Final Project Report - Doggo

David Stokes, Andy Vo, Mark McAfoose, Adriel Campos
GIT: https://github.com/Fallt0earth/Doggo_Classifier

I. INTRODUCTION AND MOTIVATION

The primary aim of the doggo classifier is to allow for a lightweight mobile deployable model to achieve acceptable accuracy in classification of dog breeds. This is motivated by general interest in the area as well as potential future development allowing for the project to be beneficial towards the group members resumes. The goal was to develop a deployable IOS application that allowed a user to take a photo of their dog and the model provide its best guess as to the breed.

II. APPROACH

The approach was relatively simple. The idea was to set up a supervised classification problem which could then be trained locally. Defining the acceptable accuracy was done through examining other authors' accuracy numbers with significantly more complex models such as VGG and Resnet-152. The team settled on a target accuracy of at least 50 percent due to these more complex models achieving around 80% accuracy.

The organization of the team was as follows: David developed the model in pytorch, Andy led development on IOS, Mark was in charge of IOS integration, and Adriel was a float intended to help whichever other team members needed assistance. The modeling work was completed after around 2 weeks at which point the IOS development began. Due to inexperience in IOS app development we elected to go for the minimum viable product for the app. This also meant the team had to accept that it might not be feasible to learn the skills needed in time for the project but this was a risk we decided to take.

III. DATASET AND TRAINING SETUP

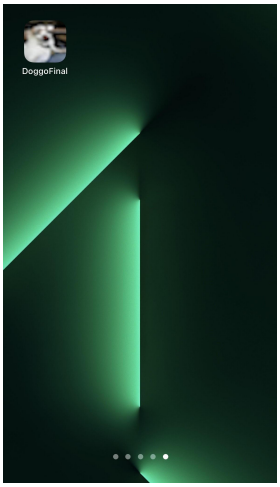
The dataset selected was the stanford dogs dataset, obtained from Kaggle. The dataset contains 20,580 samples of 120 breeds. The set came prelabeled and was selected for its wide range of dog breeds. The image sizes are 224x224 which did lead to problems in training time that will be discussed later.

Once the data was set up in a data loader, the search for the correct model began. The first model examined was shufflenetv2 as this is a lightweight model. We knew that heavier networks would be too cumbersome on a mobile device. Shufflenet did not provide adequate results hovering around 15 percent accuracy. The next model tried was the model suggested by Professor Hamed. This was mobilenetV3 which immediately provided much better results. Training was initially done with a simple SGD optimizer and Cross Entropy Loss but this was taking too long to converge. It did increase the stability of the training though in terms of loss values seen. After a closer examination of the publication of mobilenetv3 we attempted to use RMSProp with the exact same configuration that was used by the developers of mobilenetV3 but this led to massive instability in training likely indicating a missing hyperparameter or incorrectly tuned one. The next optimizer used was ADAM which proved to be the optimal optimizer for convergence with minimal interference on the stability of training. It was found that after just 75 epochs the model had converged and without significant additional compute it would be unfeasible to obtain higher than the 80% accuracy we did obtain.

The IOS development was a bit difficult for this project. The main requirement for this part of the project was having to learn a new programming language, Swift. Since the main goal was to build an IOS Application that could identify dog breeds, there were some challenges that were encountered during this process. The challenges were figuring out how to convert a Pytorch model into a MLModel and figuring out how to use an external application, XCode, to be able to build an IOS application. After a bit of research, there was success in being able to convert the Pytorch model into an MLModel and calling it into the application; however, there was no success in being able to get the results to display to the user. This application involved camera functionality and user photo library to be able to obtain an image from either one.

The photos below are a description of how the view controller is designed for the IOS application. There are two functional buttons for having the option to take a photo or choosing one from the library. After creating a few IBAction

functions for the buttons, the way the app was intended to work was to display the result of the dog breed along with the percentage of confidence in the center. If a fatal error occured in the application, it would display “Unexpected results” to the user else, it would display the confidence. Unfortunately, with the amount of time left, the IOS Developers were unable to get this app to properly work. This application is still a work in progress as a project for another day to complete.



IV. RESULTS

Overall the model took around 20 hours of training cumulatively to find the optimal solution to the problem.

1. The results of the work are encouraging with nearly 80% training accuracy and nearly 70% validation accuracy. Based on these results, they are considered as satisfactory for confirming the correct dog breed. The overall value of the predictions was pretty high with the confusion matrix having a few breeds it struggles to classify but primarily functions well. The model's inference time was nearly real time which suits its intended use as well.
2. The test image that is called to test the created model will be a Terrier dog breed. After using transforms to modify the image by resizing, normalizing, and converting it to a tensor, the output for the test image outputs ``Austrailian_terrier'' as the best choice for identifying the dog breed.

```
file_path = "/content/drive/MyDrive/Colab Notebooks/terrier.jpeg"
image = Image.open(file_path)
transform = transforms.Compose([
                                transforms.Resize((224,224)),
                                transforms.ToTensor(),
                                transforms.Normalize(torch.Tensor(mean),
                                                     torch.Tensor(std))])
image = transform(image)
```

```
[9]  model.eval()
     image = image.to(device)
     # print(img.shape)
     output = model(torch.unsqueeze(image, 0))
     breed = class_names[torch.argmax(output)]
     breed = breed[breed.index("-") + 1: len(breed)]
     print(breed)

     Australian_terrier
```

The result of the test image appears to be correct for identifying dog breeds as shown in the photo above.

V. LESSONS LEARNED

Many important lessons were learned throughout the course of this project.

A. Training Time

The dataset used for this project was enormous, and proved to be incredibly time consuming when developing the project code.

B. Working With IOS Applications

Apple, as a company, values their exclusivity greatly, i.e. it is nearly impossible to develop IOS applications on anything other than an Apple product. The team only had one Macintosh computer at its disposal, making the effort of creating an iPhone app extremely arduous. It is very important to consider the technological limitations of a project before deciding the proper course of action.

C. Technical Complexity

The team had no prior experience in coding using the Swift programming language. Despite the fact that many programming languages look very similar at a quick glance, the numerous differences and nuances to using a new language quickly become apparent after getting started for the first time. It should be taken into consideration how long it will take to learn the new material, and to set aside time for that. The mistake made for this project was trying to learn how the new language works and how to implement it at the same time.