NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Assignment for CZ4042 Neural Network & Deep Learning

AY2023-2024

Group members:

Name	Matric No.
Kelly Wong Jie Yin	U2020126H
Wong Kar Onn Nicholas	U2021172D
Sanskriti Verma	U2023954E

Content Page

1. Introduction

Flowers are a quintessential part of the natural world, offering not only a visual delight but also a rich tapestry of biodiversity. Recognizing and categorizing different flower species is a challenging task that has captivated botanists, horticulturists, and computer scientists alike. Therefore, we will delve into flower recognition, using the Oxford Flowers 102 dataset as our reference point [1]. The dataset poses a unique set of challenges due to the inherent complexity of natural environments, including variations in scale, pose, and lighting conditions. One of the distinctive features of the Oxford Flowers 102 dataset is the diversity it encapsulates. Each category within the dataset contains a variable number of images, ranging from 40 to 258. This variability not only reflects the rich tapestry of the botanical world but also simulates the real-world challenges of recognizing and categorizing plants with varying characteristics. Furthermore, within these categories, there are often substantial intra-category variations, making the classification task even more intricate.

To facilitate robust model development and evaluation, the Oxford Flowers 102 dataset is thoughtfully divided into three subsets, including a training set, a validation set, and a test set. The training and validation sets, each composed of 10 images per category, amount to 1020 images each. These sets serve as essential tools for model training and validation. The test set, comprising the remaining 6149 images, consists of a minimum of 20 images per category. This large test set simulates the practical challenges of recognizing diverse flower species in real-world scenarios, adding to the dataset's richness and complexity.

In this report, we aim to explore the Oxford Flowers 102 dataset in-depth and investigate various methods and techniques for flower classification. By doing so, we hope to contribute to the broader understanding of image classification and recognition in the context of natural environments, where variability and diversity are the norm.

2. Review of existing techniques

The Flowers102 dataset, developed by Nilsback and Zisserman, aims to tackle the challenging task of differentiating between numerous flower categories [1]. This dataset highlights the complexity of flower classification, a domain fraught with subtle differences between classes, distinct from more diverse categories like bicycles, cars, and cats. Their approach is to use a Support Vector Machine (SVM) with multiple kernels, each tailored to specific color features, achieving an accuracy of 88.33(± 0.3)% on a 17-class flower dataset.

In the realm of image classification, ResNet emerged as a pivotal architecture upon its introduction by He et al. [2]. It remains a benchmark in various studies for evaluating new architectural advancements [3]. Wightman et al.'s re-examination of ResNet-50 underscored this, yielding a remarkable 97.9% accuracy on the Flowers102 dataset [4]. Further innovation came from researchers who introduced a novel optimization method. This technique, which synergizes direct parameter value optimization using the Moore–Penrose inverse with traditional gradient adjustments, significantly boosts performance in terms of both convergence rate and accuracy, maintaining computational efficiency comparable to Stochastic Gradient Descent (SGD). Implementing this approach enhanced the accuracy of several CNN models on the Flowers102 dataset; for example, ResNet-50 accuracy increased from 89.2 (±0.1)% to 93%, and MobileNet from 85.2 (±0.4)% to 92.4% [5].

Taha et al. contributed another significant development, enhancing classification performance in standard architectures like ResNet and Inception by incorporating triplet loss as a feature embedding regularizer

[6]. While traditionally underutilized due to assumed requirements for large batch sizes and high computational costs, their experiments dispelled these beliefs. They demonstrated that their method allows networks to support both classification and embedding tasks efficiently with minimal hyper-parameter adjustments, providing considerable benefits, particularly in imbalanced video datasets [7].

Following the initial breakthrough of Transformers in natural language processing, transformer architectures have swiftly become prominent in the field of computer vision. They have achieved leading-edge results in various tasks including image classification, detection, segmentation, and analysis of video content [8]. Another study conducted by Chen et al. argues that Vision Transformers (ViT) outperformed traditional Convolutional Neural Networks (CNN) such as ResNets even without pre-training or strong data augmentations [9]. For instance, they achieved notable improvements in top-1 accuracy on the ImageNet dataset for ViT-B/16 and Mixer-B/16 models.

3. Description of Methods

All neural network architectures are imported from Torchvision API, except for Transformers ViT. The models that will be compared are ResNet50, MobileNet V2, K-Shot learning, Transformers ViT, and Triplet Loss network. For each architecture, we experimented with different parameters and methods to obtain the best model. We also investigated how the accuracy changes when layers are added on top of the base model and when data augmentation and fine-tuning are performed.

3.1. Preprocessing

The datasets were fetched using the Flowers102 function from the Torchvision dataset preprocessing utilities [10]. The dataset is split into three parts, training, validation, and testing specified in the setid.mat file. Data preprocessing and augmentation are critical steps in the development of machine learning models, particularly in computer vision tasks. Therefore, for the training set, we applied transformations such as random rotation up to 30 degrees [11], in order to help the model become more robust to different orientations of objects. Furthermore, we randomly cropped and resized the input images to 224 x 224 pixels [11], for the model to learn to focus on different parts of the image. Also, we applied another transformation technique to flip the input images horizontally such that the model learns to recognize objects regardless of their orientation in the image [13]. Another normalization step is implemented to scale the pixel values in the image to have a mean of [0.485, 0.456, 0.406] and a standard deviation of [0.229, 0.224, 0.225] [14]. This aligns the data distribution with what the model was pre-trained on, allowing for better convergence during training. These are some transformation techniques that we applied to artificially increase the number of data points that the model learns, to aid in the generalization of the network.

As for the validation and testing datasets, we resized the image to a fixed size of 224 x 224 pixels to ensure consistency in the input size for the model during validation and testing [15]. We also applied the normalization technique using the same mean and standard deviation normalization as the training set in order to ensure that the validation and test data share the same data distribution for fair evaluation [14].

3.2. Transfer Learning and Fine-Tuning

Transfer learning in image classification is a technique that leverages the knowledge and features learned by a pre-existing neural network, typically trained on a large dataset, to address a new image classification problem [16]. This approach offers several advantages, including increased efficiency and

the ability to achieve strong performance without building a new network from the ground up. To adapt the base model for the new classification problem, we freeze some of its layers such that the weights and parameters of these layers are kept fixed and not updated during training. These frozen layers serve as a feature extractor, providing the new model with a set of meaningful features that are likely to be relevant for the new classification task. On top of the frozen layers, new trainable layers are added on top of the frozen layers which are then trained on the dataset and are responsible for adapting the extracted features to the specific requirements of the new classification task [17]. Fine-tuning is an optional process in which the obtained model is unfrozen and trained again on the data with a significantly reduced learning rate. While this procedure has the potential to enhance the accuracy of the model as fine-tuning helps the model converge to a better local minimum, it also increases the risk of overfitting [18]. In this study, we will explore the effectiveness of fine-tuning for performance improvement and examine the impacts of incorporating additional layers into the base model.

3.3. Data Augmentation

Mixup is a data augmentation technique introduced to combat overfitting and enhance model performance in various tasks including image classification. Mixup works by randomly selecting two images from the training dataset and forming pairs. In Mixup, the key idea is to interpolate both the features and labels of the selected images. For each pair of data samples, a mixing ratio (alpha), which is a value between 0 and 1, is selected. The features of the two images are linearly combined according to this mixing ratio, such that each pixel or feature in one image is mixed with the corresponding pixel or feature in the other image. Similarly, labels are blended by interpolating them based on the mixing ratio. During training, the model learns from both the original and the mixed data samples. Mixup acts as a form of regularization, reducing the risk of overfitting by introducing diversity into the training data. We will investigate whether advanced data augmentation techniques such as Mixup are truly effective in improving the performance of the model.

4. Experiments and Results

All early stopping was set to patience = 8 based on validation accuracy (except for few-shot which uses validation loss) and restore_best_weights was set to True, such that the final model will take on the weights that resulted in the highest validation accuracy. The training accuracy and loss, and validation accuracy and loss values below will take on the values at the point where the model stopped training due to early stopping. The model updates its optimal weights only if the current validation accuracy surpasses the highest validation accuracy achieved so far.

