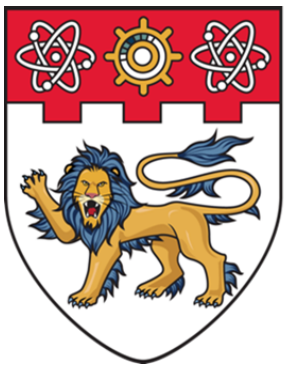


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4042 Neural Networks and Deep Learning

Group Project Report

2022-23 Semester 1

Flower Recognition:

Dealing with Less Data via Few-Shot Learning

Group Members:

Agarwala Pratham (U2023384F)

George Rahul (U2023835C)

Chopra Dhruv (U2023974A)

Table of Contents

1. Introduction

Computer Vision is an ever growing field with a plethora of interesting problems to tackle ranging from object detection to image segmentation. For decades traditional machine learning was used to tackle these problems (and have done so quite well) but the emergence of deep learning based approaches have allowed us to reach groundbreaking results. Intrigued by this, our team selected the classic problem of image recognition and explored the current deep learning models for this task.

1.1 Problem Statement

Image recognition is the process of identifying an object or a feature in an image or video. Existing methods work by learning features and patterns in the training samples and use these to perform a multi-class classification. Convolutional Neural Networks were the forerunners in this field with modern approaches leveraging the attention mechanism of the transformer architecture to generate encodings for the input images.

However it is well known these models are quite data hungry, requiring a large number of training samples per class. This poses a challenge in domains where labeling data is expensive or sufficient data does not exist. With the aim of tackling this problem our team decided to delve deep into the problem of small datasets and developing clever architectures to overcome them.

1.2 Literature Review

The problem of learning with insufficient training examples has been explored extensively in the statistical and machine learning domains. This problem is especially relevant for deep learning methods, as these algorithms are characteristically data hungry. One of the solutions to this problem is the use of deep transfer learning, where the latent knowledge of one problem domain learned by the deep networks can be utilized to solve the problems in another domain (Tan et al., 2018).

Another approach to tackle the insufficient training data problem is the use of Siamese networks. The first Siamese network was created for signature verification by comparing images (Bromley et al., 1993). Later, a contrastive energy function was proposed to increase the difference between dissimilar pairs and decrease the difference between similar pairs (LeCun et al., 2005). A seminal paper by Facebook explored the viability of Siamese networks in Face Verification, and generated embeddings by using the second to the last layer in a deep neural network trained with a SoftMax loss (Taigman et al., 2014). The triplet loss function utilizing the Anchor-Positive-Negative framework with the margin parameter was described in a paper by Google (Schroff et al., 2015). Additionally, this paper also discussed the methods for sampling these triplets and suggested training on hard examples rather than randomly sampled triplets from a batch during training.

The domain of flower classification with the Oxford 102 flowers dataset has been explored before by various researchers (Xia et al., 2017; Wu et al., 2018) to varying degrees of success. Our model performs better than a majority of the published methods, and matches the benchmarks of contemporary approaches with 98% accuracy on the test set.

1.3 Dataset Description

We felt the Oxford Flowers 102 dataset was very relevant for our problem. The dataset is a collection of 102 flower categories commonly occurring in the United Kingdom. Each class consists of between 40 and 258 images. The images have large scale, pose and light variations. Additionally, there are categories that have large variations within the category and several very similar categories.

The dataset is divided into a training set, a validation set and a test set. The training set and validation set each consist of 10 images per class (a total of 1020 images each). The test set consists of the remaining 6149 images (minimum 20 per class).

In summary we wanted to train a model to perform a 102 class classification problem, using a dataset that only had 10 samples per class. Sounds ambitious right! But this is a good example of how there might not be an abundance of training data to work with.

1.4 Our Approach

To tackle this classification problem we have to reformulate it in terms of a Few Shot learning task. In case of standard classification, the input image is fed into a series of layers, and finally at the output layer we generate a probability distribution over all the classes (typically using a Softmax layer). However, in few shot classification the network learns a similarity function that takes in the encodings of a pair of images as input and will produce a similarity score denoting the chances that the two input images belong to the same class. In other words, our network is not learning to classify an image directly to any of the output classes. Rather, it is learning a similarity function to numerically express how similar they are.

This has 2 major benefits,

- From our experiments we will show that this network does not require many instances of a class to attain a reasonable accuracy. This is the domain of **Few Shot Learning**.
- We can easily expand the dataset with more classes without touching the architecture of the network. This makes it **more flexible** than a classification network that would require changes to both the last layer and retraining to accommodate for the new class.

A valid question is how to build a classifier from output. A simple solution would be to keep representative images for every training class on hand, allowing us to quickly form pairs with a given training image before feeding it into our network. If our network is trained well, the pair with the highest similarity score would the pair form the same class!

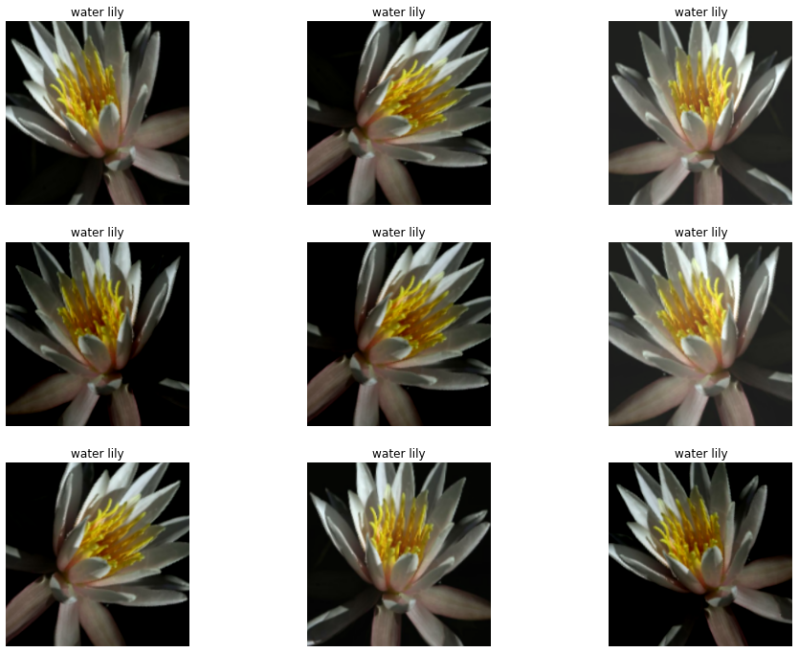
We implemented the idea outlined above and this report will go through the process step by step highlighting the successes and key learnings in detail.

