**Assignment 3 Report**

CPS 584 - Advanced Intelligent Systems and Deep Learning

University of Dayton

April 25, 2023

Authors:

Connar Hite

Ranjita Piratla

Sean Saud

Monika Somu

T.Shiva Harshith Varma

**1 Introduction**

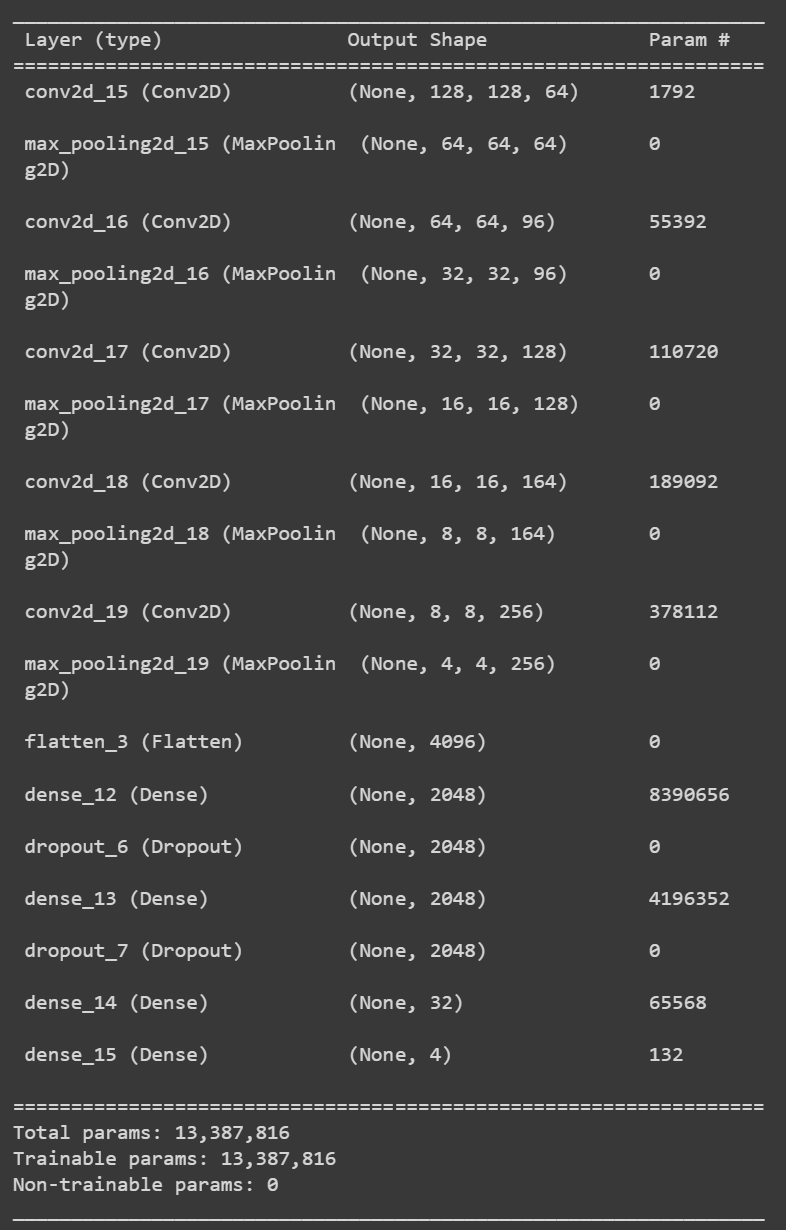
The purpose of this assignment is to get hands-on experience with deep learning networks. This is accomplished by designing our own custom network to classify various flowers: Roses, Tulips, Dandelions, and Sunflowers. A set of training and testing images, containing 40 and 20 images per class respectively, is provided. 30 additional images, per class, are obtained to be used during training. This will result in a total of 70 images per class. A deep learning network is then designed to solve the given problem. This network needs to be different from LeNet, AlexNet, and CustomNet. The created network will be tested and adjusted with the intention of creating the most efficient model possible. The total accuracy, and accuracy per class, will be calculated per test. In order to improve the accuracy of the system, the training images will be transformed and provided for training. The first of these transformations will involve flipping all of the images horizontally. To further improve the accuracy, other image transformations will be tested and reported on. This can include, but is not limited to, rotating, increasing the brightness, and shifting the images. Finally, the results of the tests will be analyzed and certain discussion questions answered.