4.1. ResNet-50

The ResNet-50 is a CNN architecture relatively deep neural network consisting of 50 layers. The core innovation in ResNet-50 is the use of residual blocks, which help address the vanishing gradient problem. Each residual block contains two main paths: the identity path, which is a shortcut connection, and the main path which is a series of convolutional layers. ResNet-50 employs a bottleneck structure in its residual blocks, where each block includes three sets of convolutional layers, 1x1, 3x3, and 1x1. The 1x1 convolutions reduce and then increase the dimensions, acting as bottleneck layers, which are computationally more efficient, while the 3x3 convolution handles the feature extraction. He et al. popularized the approach of using skip connections which allows the gradient to flow more directly through the network [2].

Our initial step was to compare the effect of data augmentation on the performance of ResNet, with the details summarised in Table 4.1.1 below, and the plots for training and validation accuracy, and training and validation loss are placed in Appendix 7.1. We observed that the model without data augmentation achieved 100.00%, which indicates perfect accuracy on the training dataset, suggesting that the model may have overfitted to the training data, thus compromising its ability to generalize. Conversely, introducing data augmentation appeared to mitigate overfitting, as indicated by a reduced training accuracy, while also enhancing the model's performance on unseen data, which was reflected by higher validation and test accuracy scores.

Subsequently, we explored how incorporating mixup, as an extra step of data augmentation described in Section 3.3, affects model performance. The results (Table 4.1.1) show that while the validation accuracy dipped slightly and the test accuracy increased marginally against the baseline, the F1 score experienced a slight drop. This implies that mixup does not drastically outperform conventional data augmentation methods in this scenario.

Furthermore, we experimented to determine if a more complicated classification layer would elevate the performance of our model. Following the same preprocessing protocols as the ResNet-50 model with data augmentation, which surpassed both the plain ResNet-50 and the version with the mixup, we observed that this complexity did not translate to enhanced performance. In fact, the validation accuracy is slightly lower, and the test accuracy is noticeably lower compared to the baseline, along with the lowest F1 score among all the models, suggesting that adding complexity to the classification layer did not result in performance benefits and might have introduced difficulties for the model to generalize. Lastly, as evident in Table 4.1.1, the model with fine-tuning shows the most promising results with significant improvements in generalization as it has the highest training accuracy among all the data augmentation models and it demonstrates the highest validation and test accuracies as well as the highest F1 score.

Model	**Train Accuracy**	**Train Loss**	**Validation Accuracy**	**Test Accuracy**	**F1 Score**
ResNet-50 without Data Augmentation	100.00%	0.0471	84.61%	82.29%	0.824
ResNet-50 with Data Augmentation	95.78%	0.267	85.69%	84.52%	0.845
ResNet-50 with Data Augmentation and Mixup	86.68%	0.920	83.33%	82.79%	0.829
ResNet-50 with Data Augmentation & Complex Classification Layer	93.43%	0.276	82.94%	81.51%	0.815
ResNet-50 with Data Augmentation & Fine-tuning	98.33%	0.0650	90.49%	88.13%	0.882

Table 4.1.1: Summary of train, validation, and test accuracy, train loss, and F1 score of Resnet with and without data augmentation.

4.2. MobileNet V2

MobileNet V2 is a deep convolutional neural network architecture designed for efficient and lightweight deep learning applications, particularly on mobile and embedded devices [19]. Its key features include inverted residuals, which expand input to a higher-dimensional space, convolve, and then project back down, preventing information loss [20]. Depthwise separable convolutions, splitting a standard convolution into depthwise and pointwise convolutions, significantly reduce computational costs [21]. Linear bottlenecks maintain information flow during these convolutions. Introducing width and resolution multipliers allows users to customize the model size and computational complexity. The width multiplier controls channel numbers, and the resolution multiplier scales down the input image resolution. The core building block, the inverted residual block, encompasses depth-wise separable convolutions, linear bottlenecks, and shortcut connections, aiding gradient flow and faster convergence. Efficient building blocks like ReLU6 activation functions and batch normalization enhance model efficiency and training stability [22]. Global average pooling at the end of the network reduces spatial dimensions, providing a fixed-size feature vector for classification.

We performed similar steps as ResNet-50 (Section 4.1) for the MobileNet V2. Firstly, we experimented to determine if data augmentation would improve the performance of the model. The results are presented in Table 4.2.1, and the plots for training and validation accuracy, and training and validation loss, as well as the confusion matrices, are placed in Appendix 7.2. The MobileNet model trained without data augmentation exhibited perfect training accuracy, which likely signifies that it memorized the training data rather than learning to generalize, as evidenced by its comparatively lower performance on unseen validation and test data. In contrast, the MobileNet model trained with data augmentation demonstrated a commendable balance between learning from the training data and generalizing to new data, reflected in its consistent validation and test performance and a solid F1 score. In the confusion matrices for MobileNet V2 models (Section 7.2), we can generally observe that there are very faint colors off the diagonal, indicating that there are some misclassifications, but they are relatively few compared to the correct predictions.

When mixup was incorporated into the data augmentation process, the model's training accuracy dipped notably, possibly due to the heightened challenge of learning from the more complex data. However, this did not translate into improved generalization, as both test performance and F1 score decreased, indicating that mixup might not be advantageous in this setting.

Furthermore, to determine if the complexity of the classification head will affect the performance of the model, we build two other models, MobileNet Transfer with a simple classification layer, and MobileNet Transfer with a complex custom classification layer to compare the performances with the default MobileNet model. The model with a simple classification layer achieved validation and test results that were marginally better, suggesting that simplifying the classification layer does not significantly impact the model's ability to generalize. Conversely, the model with a complex classification layer underperformed across training, validation, and test metrics, achieving the lowest F1 score, hinting that the additional complexity could be detrimental to learning efficiency.

Finally, the fine-tuned MobileNet model achieved impressive training accuracy without hitting the perfect score, indicating a robust learning process. It stood out with the highest validation and test accuracies and an excellent F1 score, underscoring that fine-tuning considerably enhances the model's ability to generalize effectively to new data.

Model	Train Accuracy	Train Loss	Validation Accuracy	Test Accuracy	F1 Score
MobileNet Transfer without data augmentation	100%	0.109	83.24%	81.56%	0.816
MobileNet Transfer with data augmentation	92.75%	0.415	83.33%	81.28%	0.811
MobileNet Transfer with data augmentation and mixup	81.27%	1.201	83.24%	80.13%	0.801
MobileNet Transfer with simple classification layer	93.14%	0.476	82.84%	80.44%	0.807
MobileNet Transfer with data augmentation and complex classification layer	88.04%	0.450	80.00%	77.80%	0.774
MobileNet Transfer with fine-tuning	96.37%	0.135	87.35%	84.92%	0.850

Table 4.2.1: Summary of the train, validation, and test accuracy, train loss, and F1 score of MobileNet V2 with and without data augmentation.

4.3. Transformers ViT

In Vision Transformers, the initial step involves dividing the input image into small segments, similar to how sentences are broken into words in natural language processing. These segments, typically 16x16 pixels, are then converted into one-dimensional vectors and processed through a linear layer to adjust their dimensionality for the transformer [24]. In addition to these patch embeddings, a special embedding, often referred to as the classification token, denoted as [CLS], is prepended to this sequence. This token is not part of the actual image data but is a learnable vector and the purpose of the [CLS] token is to aggregate or capture the global information from the entire image as it passes through the Transformer layers. To compensate for the lack of inherent spatial awareness in Transformers, positional encodings are incorporated into the patch embeddings, providing the model with information about the relative or absolute locations of these image segments. The core of the ViT is a series of transformer encoder layers, which consist of two primary elements, the Self-Attention Mechanism and a Feed-Forward Neural Network. The Self-Attention Mechanism allows the model to weigh the importance of different patches when processing each patch and helps the model to understand the image contextually by considering how each patch relates to others. The Feed-Forward Neural Network is applied after the attention mechanism and helps in further processing the information. After processing through the Transformer encoder, only the transformed [CLS] token at the top of the sequence will be passed to the classification head, as the [CLS] token has embedded the contextual information from the entire image. The final output of the classification head is typically passed through a softmax function, which converts the raw scores in the output vector into probabilities. In our implementation, we employed the base-sized ViT model “google/vit-base-patch16-224-in21k” [25].