2. Data Preprocessing

The images in the Oxford Flower 102 dataset are of different shapes and sizes. Hence, we will be resizing all the images in each data splits to 224 x 224 pixels. And rescaling the arrays between 1 and 0 by dividing the image array by 255.

Raw data from the Oxford Flower 102 dataset

After basic image preprocessing we will implement data augmentation on our training set. We will use Tensorflow Image Augmentation Layers to help us perform random transformation. Below we have provided the Sequential Image Augmentation Model we prepared. We perform four basic Transformation which includes Flipping, Rotation, Zoom and Brightness.



Example of Data Augmentation on a sample Flower Image

3. Transfer Learning

As we did not have enough data to train a network from scratch, we decided to perform transfer learning from an existing pretrained network. We explored various different architectures highlighted below and the best one was used in the Siamese Network described in detail in the next part.

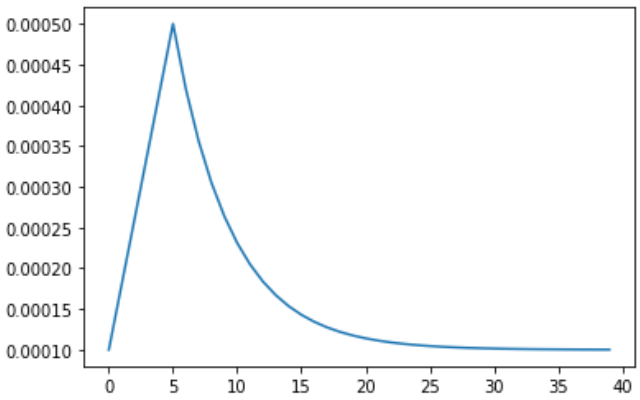
Our objective here is to transfer learn and fine-tune the existing models on the Oxford Flower 102 dataset. We are using the following pretrained models to set the benchmarks:

1. DenseNet201 - Weights on Imagenet
2. EfficientNetB7- Weights on Imagenet and Noisy Student ImageNet
3. InceptionV3 - Weights on Imagenet
4. BigTransfer (BiT) Feature Extractor - Weights on ImageNet

We loaded the above models and removed their last layer (classification layer). We then added our own last layer with neurons equal to the number of classes, and used Softmax activation. Before training, we froze the layers of the pretrained models. We then trained the model using an Adam optimizer, along with a Learning Rate Scheduler which decays the learning rate after a certain number of epochs (ramp-up epochs, which we set as 4 epochs based on trial and error). The learning rate then decays exponentially for following epochs.

The reason for using a large learning rate is that when training transfer learning models with a frozen pretrained model, a low learning rate can cause the training process to take a very long time to converge. By starting with a large learning rate and then decaying it as the number of training steps increases, we achieved better results. Except for the BiT model, we will be using SGD with a decay factor of $1e^{-1}$ after

30%, 60% and 90% of the training steps (BiT-HyperRule proposed by Google for downstream fine-tuning). We also implemented an early stop callback with a min delta of 0.001 with patience 3 on validation loss to avoid overfitting.



Learning rate Curve using LR Scheduler

After the above step we unfreeze some layers except the Batch Normalization layers because it has non-trainable weights to keep track of the mean and variance of its inputs during training and fine-tune the model with a very low learning rate of $1e^{-4}$.

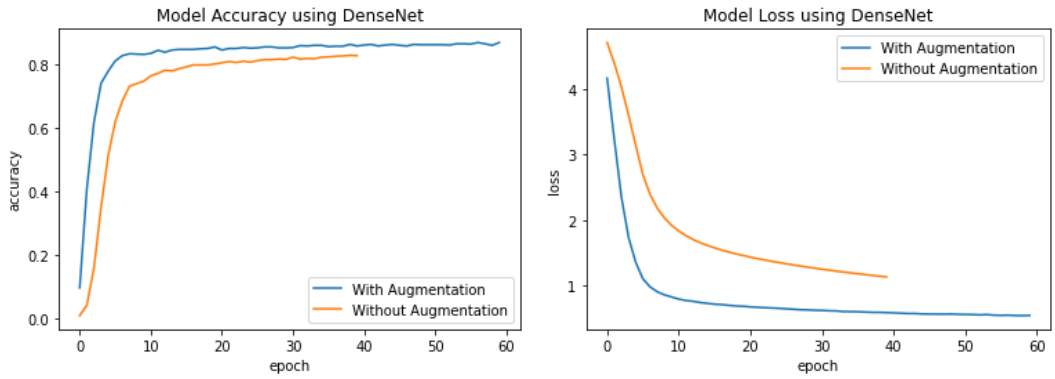
Now after fine-tuning, we will alter the architecture to attempt to outperform the previous fine-tuned models by making the following additions:

- **Dropouts** - We will add dropouts to avoid overfitting and try out different network architectures while tuning.
- **Increasing Network Depth**- We will try to add more learnable parameters during the first stage of training.
- **Batch Normalization** - We added Batch Normalization layers as we are unable to unfreeze these layers during training hence we will add during the first stage of training to make our model learn the mean and variance of our Dataset too.

