNet18 architecture. Currently, the weights are randomly initialized, which means we have to learn from scratch. If pre-trained weights for the deformable convolution layers exist, it might assist the model in learning important features more quickly.

References

deform_conv2d — Torchvision main documentation. (n.d.). PyTorch. Retrieved November 5, 2023, from https://pytorch.org/vision/main/generated/torchvision.ops.deform_conv2d.html

hirotaka0122. (2020, January 11). *Triplet loss with pytorch*. Kaggle. https://www.kaggle.com/code/hirotaka0122/triplet-loss-with-pytorch

Raza, A. (2019, July 7). *Type of convolutions: Deformable and Transformable Convolution | by Ali Raza*. Towards Data Science. Retrieved November 5, 2023, from https://towardsdatascience.com/type-of-convolutions-deformable-and-transformable-convolution-1f660571eb91

Tan, M., Le, Q. V. (2020, September 11). *EfficientNet: Rethinking model scaling for Convolutional Neural Networks*. arXiv.org. https://arxiv.org/abs/1905.11946v5

Van Der Merwe, R. (2020, December 3). *Triplet entropy loss: Improving the generalisation of Short Speech Language Identification Systems.* arXiv.org. https://arxiv.org/abs/2012.03775v1

Wang, Q., Wu, B., Zhu, P., Li, P., Zuo, W., & Hu, Q. (2020). ECA-Net: Efficient Channel Attention for Deep Convolutional Neural Networks. ArXiv:1910.03151 [Cs]. https://arxiv.org/abs/1910.03151