Similar to the two other models in Section 4.1 and 4.2, our experimentation on this ViT is to determine if data augmentation will improve the performance of this pre-trained model. The results are summarised in Table 4.4.1 below. The model without data augmentation achieved a perfect training accuracy of 100.00%, while the model with data augmentation had a slightly lower training accuracy of 97.75%. This

suggests that data augmentation introduced more variety and complexity to the training process, preventing the model from reaching perfect accuracy on the training set. This can be a sign of reduced overfitting. Both confusion matrices in Table 4.4.2 show high values along the diagonal, indicating that both models performed well in correctly classifying the test samples. The F1 score was slightly higher for the model without data augmentation (0.971) compared to the model with it (0.968). One possible reason could be that the model was overfitting on the training dataset, as evident from the training accuracy being 100.00%, therefore leading to a slightly higher accuracy as compared to the model with data augmentation, whereas the model with data augmentation shows a slightly lower training accuracy (97.75%), which shows an improvement in generalization to unseen data. While the Vision Transformer model already performs well without data augmentation, the practice of data augmentation seems to provide a slight improvement in generalization to unseen data without compromising the model's confidence in its predictions as Transformers rely on self-attention mechanisms and can sometimes be data-hungry.

Model	Train Accuracy	Train Loss	Validation Accuracy	Test Accuracy	F1 Score
ViT without Data Augmentation	100.00%	0.154	97.84%	97.11%	0.971
ViT with Data Augmentation	97.75%	0.291	97.35%	96.81%	0.968

Table 4.4.1: Summary of test accuracy, test loss, and F1 score of Transformers ViT with and without data augmentation.

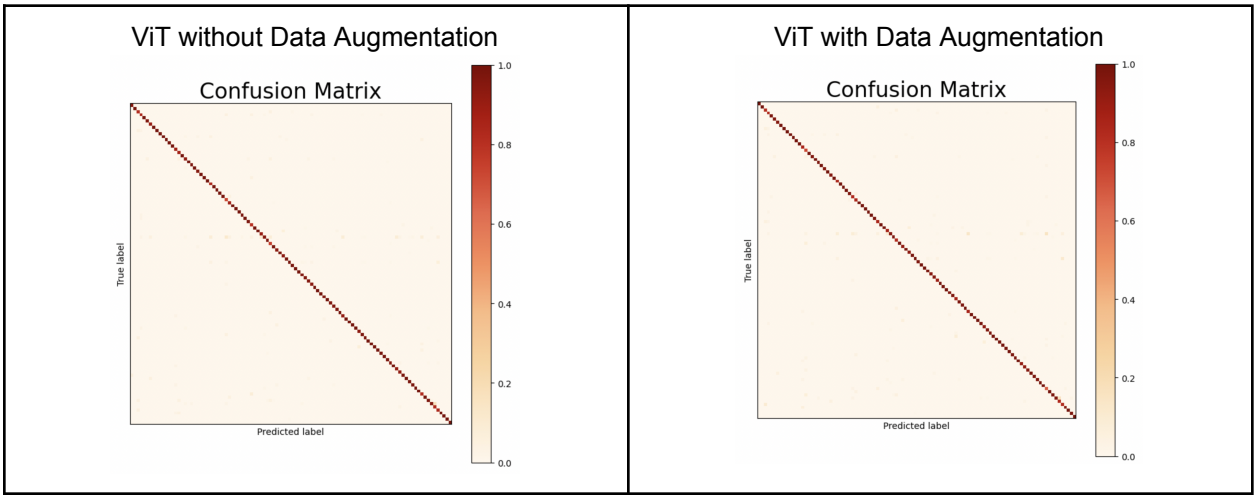


Table 4.4.2: Confusion Matrix for ViT with (right) and without (left) the implementation of Data Augmentation.

4.4. K-Shot Learning (K = 5)

K-shot learning is a concept within the framework of N-shot learning, where a model is designed to learn from a very limited amount of data [23]. The term "K-shot" refers to the number of examples per class that the model has access to during training. In our implementation, we set k to be 5, which means that the model is provided with only five images for each of the 102 flower classes. We make use of a Siamese net to help learn the important features of each class. As the amount of training data is severely limited, it makes the learning task much more challenging compared to traditional methods where large amounts of data are available. Due to the pre-training on large datasets like ImageNet, we implemented MobileNet

V2 as the backbone of our Siamese network. Furthermore, the architecture of MobileNet V2, with depthwise separable convolutions, helps in generalizing well even with limited training data [21]. This is critical in k-shot learning scenarios, where the model needs to make the most out of a few training samples.

The process of our 5-Shot Learning is as follows. Firstly, we randomly sample 5 images from each class in the training set. Next, we generate pairs at random with equal probability to be either a positive pair or a negative pair. Positive Pairs are pairs of the same class and negative pairs are pairs of the opposing class. Next, we train our Siamese network with contrastive loss as our loss function. Once the Siamese network is trained, we will pass the image pairs to get the embedding and calculate the pairwise distance of each embedding. Setting our threshold at 0.5, any pairs that have a distance greater than 0.5 are deemed as different, and any pairs with less than 0.5 are deemed as similar. Lastly, we try to extend the idea to do a full classification task. The idea is that given a support set (in this case our training set), if the image in question has the lowest cumulative distance in all classes, then that image must belong in that class.

Model	Train Set F1 (Determine Similarity)	Validation Set F1 (Determine Similarity)	Test Set F1 (Determine Similarity)	F1 Score Classification task (train set support, test set query)
Siamese Net with MobileNet Backbone	0.860	0.828	0.791	0.443

Table 4.3.1: Summary of test accuracy, test loss, and F1 score of K-shot learning with and without data augmentation.

We can observe that while the model is good at distinguishing image pairs, the idea that it can be extended to carry out image classification may not be very viable. One reason for this could be that certain flowers look very similar despite being a different class. In Table 4.3.2 below we can see that when the model classifies badly such as class 38, the support set (center row images) is very similar to the query set and also the predicted image. However, in the situation where the model performs well, the support set is only very similar to the query set but very different from the mistakes that it makes. This could imply that the model is able to pick up more general details about a class but not to a fine enough level to distinguish 2 very similar classes which is to be expected given our very limited training data.

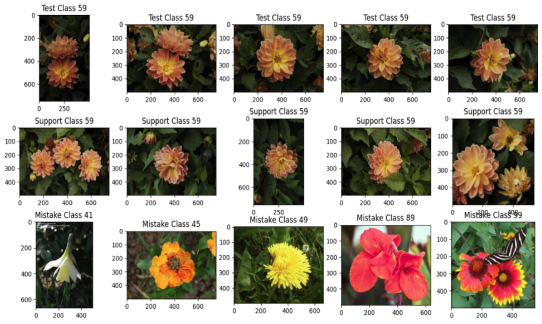
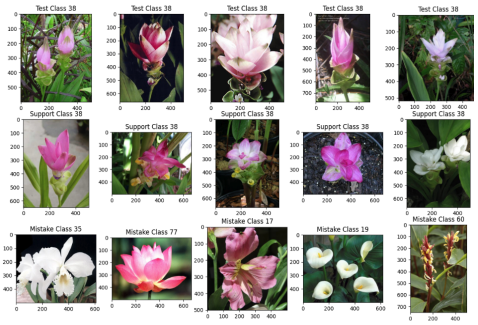
Class 59	Class 38
F1 Score = 0.94	F1 Score = 0.09
	

Table 4.3.2: Examples of one class that performs well (left) and another class that performs badly (right)

4.5. Triplet Loss Network (Resnet backbone + Classifier & Embeddings output)

The triplet loss network takes three input images, anchor, positive, and negative. These images are chosen to create a triplet, where the anchor and positive images belong to the same class, while the negative image is from a different class. These three images are passed into a backbone model, ResNet-50 in our case. The reason for our choice of ResNet-50 as the backbone model is that it performs well compared to other models like MobileNet V2. The final representation of the image in the final ResNet-50 convolutional layer is a 7 x 7 x 2048 representation. Next, we apply average pooling on this representation to obtain a 1 x 1 x 2048 convolution layer. This feature mapping will then be used to generate the embeddings used in a Triplet Loss Function as well as the predicted labels used in classification. The idea behind creating the model was inspired by the study conducted by Taha et al. [6]. Although our implementation uses a smaller embedding size (64 vs 256) and a more simplified optimizer (Adam with constant lr of 0.001 vs SGD with momentum 0.9 and decaying learning rate). The reason for these changes is so that we have a similar baseline to compare our models which all use Adam optimizer and to compare how the difference in performance is compared to our Siamese network without a dedicated classification branch.

In this experiment, we will investigate the effect of data augmentation on the performance of the Triplet Loss Network.