Model	Test Loss	Test Accuracy
DenseNet201 + Softmax	0.4231	0.9005
Altered InceptionV3	1.0392	0.7544
BiT(Big Transfer) + Softmax	0.1011	0.9819
Altered BiT	**0.0817**	**0.9850**

Benchmark of the Transfer Learning Experiments (Best model in bold)

Now, we also want to experimentally check the importance of Data Augmentation. We believe that Data Augmentation will help our model to generalize better specially because while going through the dataset images we noticed that most of the images provided are very clear and properly oriented in the image frame but in the real world this does not seem to be the case.



Learning Curves. Loss vs Epochs (left). Accuracy vs Epochs (right)

4. Few Shot Learning: Siamese Network

Siamese networks are popular neural network architectures for Few Shot learning tasks that use two identical sub-networks to learn a similarity function. This architecture is primarily based on the idea of reducing the number of parameters in the neural network by sharing the weights between the two similar inputs.

4.1 Architecture

The beauty of the Siamese Network architecture is that we only require the feature extractor from an external model as our loss function calculates the distance between the encodings of a pair of input images. After the transfer learning experiments, we selected the BigTransfer (BiT) Feature Extractor and proceeded to construct the network with the following architecture.

```
# Define the tensors for the two input images
left_input = Input(shape=input_shape, batch_size=batch_size)
right_input = Input(shape=input_shape, batch_size=batch_size)

# Initialize model
model = Sequential()

# Generate the encodings (feature vectors) for the two images
encoded_l = module(left_input)
encoded_r = module(right_input)

# Add a customized L2 layer to compute the euclidean distance between the encodings
L2_layer = Lambda(lambda tensors: K.sqrt(K.square(tensors[0] - tensors[1])))
L2_distance = L2_layer([encoded_l, encoded_r])

# Add a dense layer with a sigmoid unit to generate the similarity score
prediction = Dense(1, activation='sigmoid')(L2_distance)

# Connect the inputs with the outputs
siamese_net = Model(inputs=[left_input,right_input], outputs=prediction)
```

A code snippet on how we built the model using the Keras APIs

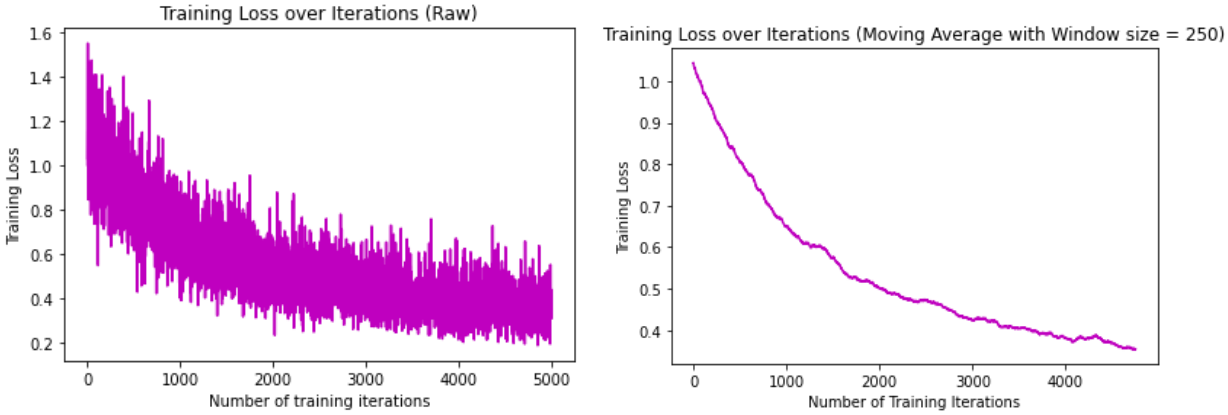
We implemented a custom layer for the similarity function which calculates the L2 distance between the pair of encodings and finally added a single neuron sigmoid layer to output a probability that the pair of images belonged to the same class (similarity score, 1 being identical encodings).

4.2 Preparing the Data

We first had to create a custom data loader as the Siamese Model takes a pair of input images. We paired the images within the batch, else there would be around 500,0000 pairs. We shall explain how the data loader worked with the batch size 50 (which was the batch size we used for training). The data loader randomly sampled an image from 50 classes, for half the pairs it sampled another image from the same class and for the remaining pairs it sampled an image from a different class. We ensured the batches had a 1:1 ratio of positive and negative instances as this would help in training the model's similarity function. *Refer to Appendix for an illustration.*

4.3 Training the Model

After creating the pipeline to retrieve batches, we trained the model for 5000 iterations using the SGD optimizer with a learning rate of 0.0001. In each iteration a batch of 50 pairs were sampled and fed into the model. We trained the model with the binary cross entropy loss which incentivized it to make images from the same class have similar embeddings and images from different classes have different embeddings. A plot of the learning curve is shown below,



Training Loss vs Number of Iterations. Raw data (Left). Data after applying Moving Average (Right).

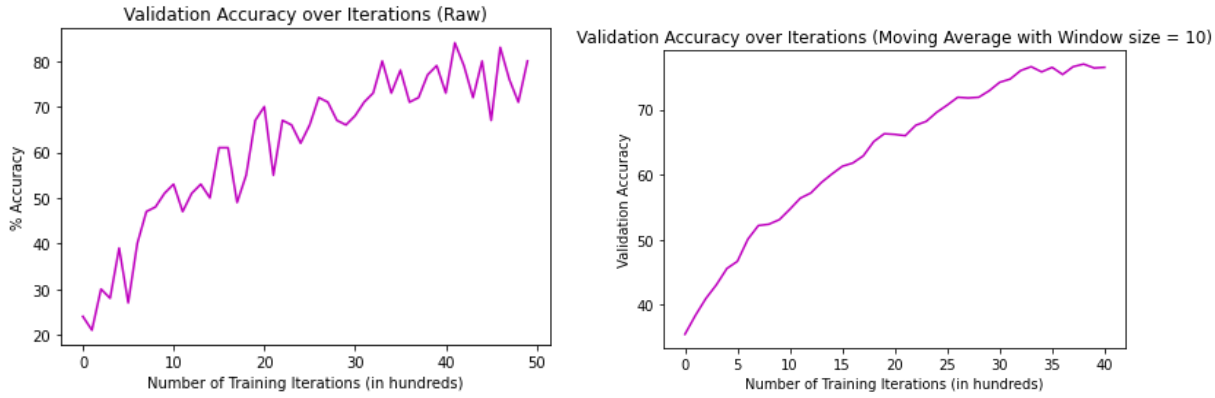
4.4 Validating the Model

For every pair of input images, our model generates a similarity score between 0 and 1. However with this score alone it is difficult to ascertain whether the model is really able to recognize similar images (from the same class) and distinguish dissimilar ones. With this in mind, we used the **N-way few shot** learning task strategy to validate and test our model.

