Software Design Project 4 Report

In this project, I was given the liberty to choose a dataset and train models of my choice on this dataset. I chose a TensorFlow dataset 'stanford_dogs' that consists of 20,580 images, split into ~60/40 training/testing data respectively with 120 different breed labels. My goal for this project was to train the 2 Lenet-5 CNN architectures used in the lecture and compare their results to those of another project I found online, I will refer to it as the AriClassifier.

I used the Lenet-5 model and the Alternate Lenet-5 model from the paper provided for the previous project. From the AriClassifier I used the mobileNetv2 and AlexNet-like models. The reason for training these models is to be able to produce an accurate model and classify the images well. Although classifying dog breeds isn't as detrimental as hurricane damage datasets it is still useful in the bigger picture of training models to classify images as specific labels. This can go further into facial recognition to help in Police searches whether it be for, missing persons, criminals, etc.

Data Preprocessing

As mentioned I used a dataset from TensorFlow, it was already split 60/40 into their respective training and testing data so I just needed to preprocess the images to assure the models can actually read them and manipulate the data. After looking at some of the image examples I noticed the dimensions of the examples were pretty varying which meant I needed to do some general reshaping. The data had been split into a specific datatype that I wasn't sure how to manipulate to change each the individual image dimensions. This led me to looking into other resources and I ended up finding the AriClassifier github!

As I know, for CNN models the preprocessing isn't as intricate as for ANN models seeing as we mostly just need to reshape the images and rescale them. My biggest issue was reshaping the images since the training/testing datasets had an odd datatype of "_PrefetchDataset element_spec…". After looking into the AriClassifier project I saw that they made some functions to manually go in and reshape the images. I used 3 of their functions to be able to reshape the images to be size (224,224,3), Ari had done an analyzation of the images and determined that these were the most frequent dimensions.

AriClassifier: https://github.com/aribiswas/stanford-dogs-classifier?tab=readme-ov-file

The 3 functions I used were `load_datatset, preprocess, and prepare`. Generally the `load_dataset` uses the TensorFlow dataset function to load in the dataset into its respective split datatsets. The `prepare` function essentially iterates through each image in the dataset while calling the function `preprocess` for each image. That is when the `preprocess` function takes the image and resizes it based on the size parameter given, in this case (224,224,3), and then rescales it by diving the image by 255, as we know for the RGB color values of 255 total. This function also one-hot encodes the labels to their respective images.

Model Architectures

For this project I implemented 4 different CNN model architectures 2 being from our lectures and 2 being from the AriClassifier: Lenet-5, Alternate Lenet-5 described in this paper, a MobileNetv2 model from the AriClassifier or as described here, and an AlexNet-like model described better here.

For the Lenet-5 Architecture I used the layers as shown below in figure 1. I used the template as done during lecture but had to change a couple of specifications in some of the layers. Such as making sure the input_shape parameter was the proper dimensions of the images I preprocessed (224,224,3). Making sure the Output layer actually output the proper amount of classes in this case being the 120 labels of breeds. Finally the most difficult change to which I needed help from ChatGPT for, changing the loss parameter in the compiler. At first my model wouldn't compile and I would get the error

`sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropyWithLogits}}]] [Op:__inference_train_function_16809]`

ChatGpt told me that it was because my labels were one-hot encoded and not represented as integers and recommended I change the loss to `categorical_crossentropy`. As a double measure I checked to see what loss parameter was used in the AriCLassifier and sure enough they also used categorical_crossentropy!

As seen below in figure 2, I used the same specifications for the Alternate Lenet-5 model as In Project 3. I didn't change much else compared to the Lenet-5 model, so it was smooth sailing for this one.

AriClassifier: https://github.com/aribiswas/stanford-dogs-classifier?tab=readme-ov-file

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 6)	168
average_pooling2d (AveragePooling2D)	(None, 111, 111, 6)	0
conv2d_1 (Conv2D)	(None, 111, 111, 64)	3520
conv2d_2 (Conv2D)	(None, 109, 109, 16)	9232
average_pooling2d_1 (AveragePooling2D)	(None, 54, 54, 16)	0
flatten (Flatten)	(None, 46656)	0
dense (Dense)	(None, 120)	5598840
dense_1 (Dense)	(None, 84)	10164
dense_2 (Dense)	(None, 120)	10200

Total params: 5632124 (21.48 MB)
Trainable params: 5632124 (21.48 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 1: Specifications for Lenet-5 Model