**2 Problems**

**2.1 Deep Learning Model**

Our deep learning model, SeanNet comprises 14 layers, not including the input, which is a 128x128 pixel image. The first 10 layers are convolutional (convolution followed by max pooling), and the last 4 layers are fully-connected. Each convolutional layer uses the Leaky Rectified Linear Unit (Leaky ReLU) activation function and has a stride of 1 pixel. After each convolutional layer, a subsampling layer is applied using max pooling with a pool size of 2x2, halving the size of each feature map. Each fully-connected layer uses the Scaled Exponential Linear Unit (SELU) activation function and Lecun normal kernel initialization strategy. Directly after each of the first 2 fully-connected layers, 50% of their outputs are randomly dropped for weight regularization to prevent hidden units from relying too heavily on other hidden units. The last fully-connected layer output is passed to a 4-way softmax that results in a distribution over the 4 possible flower classes. The highest value in the distribution is chosen as the predicted class.

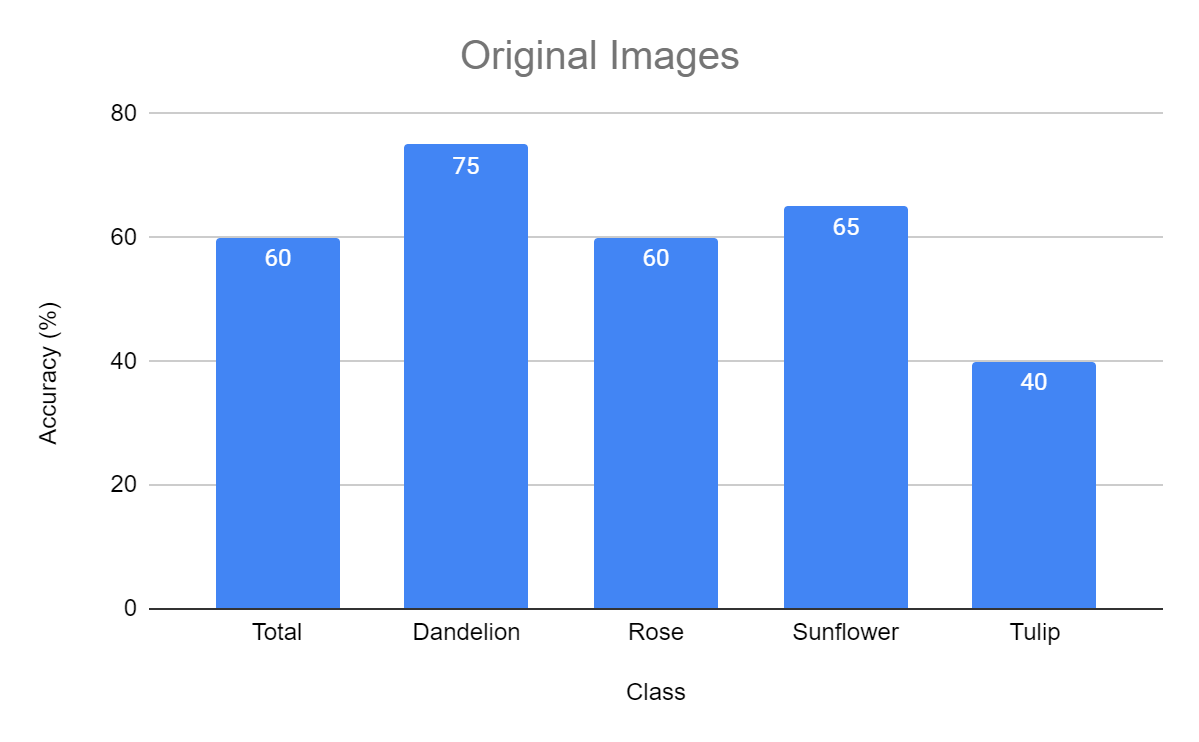
The 128x128x3 input image is filtered by the first convolutional layer (C1) with 64 kernels of size 3x3x3 and produces an output with a shape of 128x128x64. The C3 layer uses the output of the first layer (after max pooling) as input and filters it with 96 kernels of size 3x3x64. Layer C3 produces an output of size 64x64x96. The output of C3 after max pooling is passed as input to layer C5 which filters it with 128 kernels of size 3x3x32. C5 produces an output of size 32x32x128, which is then passed to the next max pooling layer, S6. The output is filtered by C7 using 164 kernels of size 3x3x16, producing an output of size 16x16x164. After max pooling, the output of C7 is passed to C9 as input, which is filtered with 256 kernels of size 3x3x8. Layer C9 produces an output of size 8x8x256, which is passed to the last max pooling layer, S10, which produces an output of 4x4x256. The output of S10 is flattened to a size of 4096, and then passed to the first fully-connected layer, F11, with 2048 nodes. The output of F11 is passed to a dropout layer which randomly removes half of the hidden units. Then, the remaining half of hidden units is passed to the next fully-connected layer, F12, which has 2048 nodes. Again a dropout layer randomly removes 50% of the hidden units. The output of the previous dropout layer is compressed in F13, a fully-connected layer with 32 nodes. Finally, the output of F13 is passed to the fully-connected softmax output layer of 4 nodes (for 4 classes) and 132 parameters.