In this method, we prepared 100 tasks (arbitrary) for the network to validate on. In each task, N categories were selected randomly and pairs of images were constructed such that only 1 pair had images from the same class while the rest were from different classes. [Also, we ensured that all pairs had the same left image by duplicating the image sampled from the first class N times]. *Refer to Appendix for an illustration.*

Now it is easy to tell whether the model is working by checking if the highest similarity score belongs to the pair of images from the same class. Then we can define the validation accuracy as the number of trials that the model correctly assigned the highest similarity score to the matching pair!

We used 100 trials of 5-way few shot learning tasks and validated our model every 100 iterations during the training process. We used the validation accuracy as a metric to save the best model. A plot of the validation accuracy against every 100 iterations is shown below.



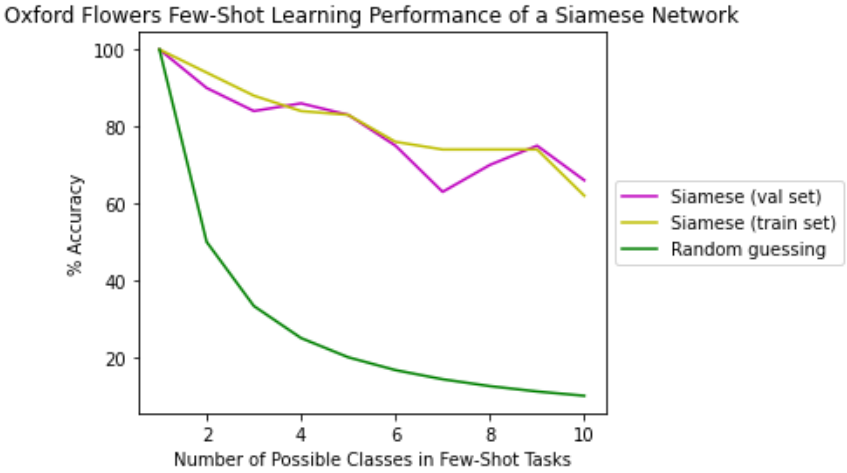
Validation Accuracy vs Number of Iterations. Raw data (Left). Data after applying Moving Average (Right).

```
Best Validation Accuracy for 5 Way One-Shot Learning:  84.00%
Best model file: 'weights.4200.h5'
```

The best validation accuracy attained was with the model after 4200 iterations.

4.5 Varying the Number of Ways

Now that we have the best model. We can test how well it generalizes to other N-way classification. We took the best model and ran the same validation script, increasing the number of possible classes (N) from 1 to 10. Our baseline was random guessing i.e., y = 1/N curve (as each class would have an equal chance of being selected). The plot is shown below and we can see that our model performs fairly well.



Trend of Validation Accuracy as we increase Number of Ways. Baseline is Random Guessing (Green)

Although the validation accuracy decreases as we increase the number of ways, our model greatly outperforms the baseline of randomly guessing the class. We could build our classifier now, using the

encodings generated by the network. However we felt we needed to further improve them as classification would be a 102-way task. Thus we decided to try out the more sophisticated Triplet Loss as the loss function to generate better embeddings for the images.

5. Few Shot Learning: Triplet Loss

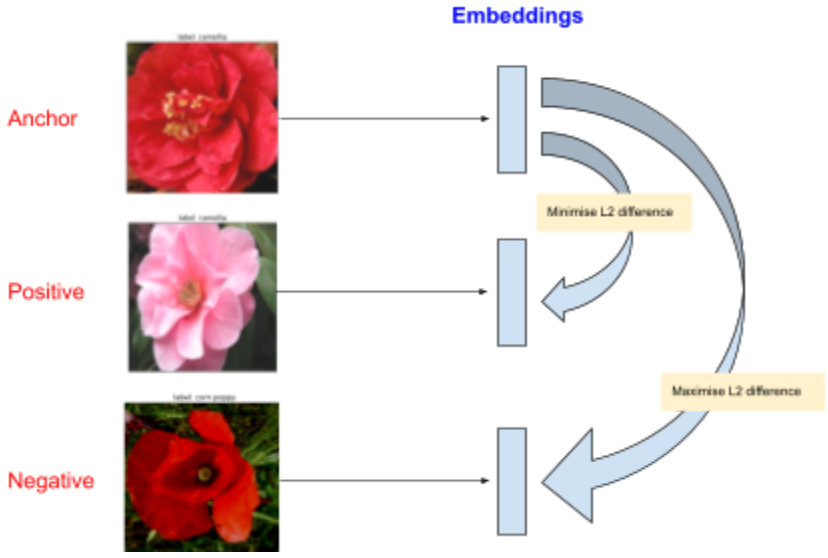
Our goal in Few Shot Learning is to be able to classify images of different entities into different categories by training on only a few training examples. Here, we attempt to classify the images of the 102 different categories of flowers found in Britain by using just 10 images per class in the training dataset. We'll be improving upon the embeddings that the Siamese Network could generate.