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_4 (Conv2D)	(None, 109, 109, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_5 (Conv2D)	(None, 52, 52, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
conv2d_6 (Conv2D)	(None, 24, 24, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 128)	0
flatten_1 (Flatten)	(None, 18432)	0
dropout (Dropout)	(None, 18432)	0
dense_3 (Dense)	(None, 512)	9437696
dense_4 (Dense)	(None, 120)	61560

Total params: 9740088 (37.16 MB)
Trainable params: 9740088 (37.16 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 2: Specifications for Alternate Lenet-5 Model

Although for the last 2 models they were a lot simpler to implement and not only because I just added them from AriClassifier, their structure was overall a lot less layers than the 2 I used. You can see the specifications for the MobileNetv2 and AlexNet-like in figure 3 and 4 respectively below.

AriClassifier: https://github.com/aribiswas/stanford-dogs-classifier?tab=readme-ov-file

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Func  (None, 7, 7, 1280)        2257984
 tional)

 global_average_pooling2d (  (None, 1280)              0
 GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 120)               153720

=================================================================
Total params: 2411704 (9.20 MB)
Trainable params: 153720 (600.47 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Figure 3: Specifications for the MobileNetV2 Model

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Func  (None, 7, 7, 1280)        2257984
 tional)

 global_average_pooling2d (  (None, 1280)              0
 GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 120)               153720

=================================================================
Total params: 2411704 (9.20 MB)
Trainable params: 153720 (600.47 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Figure 4: Specifications for the AlexNet-Like Model

Model Results

```
test_accuracy: 0.2260833382606506
test_accuracy: 0.8138333559036255
test_accuracy: 0.9929999709129333
test_accuracy: 0.0083333337679505
```

Figure 5: Accuracy results for each model going top-down from Lenet-5 to AlexNet-Like.

I only ran 10 epochs per model due to the computational cost of each model. At first I tried 20 and after 4 epochs and 10 minutes I realized 20 epochs didn't seem like a good idea. To my surprise the Lenet-5 model performed very poorly when I took into consideration how well it performed in Project3. I understand that 10 epochs was not substantial to allow the model to train properly and get good accuracy results but it still caught me by surprise. Especially after seeing that the Alternate Lenet-5 model performed almost 4x better than the Lenet-5 model in the same 10 epoch span!!

After seeing these relatively disappointing results I chose to give the benefit of the doubt that this dataset is more intricate and has over 100 labels for the images so these models aren't as well modeled for these types of datatsets. After seeing those results my hopes weren't high with the models I got from the AriClassifier but the MobileNetv2 model impressed me. Having an early show of over 0.9 accuracy by the 4th epoch was the best growth from the 4 yet. Even then this high accuracy didn't prepare me for the accuracy of the AlexNet-like model, being even lower than the Lenet-5 and even the VGG16 model from the early stages of my project. I had decided to keep the VGG16 model out of my project because of the computational cost and time it would take for what seemed to be very poor results. If only I were

AriClassifier: https://github.com/aribiswas/stanford-dogs-classifier?tab=readme-ov-file

to have know the results of the AlexNet-like model I would have kept it for that poor comparison.

Model Deployment

For my deployment I made a 3 new files, a Dockerfile, api.py file, and a docker-compose file. The api.py file has routes that explains what each model does and its accuracy. My Dockerfile is to build the container to allow for the use of my api inferences whereas the docker-compose is for the same but allows ease with running the container in the background!

Conclusion

This project gave me a broader perspective of the different architectures and models that can be used and taught me that just because an analysis has already been conducted with my dataset doesn't mean the models used are perfect, but also not to underestimate them based off of my own results either. I appreciated the AriClassifier for provided very clear instructions and files to replicate their results and use their models. With more time I would have enjoyed to implement an inference server and allow a user input an image and have their own dog classified. I will definitely be adding this on my own time in order to classify images of my late dog Lulu. I did this dataset in memory of her and am happy to have incorporated myself in a project as intricate as this one.

Resources

- Dataset: https://www.tensorflow.org/datasets/catalog/stanford_dogs
- AriClassifier:https://github.com/aribiswas/stanford-dogs-classifier?tab=readme-ov-file
- Debugging Reference: https://www.tensorflow.org/datasets/overview
- Alternate L5 Reference: https://arxiv.org/pdf/1807.01688.pdf
- MobileNetV2 Reference: https://arxiv.org/abs/1801.04381
- AlexNet Reference: https://arxiv.org/abs/1803.01164