**Figure 1: SeanNet Structure**

**2.2 Original Images**

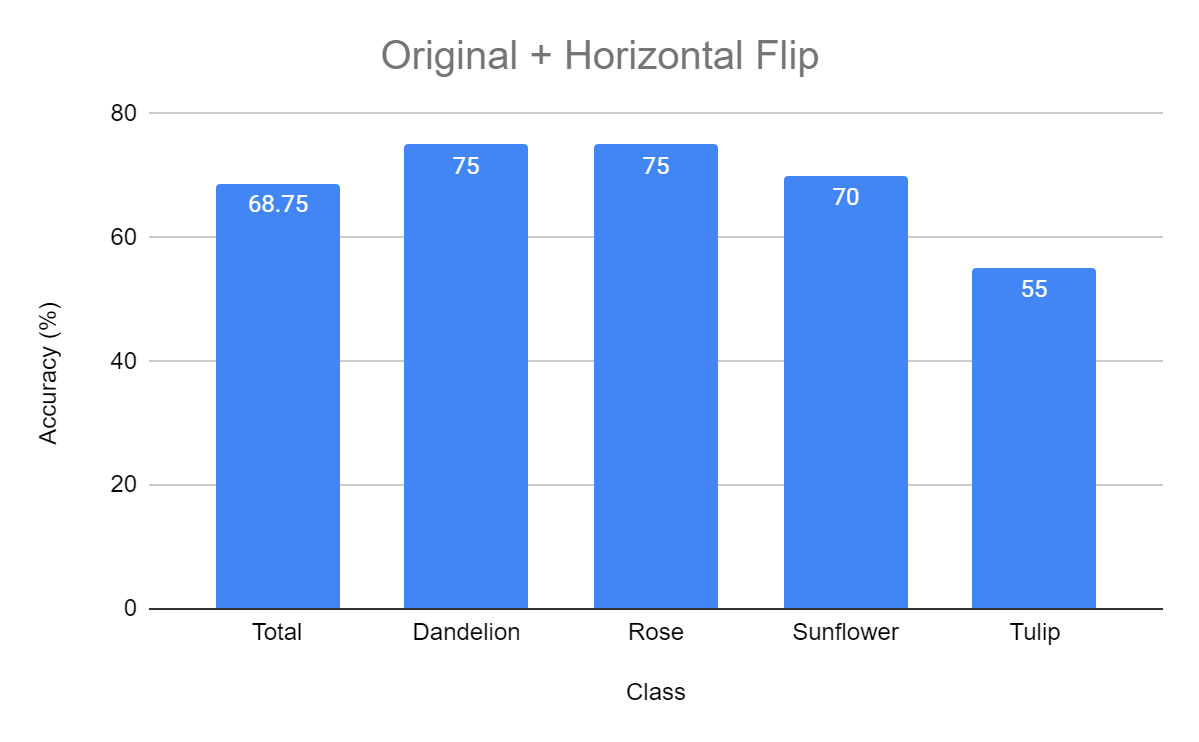
A set of training and testing images is provided. This includes four classes of flowers: Rose, Tulip, Dandelion, and Sunflower. The provided dataset contains 40 images per class for training and 20 images per class for testing. An additional 30 images per class are acquired for testing, resulting in 70 images per class. To serve as a baseline, the first test performed only involves training using these unmodified images. The number of epochs performed is set to be 50. However, an early stopping condition is set to where the training will end if the accuracy reaches 100%. This is done to prevent overfitting. Observing the loss was also observed as an early stopping condition, but this did not improve results. Figure 1 displays the total and per class testing results.