5.1 Architecture

One way to do this is to create embeddings of the images of the flowers in some n-dimensional space such that the same type of flowers are very close to one another in the embedding space whereas different types of flowers are further apart, i.e., we want to have

$$L2(rose_1, rose_2) < L2(rose_1, marigold_1)$$

in our embedding space. In Few-shot Learning terminology, the image to be compared to is called the **Anchor,** and along with the anchor image, we select one **Positive** example belonging to the same class and one **Negative** example belonging to a different class. Our goal, as stated above, is to **reduce the L2 difference between the (Anchor, Positive) pair** and **increase the difference between the (Anchor, Negative) pair**.



An illustration of how the triplet loss function works.

However, simply trying to make `L2`(Anchor, Positive) < `L2`(Anchor, Negative) does not result in the best model as the difference between them can be very small. Hence, we make the condition stricter by using:

$$L2(Anchor, Positive) < L2(Anchor, Negative) - margin$$

Where margin is any positive number. Rearranging the terms and taking a max to ensure that the Loss function never becomes negative we get:

$$TripletLoss(A, P, N) = max(0, L2(A, P) - L2(A, N) + margin)$$

Now, we can sample any two images from the same class and one image from a different class, apply our model to get an encoding of the image and improve the encoding by backpropagating using the triplet loss.

In practice, however, the model can be improved by training on 'hard examples'. These examples have L2(Anchor, Positive) > L2(Anchor, Negative) and can provide the best direction for improvement.

5.2 Building and Evaluating the Classifier

In our notebook, we have utilized semi-hard triplet loss which trains the model on semi-hard examples from a batch. Semi hard triplets are those triplets sampled from a batch such that the distance between the anchor and positive image is less than the distance between the anchor and negative image, and the Triplet loss is positive.

For generating the embeddings, we utilized transfer learning on the Oxford 102 flowers dataset with the Bit 101x base model. Data Augmentation and up sampling were used to increase the robustness of our model.In particular, we used random brightness, contrast, saturation and hue along with random flipping to augment the training set images with a 10:1 augmented to real images ratio. Random brightness, in particular, is recommended by the authors as the flowers of the dataset were captured in varying brightness settings.

A sample of the test embeddings are visualized in the next section.

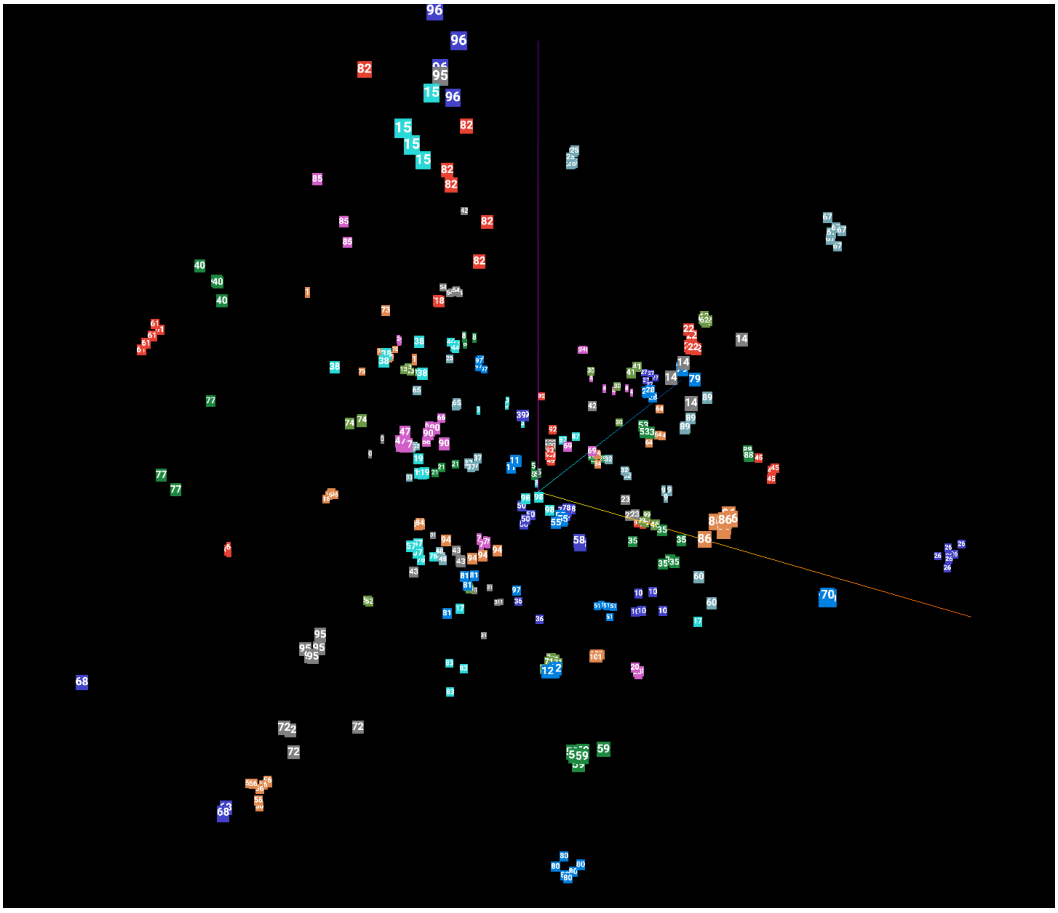
We can easily utilize these embeddings for downstream tasks such as classification. In particular, we can save an embedding of an image from each flower class as a representative image for that type of flower and compare these embeddings with the embeddings of test images to assign the test image to a class.

One interesting aspect of this architecture is that it is very robust to the **addition of new flower classes** to the model. We need not retrain the model as in the case of a softmax based classifier every time we wish to add a new flower class to the model. Additionally, our classifier can also **identify flowers that belong to some 'others class'** that were not in the train examples.

The final model was able to classify flowers in the test set with a **98.8%** accuracy.

5.3 Visualizing Results

We can visualize the embeddings in lower dimensions using PCA or Umaps. We can clearly observe that the same categories of flowers are closer to one another compared to flowers from other categories, and appear as clusters in the Principal Components space. Moreover, we also show the top-k L2 nearest neighbors for a sampled point from class 86 and observe that indeed, images from class 86 are the closest to the sampled image in the projected embedding space.