Model	**Train Accuracy**	**Train Loss**	**Validation Accuracy**	**Test Accuracy**	**F1 Score**
Triplet Loss Network without Data Augmentation on Train Set	100.00%	0.07	83.14%	81.33%	0.811
Triplet Loss Network with Data Augmentation on Train Set	97.65%	0.33	84.02%	82.9%	0.829

Table: Summary of test accuracy, test loss, and F1 score of Triple Loss Network before and after simple data augmentation.

This experiment reinforces the findings of earlier ones, demonstrating that data augmentation plays a crucial role in minimizing overfitting and enhancing the model's ability to generalize. Additionally, we can also observe that by introducing a classification branch, we can significantly boost the testing F1 Score compared to simply using a Siamese network and relying on the distance as a measure of class. While the two networks use different backbones, it is to be noted that both MobileNet V2 and ResNet-50 achieved similar F1 scores on this dataset. As such while the different backbones may have a slight part to play in the difference in performance, it is probably not the major cause for it.

Looking at how this network performs against the base ResNet-50, we can observe that both networks underperform the base ResNet-50 networks by 1-2% in their respective categories (i.e. with or without data augmentation). One possible reason could be related to the fact observed in our Siamese network that image embeddings may not be the best thing to aid in this classification task. This is especially so when we can observe certain flower classes that look similar but have a different class. As such having a triplet loss added to the regular cross entropy loss could have tuned the model to perform worse for classification in the hopes of getting a better embedding.

5. Conclusion

In this report we have explored 5 different models and evaluated their performances on the Flowers102 dataset. The Flowers102 dataset is a particularly challenging dataset since we have limited amounts of training data and a large amount of testing data.

MobileNet V2 and Resnet-50 are our explorations into the well-established CNN models that are used in many computer vision tasks. For both models, we used transfer learning to save time on training and both models are able to perform decently well on their own. However, it can be observed that both models are also overfitting to the training data when no data augmentation was in place. Hence, just by adding in a simple augmentation step (random flips and rotations), we are able to achieve slightly better results and prevent the model from overfitting. We also explored mixup augmentation which does not seem to help improve the model's classification performance for this dataset. Lastly, we also explored how the depth of the final classification layer will affect the model's performance and came to the conclusion that the default architectures (which are fairly shallow) seemed to perform the best. After fine-tuning and using the best parameters for training, we discovered that for this dataset Resnet-50 with data augmentation and no mixup augmentation performs best for our traditional CNN models with a testing accuracy of 88%

Visual Transformers represents a paradigm shift in how we can carry out computer vision tasks. Instead of using convolutions to process an image, it makes use of self-attention mechanisms that are inspired by the field of natural language processing. This novel approach is able to perform far better than any of our CNN models, achieving accuracies of 97% on the test set. It is also to be noted that data augmentation seems to play a less important role here since despite the model getting 100% accuracy on the training set, its performance on the validation and testing sets is not far from the training accuracy.

Lastly, we explored a more unconventional technique to try to classify this Flowers102 dataset in the form of Siamese Networks. We explored two methods to go about this task, the first revolves around the idea of few-shot learning where we train a Siamese network to identify patterns of different classes and use a pairwise distance to determine if 2 classes are the same. The ability of the network to distinguish between 2 different tasks was good achieving a test f1 score of around 0.79. However, we found out that just because a model is able to tell 2 classes apart does not necessarily imply that it is able to accurately classify different classes. Hence, we explored another method that made use of multiple outputs and a combination of triplet loss and cross-entropy loss to train a model to do both embedding and classification. The idea is that if I am able to teach a model to separate classes via embedding, this learned approach can aid the classifier to perform better. While there was a sharp improvement in the model's classification performance against the attempted few shot learning, it still under performed all our other models tested. However, one thing positive about this model is that we are able to get both an image embedding representation and achieve decent accuracy for classification all in one model and without any noticeable increase in inference time.

6. References

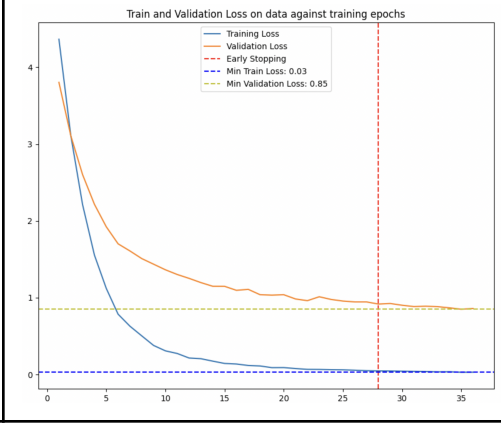
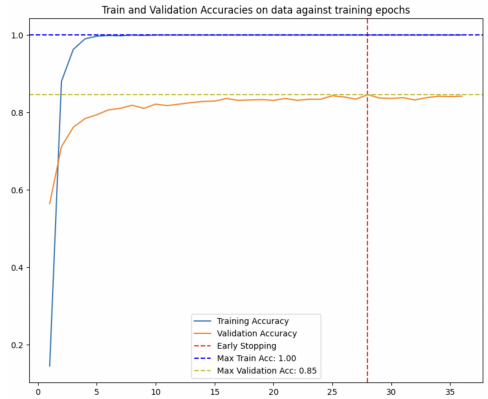
1. Nilsback, M.-E., & Zisserman, A. (n.d.). 102 category Flower Dataset. Visual Geometry Group - University of Oxford. https://www.robots.ox.ac.uk/~vgg/data/flowers/102/
2. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2016.90
3. Girish, S., Dey, D., Joshi, N., Vineet, V., Shah, S., Mendes, C. C. T., Shrivastava, A., & Song, Y. (2022, March 15). *One network doesn't Rule them all: Moving beyond handcrafted architectures in self-supervised learning*. arXiv.org. https://doi.org/10.48550/arXiv.2203.08130
4. Wightman, R., Touvron, H., & Jégou, H. (2021, October 1). *Resnet strikes back: An improved training procedure in Timm*. arXiv.org. https://arxiv.org/abs/2110.00476
5. Xiexing Feng a b, Q.M. Jonathan Wu b, Yimin Yang c, Libo Cao a. (2021, September 17). *A compensation-based optimization strategy for top dense layer training*. Science Direct. https://doi.org/10.1016/j.neucom.2020.07.127
6. Taha, A., Chen, Y.-T., Misu, T., Shrivastava, A., & Davis, L. (2020, March 2). *Boosting standard classification architectures through a ranking regularizer*. arXiv.org. https://arxiv.org/abs/1901.08616#:~:text=Boosting%20Standard%20Classification%20Architectures%20Through%20a%20Ranking%20Regularizer,-Ahmed%20Taha%2C%20Yi&text=We%20employ%20triplet%20loss%20as,with%20minimal%20hyper%2Dparameter%20tuning.
7. F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In CVPR, 2015.
8. Hugo Touvron, Matthieu Cord† Alaaeldin El-Nouby, Jakob Verbeek Herve J ´ egou ´. (n.d.). *Three things everyone should know about vision transformers - arxiv.org*. Arvix. https://arxiv.org/pdf/2203.09795.pdf
9. Chen, X., Hsieh, C.-J., & Gong, B. (2022, March 13). *When Vision Transformers outperform resnets without pre-training or strong data augmentations*. arXiv.org. https://arxiv.org/abs/2106.01548
10. *Flowers102*. Flowers102 - Torchvision main documentation. (n.d.). https://pytorch.org/vision/main/generated/torchvision.datasets.Flowers102.html
11. *Randomrotation*. RandomRotation - Torchvision main documentation. (n.d.-a). https://pytorch.org/vision/main/generated/torchvision.transforms.RandomRotation.html
12. *Randomresizedcrop*. RandomResizedCrop - Torchvision main documentation. (n.d.). https://pytorch.org/vision/main/generated/torchvision.transforms.RandomResizedCrop.html
13. *Randomhorizontalflip*. RandomHorizontalFlip - Torchvision 0.16 documentation. (n.d.). https://pytorch.org/vision/stable/generated/torchvision.transforms.RandomHorizontalFlip.html
14. *Normalize*. Normalize - Torchvision main documentation. (n.d.). https://pytorch.org/vision/main/generated/torchvision.transforms.Normalize.html
15. *Resize*. Resize - Torchvision main documentation. (n.d.). https://pytorch.org/vision/main/generated/torchvision.transforms.Resize.html
16. Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? In Advances in neural information processing systems (pp. 3320-3328).
17. Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359.
18. Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1717-1724).

