COSC420: Classifying and Generating Images of Flowers

Jake Norton (5695756)
April 15, 2024

1 Introduction

This report explores the use of deep learning models to classify and generate images of flowers using the Oxford flowers dataset, focusing on convolutional neural networks (CNNs) and diffusion models. The study involves designing and training these models using TensorFlow's Keras library, emphasizing the creation of models from scratch and the exploration of both coarse and fine-grained datasets.

2 Classification using CNN

2.1 Model Architecture

As VGG seems like a de-facto standard for convolutional neural network classifiers I decided to use a smaller version of the VGG net as my model architecture. This has 4 convolutional layers for the encoder portion, and 2 fully connected layers plus a fully connected output layer. By default this model runs into issues, mainly due to the imbalance in the dataset. The imbalance and lack of variety pushes the model towards overfitting before it can reach good scores. To combat this I added some data augmentation, to increase the variance and overall size of the dataset, hopefully to allow the model to be able to generalise better. Major transforms like flipping the image, and minor like mutating the images so that they are slightly different so that the model does not concentrate on exact patterns too much and should generalise around the shape / texture. On the same token I found that increasing the batch size from the default 32 to 100 helped for the fine-grained dataset as it allowed each batch to have some representation of each class in the dataset. I would have done more experimentation on changing the hyperparamters, but I did not have time, especially once I found that the model performed best when using the 256 pixel images, which took a long time to train, especially at 200 epochs.

2.2 Hyperparameters

- image-size: 256
- epochs: 200
- optimizer: adam
- reg-wdecay-beta: 0.1
- reg-dropout-rate: 0.4
- reg-batch-norm: true
- learning-rate: 0.0001
- data-aug: true

2.3 Data Augmentation

- zca-epsilon:1e-06
- width-shift-range:0.2
- height-shift-range:0.2
- shear-range:0.2
- zoom-range:0.2
- fill-mode: nearest
- horizontal-flip:true

2.4 Evaluation

Evaluation was based mainly on F1 score as well as looking at the confusion matrices of the classes.

-

3 Image Generation

3.1 Task 2a: Auto-encoder

3.1.1 Architecture

Initially I tried to use my VGG net style of model as an auto-encoder. However this did not give me good results as well as having high number of parameters involved in the dense layers. I believe the deep network just struggled with the upsampling process. From here I decided why not build my own U-net style model, with and without skip connections.

Architecture here —

Skip-connections allow for gradient to flow from an earlier point in the model, to skip some of the intermediate layers. For deep models, or in this case where the model has a smaller representation, this allows for the model to retain some of the earlier details that have been lost by the downscaling. The model being used purely as an auto-encoder this makes a lot of sense, however I believe this will make the model less flexible if split into a encoder and a decoder. The decoder becomes more reliant on the encoder. I wanted to test

3.1.2 Training

Compressing down the image to a small latent space then upsampling without skip-connections did a reasonably good job and I was about to get around an 80% accuracy add pixel error here. With the upsampling this loss of information during the compression phase did lead to a loss of quality and a slightly blurry representation.

With the addition of skip-connections, the image comes out a lot more crisp, the fine-grained information passed from previous layers seems to help the upsampling process

3.2 Hyperparameters

- image-size = 256
- epochs = 200
- noise-range = (0.0, 2.0) standard deviation
- load-from-file = false
- verbose = true
- encoder = false
- reg-wdecay-beta = 0.1
- reg-dropout-rate = 0.4
- reg-batch-norm = true
- data-aug = true
- flip-horizontal = false
- skips = false
- flip-colour = false
- learning-rate = 0.0001

3.2.1 Results

3.2.2 Without Skip-connections

- Train accuracy : 0.83
- Test accuracy : 0.82
- Mean per-pixel error: 0.06884613314821131
- Standard deviation of per-pixel error: 0.07722793233872194

3.2.3 With Skip-connections

- Train accuracy : 0.89
- Test accuracy : 0.89
- Mean per-pixel error: 0.06577824278159461
- Standard deviation of per-pixel error: 0.05120849701848935

./112/encoder-remove-weights/before.jpg

3.3 Task 2b: De-noising Model

3.3.1 Architecture and Training

I reused the architecture of the auto-encoder. The main focus was on the noise generation and modification of the dataset such that the model could learn to turn on level of noised flowers into a slightly less noisy representation. The option I decided for this in the end came down to engineering constraints although the results ended up quite interesti. The biggest issue is that the dataset for this is usually much larger as there

are multiple copies of the image at different noise levels. Initially I got around this by running multiple fits. Each fit of the model would run a different level of noise. I tried this going both ways, the model learning how to go from good to bad, then vise versa. This looked initially promising however, as would be a theme of this challenge, the metrics like loss, accuracy and there validation counter-parts ended up being less useful / not representative of results. I had turned the problem into a sequential learning task, each iteration potentially retraining the model to do different things.

My solution to this problem was to create a dataset which had a random representation of noise pairs within 0.1 standard deviation of gaussian noise over a given noise range. Typically this was from 0.0 to 2.0. The issue with this is that it does not contain images in all stages of noise, additionally there is a lot less data than you would want. To circumvent this somewhat I doubled the size of the dataset, and augmented this dataset with various changes. Things I tried included, swapping the colour channels, as well as flipping the image horizontally. I also added some variance where the image pairs that were created were not always 0.1 STD apart, and they could be within a range. Additionally as I was finding my images to be on the blurrier side. So I added a bias using a beta distribution to favour the less blurry images.

3.4 Hyperparameters

- image-size = 256
- epochs = 200
- noise-range = (0.0, 2.0) standard deviation
- load-from-file = false
- verbose = true
- encoder = false
- reg-wdecay-beta = 0.1
- reg-dropout-rate = 0.4
- reg-batch-norm = true
- data-aug = true
- flip-horizontal = false
- skips = false
- flip-colour = false
- learning-rate = 0.0001

3.4.1 Generation Process

For generation I found that I would often the image would not stabilise for many generations, I would go for 400 generations. To try to see what my model was doing, I decided to add some other types of base images instead of just using a normal distribution noisy image. These involve solid colour images, black, white, blue , red , green as well as different types of noise salt & pepper, speckle and perlin noise. I was especially interested in perlin noise as that is a noise used often for generating natural looking terrian / mountain scapes in games. I did not have time to explore of these

further, though here are some examples of images from different types of noise.

Using different images also exposed potential dud models weren't as bad as there performance on random noise would suggest, and could still create some interesting floral patterns from different starting images.

As I was finding it inconsistent to just use the model with the lowest loss, and to see if I could see how the models were training, I started generating models during the training phase at 10 epoch increments. This generally showed a trend for the generation to get more complex as the epochs grew, but also showed that sometimes the model would fall over and learn that creating all white image was the best it could do. I did create a diffusion model, I don't know if I would call it stable. Typically for each epoch range, the model would decide on one main colour / shape to make. No matter the starting colour of the image it would tend to make its way to its target image given enough generations

4 Conclusions and Future Work

- Summary of the key findings from both tasks.
- Discussion on the limitations of the current models and potential improvements.
- Suggestions for future research directions in the field of image classification and generation using deep learning.

Figure 1: Example figure showing model architecture or results

5 References