A visualization of the embedding generated by using the Triplet Loss Network.

6. Conclusion and Discussion

Although few-shot learning is a promising direction for future research, more work is needed to improve its effectiveness. In particular, more research is needed on how to effectively select and use training data, how to design better models, and how to better exploit prior knowledge.

The main aim of our project was to develop a model to perform image classification with a very small amount of training data. We achieved this by leveraging pre-trained models, transfer learning and creating clever architectures. We started by benchmarking the performance of transfer learning by pretraining on a large dataset and then fine tuning on our smaller flower dataset. We found the BiT models performed the best.

We then explored the use of Siamese Networks for Few Shot Learning. This architecture allowed us to generate encodings for images. We built a model and trained it to produce similar encodings for images from the same class, and different encodings for images from different classes. We validated the model using the N-Way Few Shot Learning task, and found it performed reasonably well.

We then explored the use of the Triplet Loss function to further improve the encodings generated. We used semi-hard triplet sampling to train our model, and found the encodings to be greatly improved. We built a classifier using the embeddings (in essence performing a 102-way few shot learning task) and found that the model performed exceptionally well, reaching a classification accuracy of 98.8%.

We were quite pleased with the performance of the model. However, there are several areas of improvement that we would like to work on in the future.

1. More Training Data for creating Embeddings

One of the most significant improvements we could make to the model would be to increase the number of training samples for generating the embeddings. Here, we used the 1020 images in the train dataset to train the model using triplet loss. Ideally, a much larger dataset should be used to train this deep network.

Once the model is trained on a variety of flower images, we could use it to learn new flower encodings from another dataset and predict on the test set. This would require relatively few training images of these new flowers.

2. Model Architecture

We could also try alternative model architecture for few-shot learning. For instance, the models we implemented were from papers published in 2016. Since then a variety of novel architectures have been developed to tackle this few-shot learning problem. A particularly interesting paper utilized GANs to generate images for the triplet loss, leading to higher quality embeddings of the anchor.

7. References

1. *M. -E. Nilsback and A. Zisserman, "Automated Flower Classification over a Large Number of Classes," 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing, 2008, pp. 722-729*
2. *Koch, G.R. (2015). Siamese Neural Networks for One-Shot Image Recognition.*
3. *Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., & Houlsby, N. (2020, August). Big transfer (bit): General visual representation learning. In the European conference on computer vision (pp. 491-507). Springer, Cham*
4. *Vinyals, O., Blundell, C., Lillicrap, T., & Wierstra, D. (2016). Matching networks for one shot learning. Advances in neural information processing systems, 29.*
5. *Wang, Y., Yao, Q., Kwok, J. T., & Ni, L. M. (2020). Generalizing from a few examples: A survey on few-shot learning. ACM computing surveys (csur), 53(3), 1-34.*
6. *Wang, L., & Yoon, K. J. (2021). Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. IEEE Transactions on Pattern Analysis and Machine Intelligence.*
7. *Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018, October). A survey on deep transfer learning. In International conference on artificial neural networks (pp. 270-279). Springer, Cham.*
8. *Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., & Shah, R. (1993). Signature verification using a" siamese" time delay neural network. Advances in neural information processing systems, 6.*
9. *Chopra, S., Hadsell, R., & LeCun, Y. (2005, June). Learning a similarity metric discriminatively, with application to face verification. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) (Vol. 1, pp. 539-546). IEEE.*
10. *Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1701-1708).*

11. *Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 815-823).*

12. *Xiaoling Xia, Cui Xu, & Bing Nan. (2017). Inception-v3 for flower classification. 2017 2nd International Conference on Image, Vision and Computing (ICIVC), 1106–1787. https://doi.org/10.1109/ICIVC.2017.7984661*

13. *Wu, Qin, X., Pan, Y., & Yuan, C. (2018). Convolution Neural Network based Transfer Learning for Classification of Flowers. ICSIP : 2018 IEEE 3rd International Conference on Signal and Image Processing : July 13-15, 2018, Shenzhen, China /, 115–566. https://doi.org/10.1109/SIPROCESS.2018.8600536*

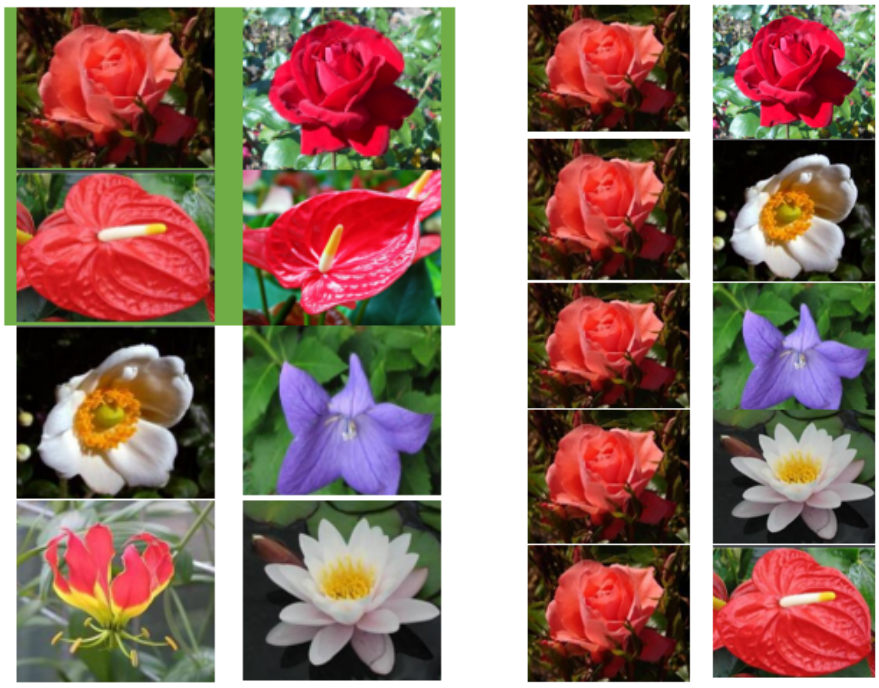
14. *Zhengfang, He & Su, Weibin & Bi, Zhimin & Wei, Ming & Dong, Yingguo & Xu, Gang. (2019). The Improved Siamese Network in Face Recognition. 443-446. 10.1109/ICICAS48597.2019.00099.*