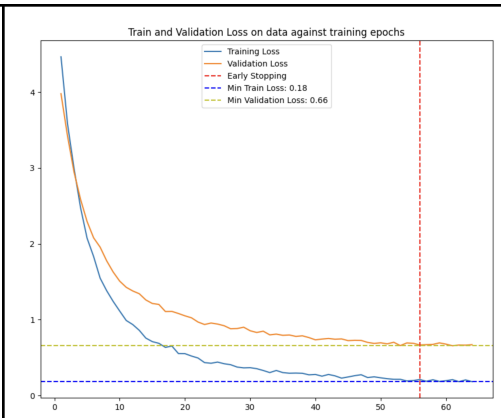
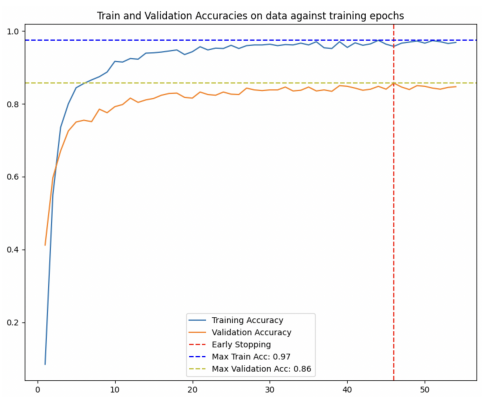
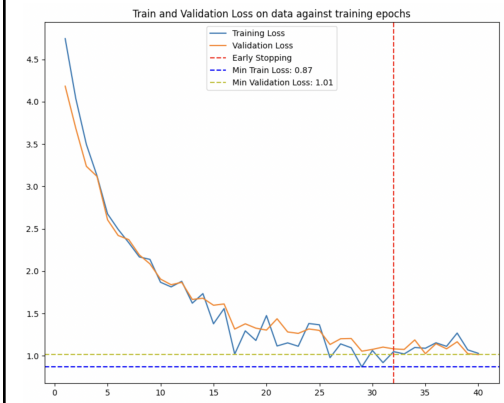
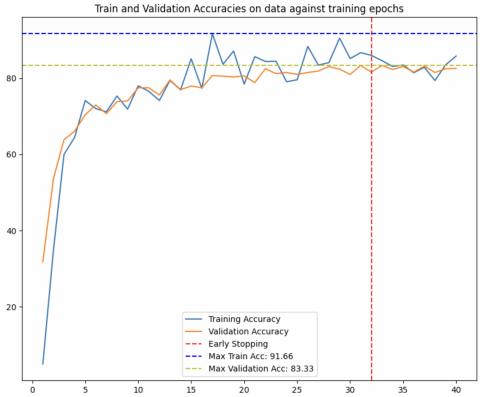
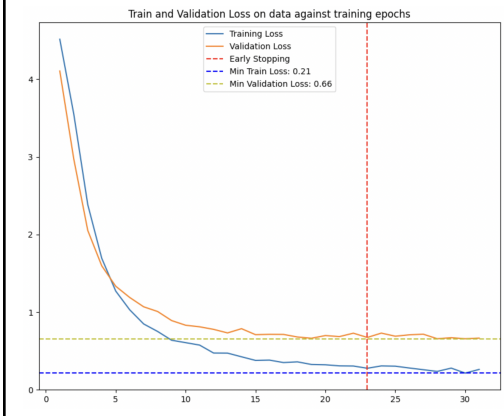
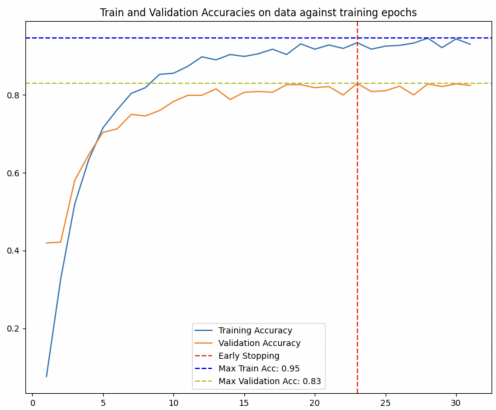
19. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861.
20. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
21. Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR).
22. Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the International Conference on Machine Learning (ICML).
23. Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., & Wierstra, D. (2016). Matching Networks for One Shot Learning. In Advances in Neural Information Processing Systems (NeurIPS).
24. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021, June 3). *An image is worth 16x16 words: Transformers for image recognition at scale*. arXiv.org. https://arxiv.org/abs/2010.11929
25. Wu, B., Xu, C., Dai, X., Wan, A., Zhang, P., Yan, Z., Tomizuka, M., Gonzalez, J., Keutzer, K., & Vajda, P. (2020, November 20). *Visual transformers: Token-based image representation and processing for Computer Vision*. arXiv.org. https://arxiv.org/abs/2006.03677

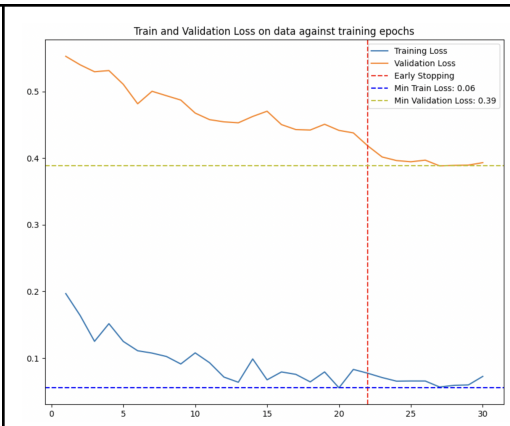
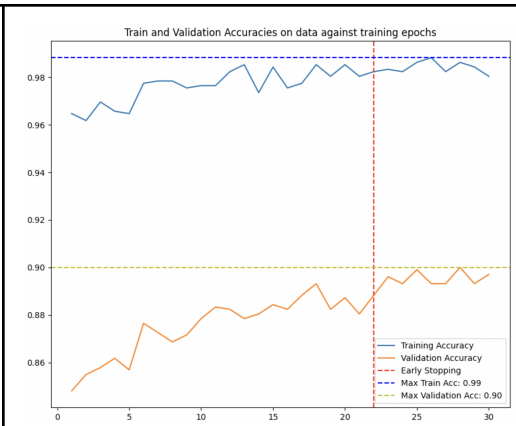
7. Appendix

7.1. ResNet-50

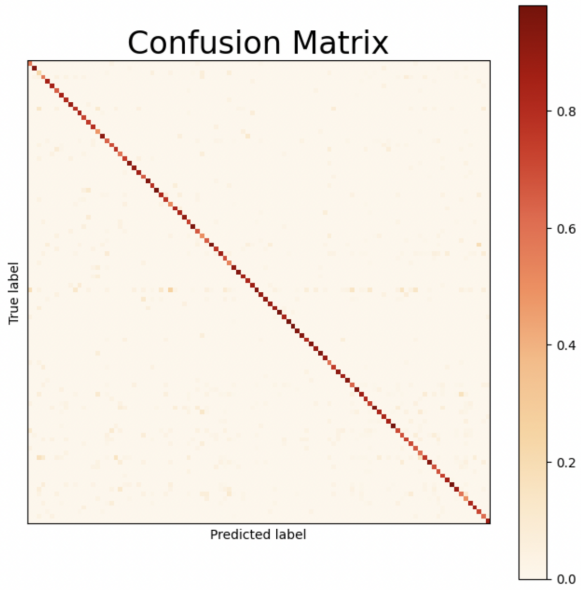
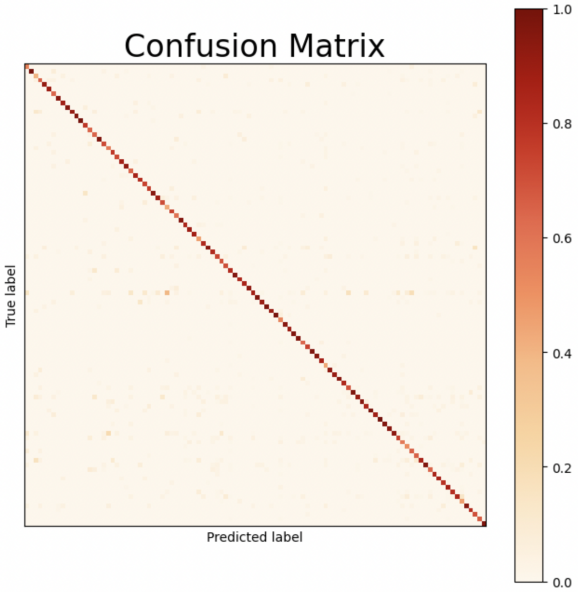
7.1.1. Training Plots for ResNet-50