**Figure 2: Original Image Results**

**2.3 Flipped Images**

To improve the accuracy of the SeanNet model, each of the original training images were horizontally flipped to produce a set of flipped images. This resulted in an additional 70 training images per class, bringing the total number of training images per class to 140 images. The data generator class does have the ability to flip images as they are imported. However, this process is random and does not increase the number of training images. Thus, an array including the original images and new horizontally flipped images needed to be created. This new dataset was then used to create the input for the network. This new set of images is then used to train SeanNet, the results of which are shown in Figure 3.

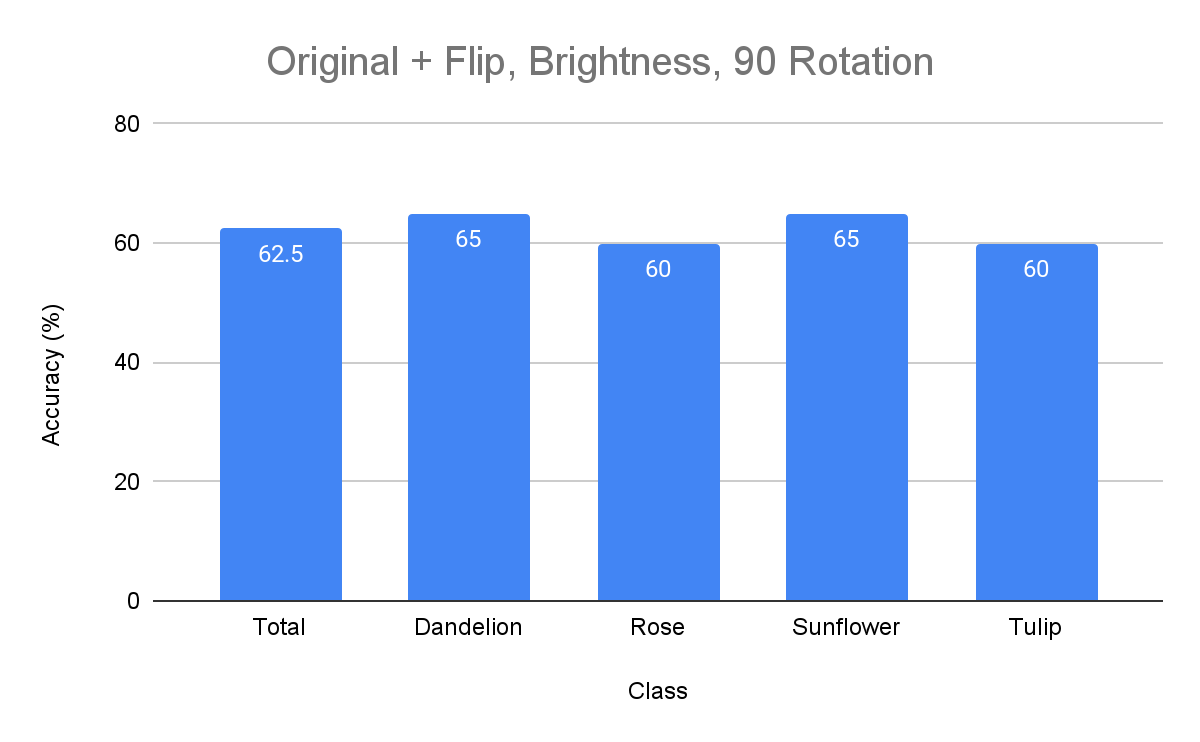


**Figure 3: Original + Horizontal Flipped Results**

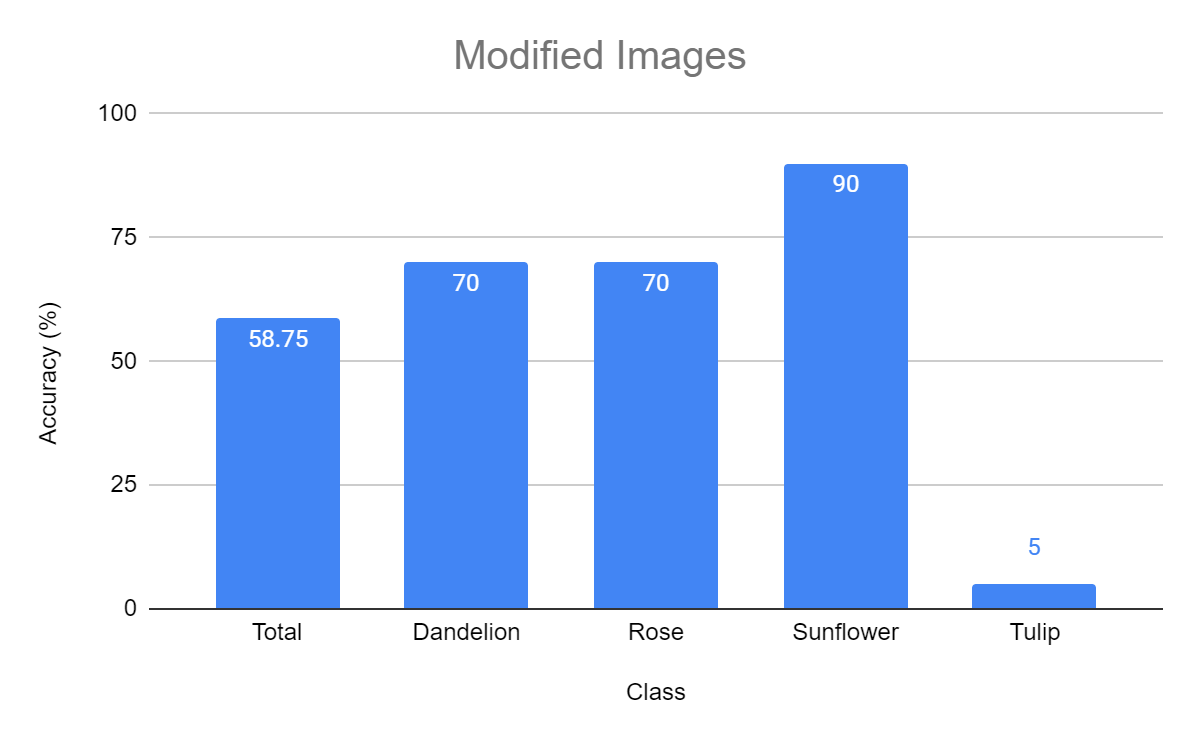
**2.4 Further Performance Improvements**

To further improve the results of the system, various image preprocessing techniques were implemented. This was accomplished by either appending the transformed images to the preexisting training set or by changing the parameters of the “ImageDataGenerator” function. The former would increase the training set, which means more samples to train on and a higher computational cost. The latter does not increase the size of the training set. Rather, it randomly applies the modifications to each input image. This does not increase the training time, but does not result in more samples to train with.