15. *https://blog.tensorflow.org/2020/05/bigtransfer-bit-state-of-art-transfer-learning-computer-vision.html*

16. *https://www.borealisai.com/research-blogs/tutorial-2-few-shot-learning-and-meta-learning-i/*

17. *https://neptune.ai/blog/understanding-few-shot-learning-in-computer-vision*

8. Appendix



***(Left)** An example of a batch of input images used for training the Siamese Network. This is an example for batch size 4, note how there are 2 similar (green) and 2 dissimilar pairs.*

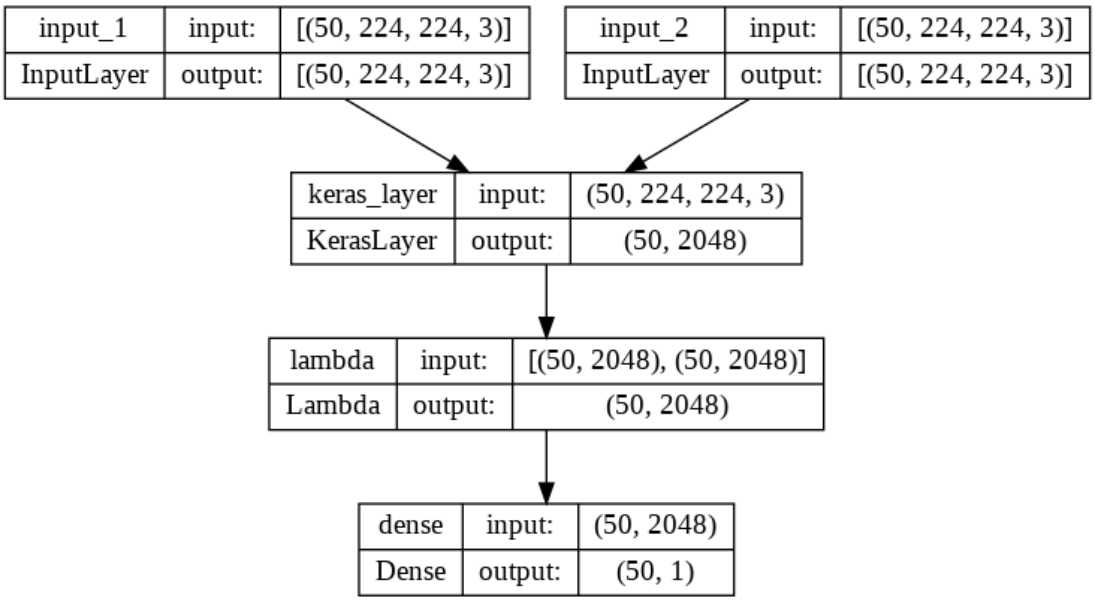
***(Right)** An example of a 5-way validation task used for validating the Siamese Model.*

```
IMG_SIZE = 224

img_processing = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Resizing(IMG_SIZE, IMG_SIZE)
])

img_augmentation = tf.keras.Sequential([
          tf.keras.layers.RandomFlip("horizontal"),
          tf.keras.layers.RandomRotation(factor=0.2),
          tf.keras.layers.RandomZoom(height_factor=(-0.2,-0.3), width_factor=(-0.2,-0.3)),
          tf.keras.layers.RandomBrightness((-0.2,0.2), value_range=[0,1])],
        name='img_augmentation'
        )
```

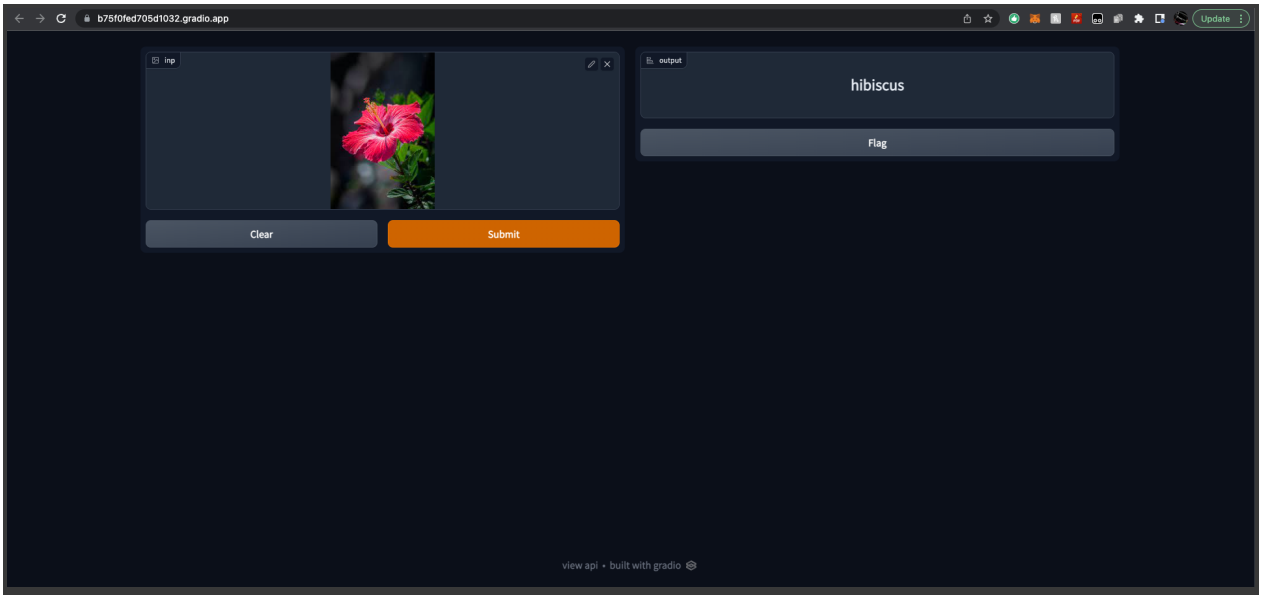
The layers of the network used for image augmentation.

```
def makeOneShotTask(self, N):
  """
    Create pairs of test image, support set for testing N way one-shot learning.
  """
  # Randomly sample several classes (flowers) to use in the pairing of test data
  flowers = random.sample(self.labels, k=N)
  true_category = flowers[0]

  # Sample the test image and duplicate it to make the left pair
  left = random.sample(self.data[true_category], k=1)[0]
  test_image = np.asarray([left]*N)

  # Initialize empty array for the right pair
  support_set = np.array([np.zeros((self.w, self.h, self.c)) for _ in range(N)])
  support_set[0] = random.sample(self.data[true_category], k=1)[0]

  for i in range(1, N):
    category = flowers[i]
    support_set[i] = random.sample(self.data[category], k=1)[0]

  # Initialize empty array for the targets
  targets = np.zeros((N,))
  targets[0] = 1

  # Shuffle the dataset and format into pairs
  targets, test_image, support_set = shuffle(targets, test_image, support_set)
  pairs = [test_image, support_set]

  return pairs, targets
```