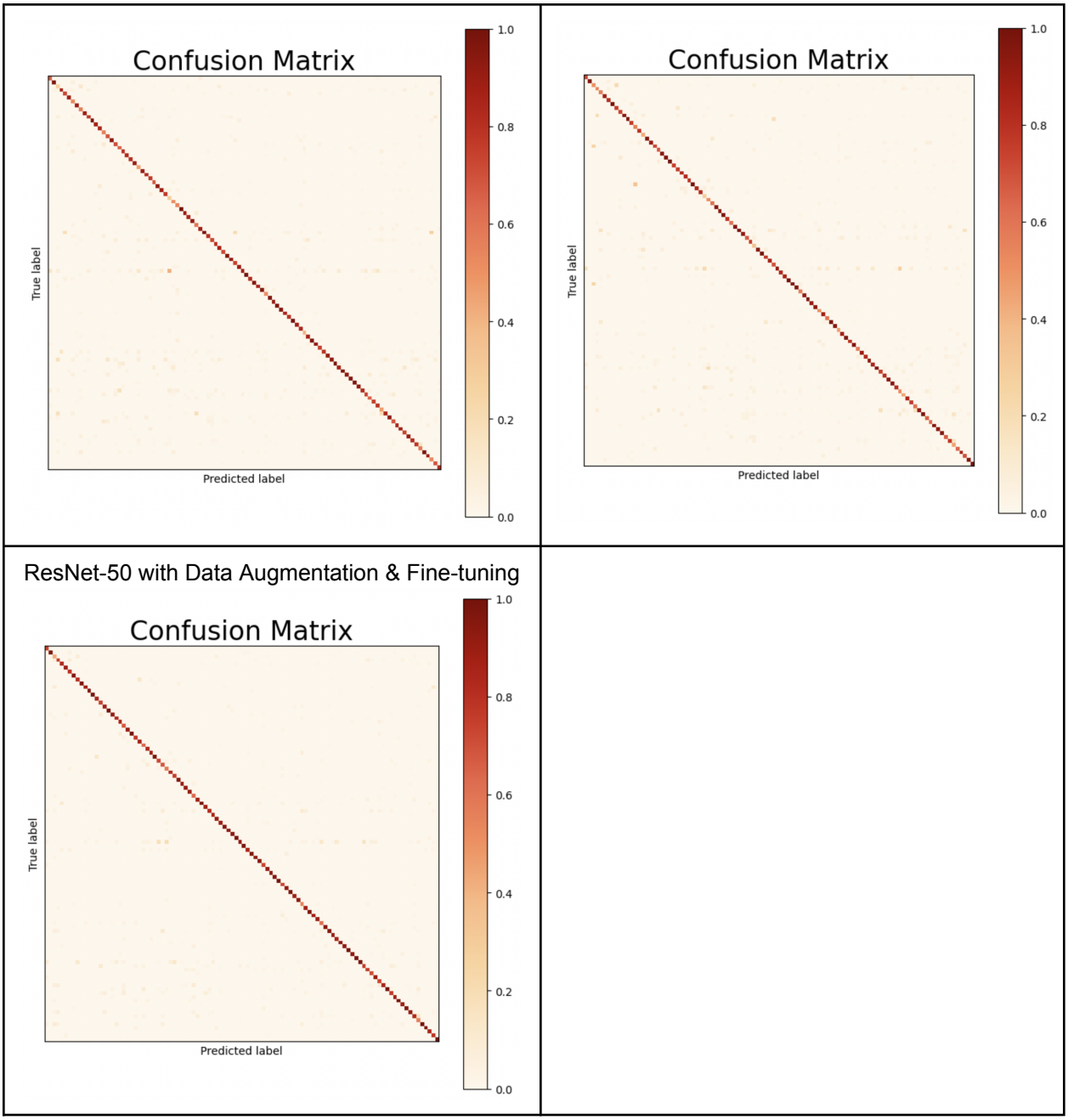
Model	Loss	Accuracy
ResNet-50 without Data Augmentation		

ResNet-50 with Data Augmentation		
ResNet-50 with Data Augmentation and Mixup		
ResNet-50 with Data Augmentation & Complex Classification Layer		

ResNet-50 with Data Augmentation & Fine-tuning		

7.1.2 Confusion Matrices for ResNet-50

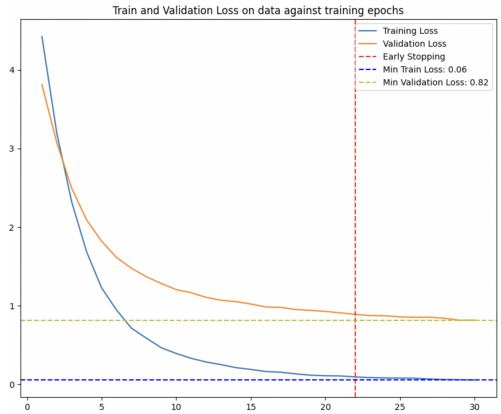
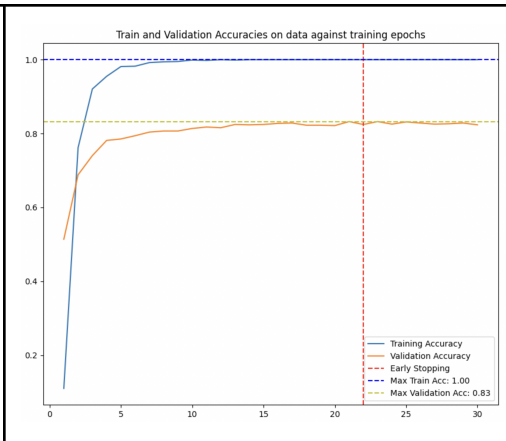
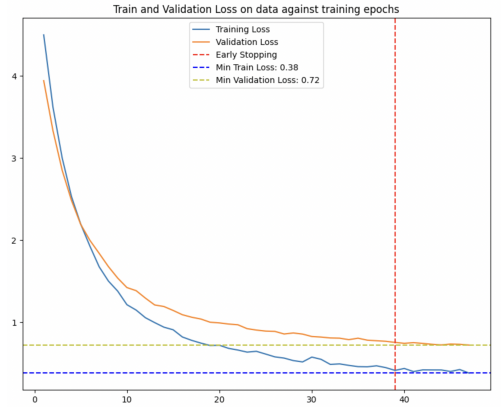
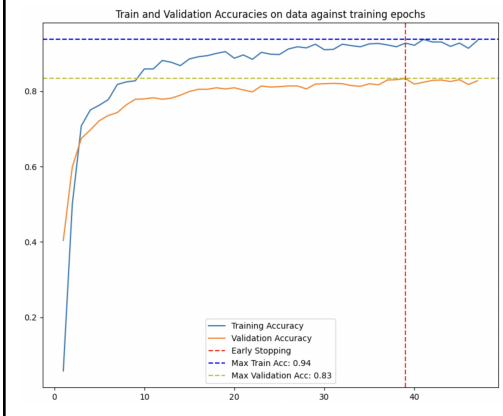
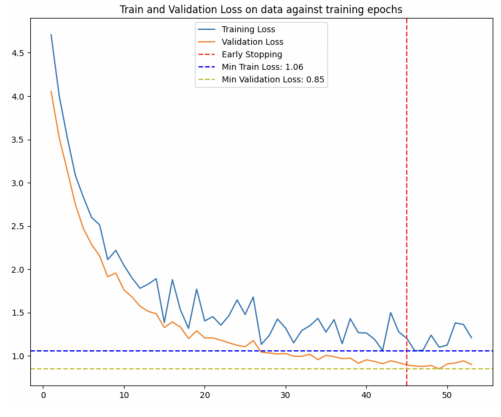
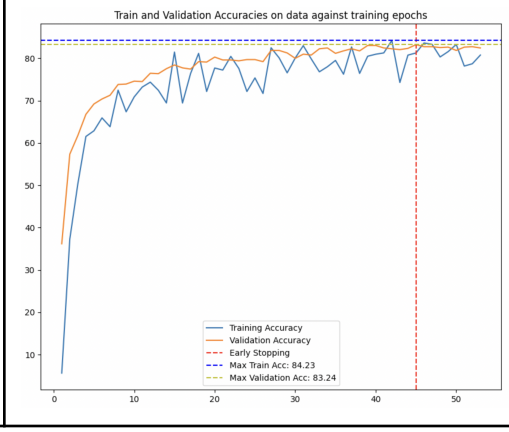
ResNet-50 without Data Augmentation 	ResNet-50 with Data Augmentation 
ResNet-50 with Data Augmentation and Mixup	ResNet-50 with Data Augmentation & Complex Classification Layer