Figure 4 shows the results of additional modified images being appended to the preexisting training set. The total set included the original, brightened, horizontally flipped, and rotated 90° images. The total training set included 280 images per class after these modifications were made. For tests involving editing the parameters of “ImageDataGenerator,” changing the rotation range, shifting along the width/height, adjusting the brightness, and zoom were assessed. Figure 5 shows the results of one such test that includes modifiers for rotation range, width/height shift, horizontal flip, and zoom.



**Figure 4: Original + Flip, Brightness, 90 Rotation Results**



**Figure 5: Modified Image Results**

**3 Issues Encountered & Solutions**

The lack of data proved to be a challenging issue for our classification system. The training accuracies being much higher than the testing accuracies indicates that our system was experiencing overfitting and memorizing the training data, making it not be able to be as successful identifying new examples. To solve this, we augmented the training data by horizontally flipping the training images for additional data. Using these additional training images resulted in improvements in testing accuracy, so we added additional training data through adding rotated images and brightened images. We also stopped the training process early by monitoring the loss and accuracy. If the loss started increasing or the accuracy reached a threshold, then the network would stop training. Additionally, to help reduce overfitting, two dropout layers drop 50% of the outputs of the first two fully-connected layers.

**4 Discussion**

While only using the original, unmodified images, the results of SeanNet were promising. The accuracy did change each time the system was trained; however, the results were usually around 60-65%. Increasing the number of training images by appending horizontally flipped images noticeably improved the accuracy. This resulted in accuracies of about 68-72%. However, continuing to increase the number of images by adding modified versions of the originals did not always prove as effective. For example, Figure 4, while still having an accuracy higher than the baseline, had a lower accuracy than the original and horizontal image test. This was despite the fact that it had twice as many images. Other tests were performed using this method and the results were the same. Simply adding more and more images did not improve the system past what was seen with just the original and horizontal images.

As for the per class accuracy, there are a couple things of note. Firstly, tulips were constantly the worst class. For most of the tests, this was a significant difference. Figure 2 and Figure 3 show tulips performing about 20% worse than the other classes. While Figure 4 shows it performing about the same. On the other hand, Figure 5 shows tulips having an accuracy of 5%. One explanation for this could be that the tulips class contains a wide variety of images. As a flower, tulips can look very different from one another in terms of shape and color. As for the other classes, they usually performed about the same. One exception being in Figure 5, where sunflowers obtained an accuracy of 90%. This could be due to the fact that sunflowers generally look the same, but this much of a difference is not shown in other tests.

One of the difficulties involved in creating our own network came in the form of all the possibilities. Everything could be tweaked with multiple parameters. This resulted in much time being spent changing small things seeing if the accuracy improved at all. It became difficult to determine a stopping point, because there was always more that could be changed. Handling more than 2 classes, while different from what was done during class, was not too difficult. Layers and parameters needed to be tuned for multiclass classification, but this was not a huge issue.

**5 Conclusion**

In this project, we created a custom deep learning network called SeanNet to classify four different types of flowers. Because we had limited training data, the model overfitted. To address this issue, we supplemented the training data by flipping, rotating, and brightening the original images. We also used early stopping and dropout layers to avoid overfitting.

Overall, our efforts improved the model's accuracy, as evidenced by testing. However, there is still room for improvement, such as experimenting with different preprocessing techniques or adjusting the parameters of the existing techniques.

This project provided valuable hands-on experience with deep learning networks and the challenges that can arise while they are being developed. This project's skills and knowledge can be applied to future projects in deep learning and image classification.

**6 Member Contributions**

Connar and Sean wrote the code individually, coming together to compare results and make any improvements. Every member created and tested their own networks individually. We then came together to decide whose model to use and improve upon. Sean’s CNN provided the best results constantly, and thus was used. Each member then spent time testing different image modifiers to try and improve the accuracy of the system.

Connar Hite: Assisted in creating & testing Deep Learning Model, wrote introduction & further improvements section, and contributed to writing & editing other sections

Ranjita Piratla: Wrote conclusion section

Sean Saud: Wrote Deep Learning Model, Contributed to writing Issues Encountered & Solutions, Helped create network diagram

Monika Somu: Wrote flipped images section

T.Shiva Harshith Varma: Wrote original images section