Code for generating N-way few-shot learning tasks for validation



A diagram of the Siamese Network Architecture.



A screenshot of the User Interface we made for easily testing our classifier.

Classification Results from the Classifier Built using the Triplet Loss Embeddings

index	precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	1.00	0.93	0.96	40
2	0.95	1.00	0.98	20
3	0.94	0.92	0.93	36
4	1.00	0.98	0.99	45

5	1.00	1.00	1.00	25
6	0.91	1.00	0.95	20
7	1.00	1.00	1.00	65
8	0.96	1.00	0.98	26
9	1.00	1.00	1.00	25
10	0.99	1.00	0.99	67
11	1.00	1.00	1.00	67
12	1.00	1.00	1.00	29
13	1.00	1.00	1.00	28
14	1.00	1.00	1.00	29
15	1.00	0.95	0.98	21
16	1.00	1.00	1.00	65
17	0.95	0.97	0.96	62
18	0.97	0.97	0.97	29
19	1.00	1.00	1.00	36
20	1.00	1.00	1.00	20
21	1.00	1.00	1.00	39
22	1.00	1.00	1.00	71
23	0.96	1.00	0.98	22
24	1.00	1.00	1.00	21
25	0.95	1.00	0.98	21
26	1.00	1.00	1.00	20
27	1.00	1.00	1.00	46
28	1.00	1.00	1.00	58
29	1.00	0.95	0.98	65
30	0.89	1.00	0.94	32
31	0.96	1.00	0.98	25
32	1.00	1.00	1.00	26
33	1.00	0.95	0.97	20
34	1.00	1.00	1.00	23
35	0.98	1.00	0.99	55
36	1.00	0.99	0.99	88
37	1.00	1.00	1.00	36
38	0.95	1.00	0.98	21
39	1.00	0.94	0.97	47
40	1.00	0.99	1.00	107
41	0.95	1.00	0.97	39
42	0.98	0.85	0.91	110
43	1.00	1.00	1.00	73
44	1.00	1.00	1.00	20
45	1.00	0.99	1.00	176
46	1.00	1.00	1.00	47
47	1.00	1.00	1.00	51
48	1.00	1.00	1.00	29
49	1.00	1.00	1.00	72
50	1.00	1.00	1.00	238
51	1.00	1.00	1.00	65
52	1.00	1.00	1.00	73
53	1.00	1.00	1.00	41
54	1.00	1.00	1.00	51
55	0.99	1.00	0.99	89
56	1.00	1.00	1.00	47
57	0.99	1.00	0.99	94
58	1.00	1.00	1.00	47

59	1.00	1.00	1.00	89
60	1.00	1.00	1.00	30
61	1.00	1.00	1.00	35
62	1.00	1.00	1.00	34
63	0.97	1.00	0.98	32
64	1.00	1.00	1.00	82
65	0.98	1.00	0.99	41
66	1.00	1.00	1.00	22
67	0.89	1.00	0.94	34
68	1.00	1.00	1.00	34
69	0.98	1.00	0.99	42
70	1.00	1.00	1.00	58
71	0.93	0.99	0.96	76
72	0.98	1.00	0.99	174
73	0.99	0.99	0.99	151
74	0.99	1.00	1.00	100
75	1.00	0.98	0.99	87
76	1.00	1.00	1.00	231
77	1.00	0.97	0.99	117
78	1.00	1.00	1.00	21
79	1.00	1.00	1.00	85
80	1.00	1.00	1.00	146
81	1.00	1.00	1.00	92
82	1.00	0.97	0.99	111
83	1.00	0.98	0.99	66
84	0.98	0.98	0.98	43
85	0.97	1.00	0.99	38
86	1.00	1.00	1.00	43
87	0.99	1.00	1.00	134
88	1.00	1.00	1.00	164
89	0.97	0.98	0.98	62
90	1.00	1.00	1.00	56
91	1.00	1.00	1.00	46
92	1.00	0.92	0.96	26
93	0.99	0.99	0.99	142
94	0.99	0.98	0.99	108
95	0.95	0.83	0.89	71
96	0.78	1.00	0.88	46
97	1.00	0.98	0.99	62
98	0.98	1.00	0.99	43
99	1.00	1.00	1.00	29
100	0.95	0.97	0.96	38
101	1.00	1.00	1.00	28

accuracy			0.99	6149
macro avg	0.99	0.99	0.99	6149
weighted avg	0.99	0.99	0.99	6149