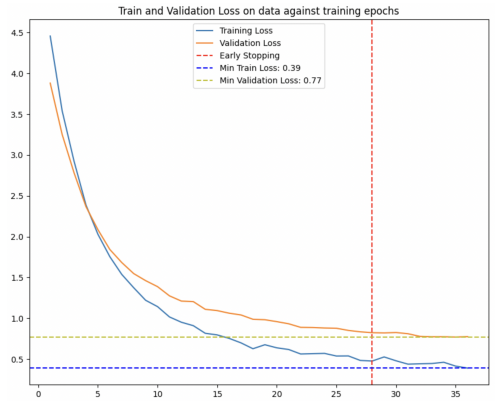
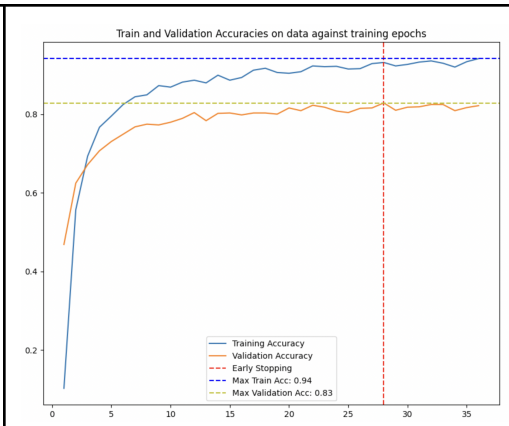
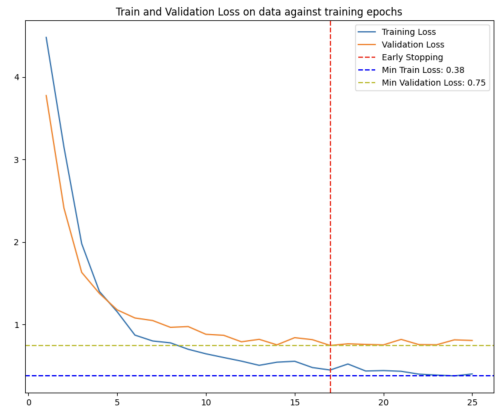
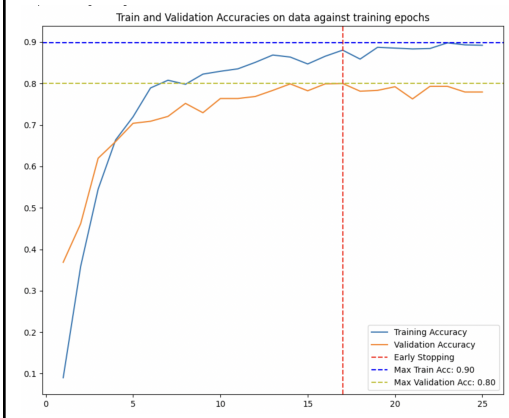
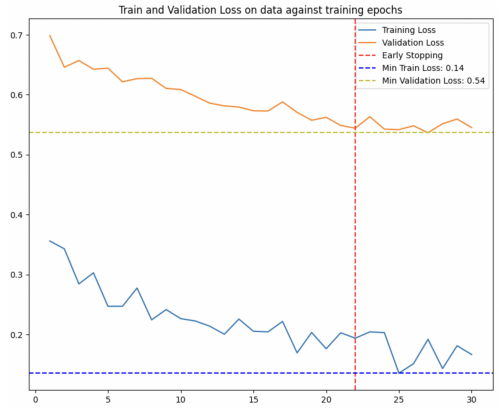


7.2. MobileNet V2

7.2.1. Training Plots for MobileNet V2

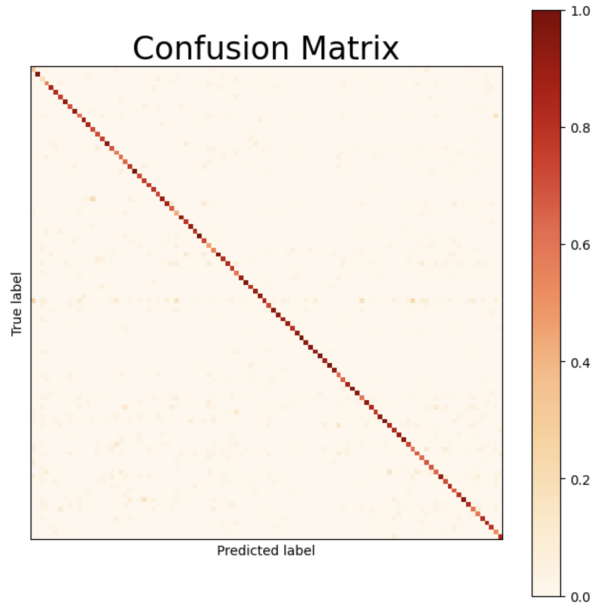
Model	Loss	Accuracy

MobileNet Transfer without data augmentation		
MobileNet Transfer with data augmentation		
MobileNet Transfer with data augmentation and mixup		

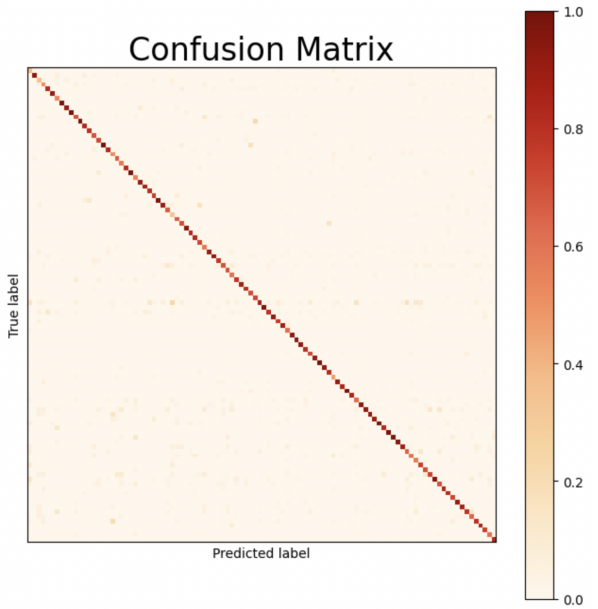
MobileNet Transfer with simple classification layer		
MobileNet Transfer with data augmentation and complex classification layer		
MobileNet Transfer with fine-tuning		

7.2.2. Confusion Matrices for MobileNet V2

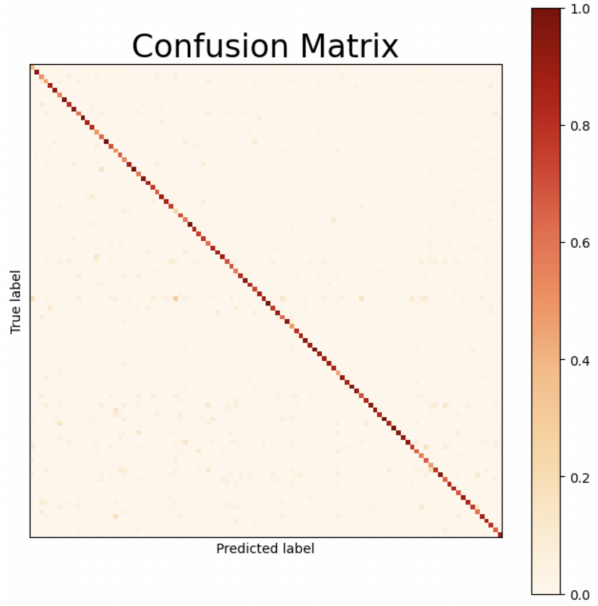
MobileNet Transfer without data augmentation	MobileNet Transfer with data augmentation



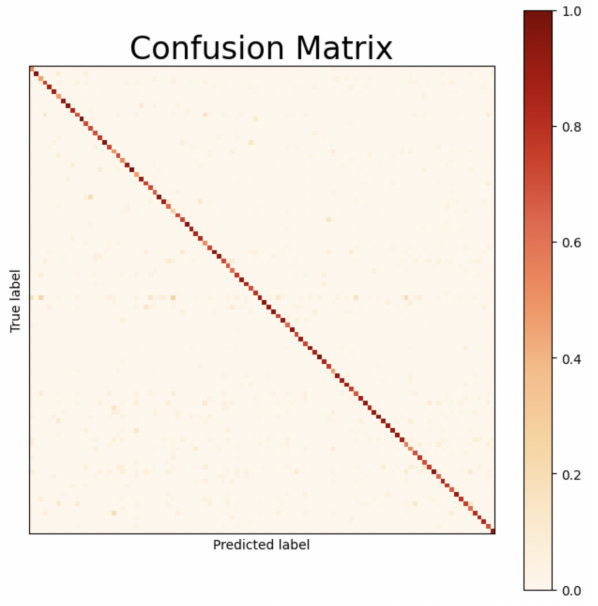
MobileNet Transfer with data augmentation and mixup



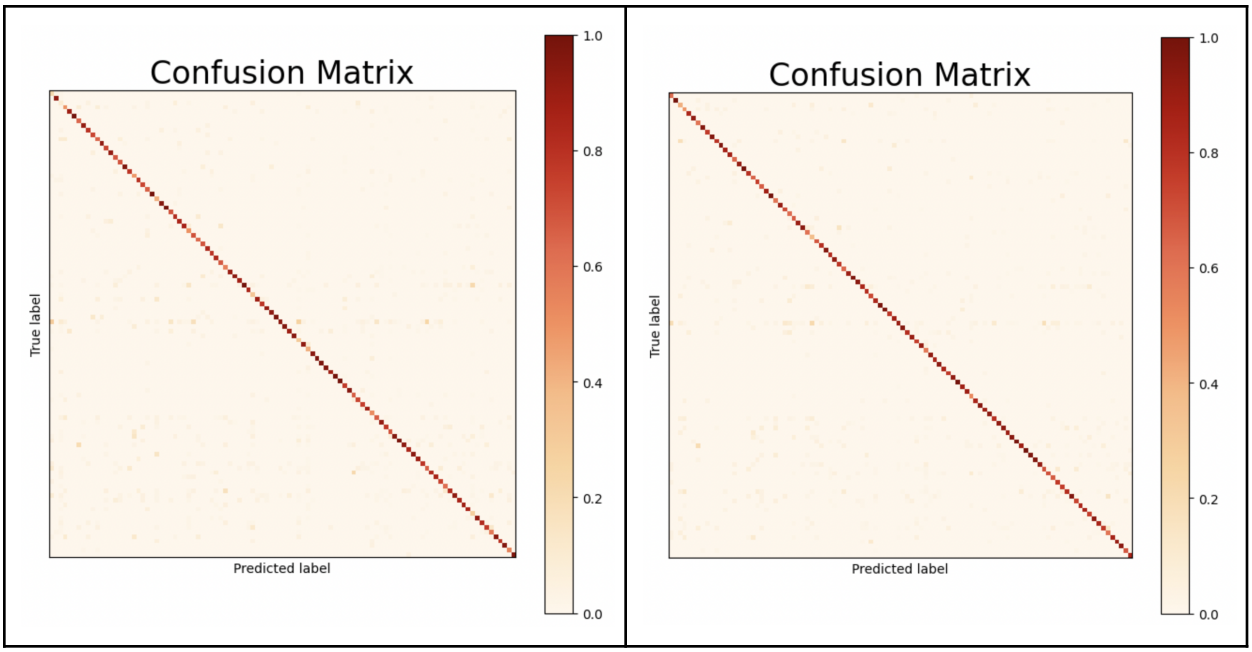
MobileNet Transfer with simple classification layer



MobileNet Transfer with data augmentation and complex classification layer

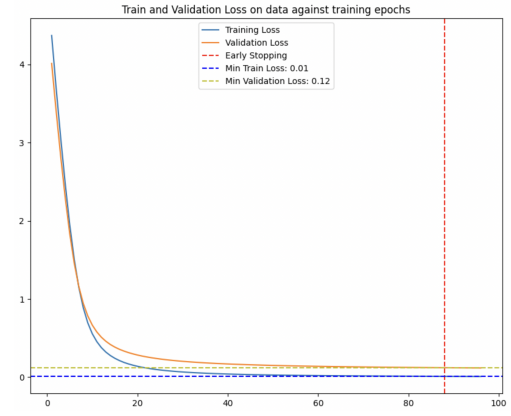
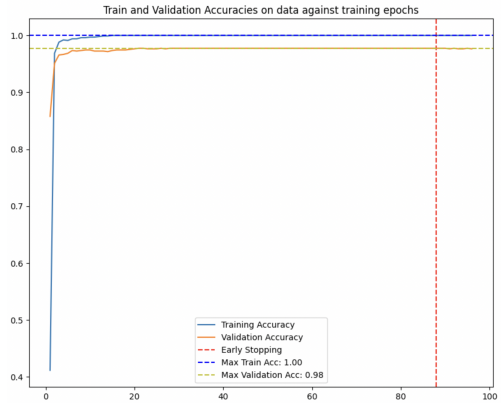


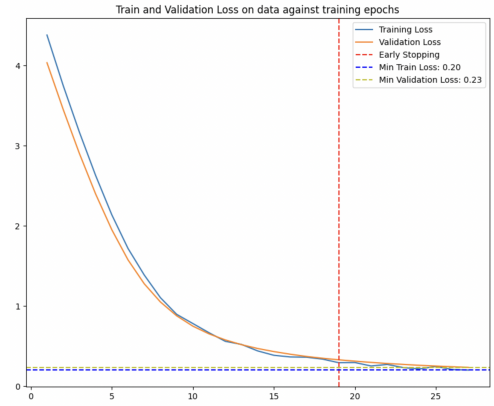
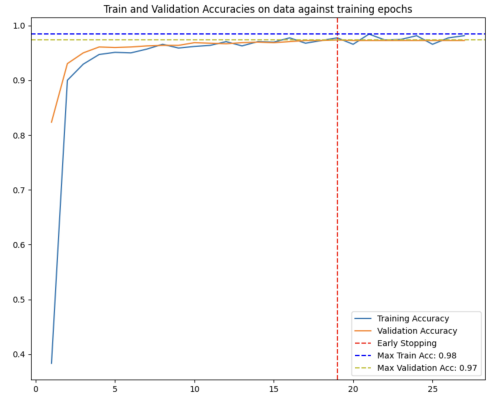
MobileNet Transfer with fine-tuning



7.3. Transformers ViT

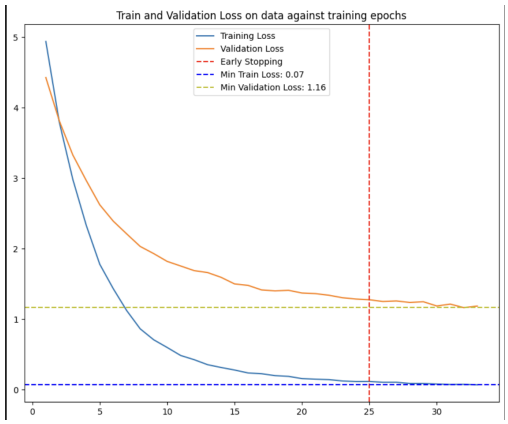
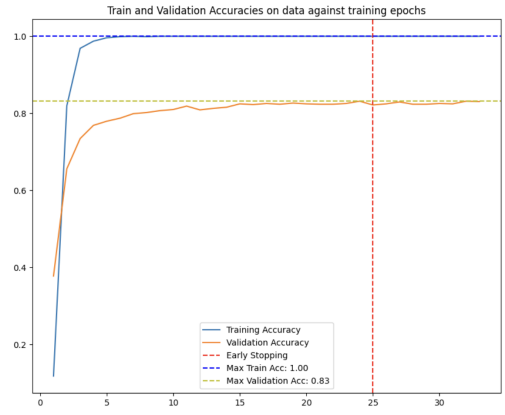
7.3.1. Training Plots for Transformers ViT

Model	Loss	Accuracy
Transformers ViT without Data Augmentation		

Transformers ViT with Data Augmentation		

7.4. Triplet Loss Network

7.4.1. Training Plots for Triplet Loss Network

Model	Loss	Accuracy
Triplet Loss Network without Data Augmentation		
Triplet Loss Network with Data Augmentation