Pet Classifier

Krishna Chaitanya Naragam (1001836274), Yeshwanth Teja Dity (1001823070)
The University of Texas at Arlington
300 W. First Street, Arlington TX 76019
krishnachaitany.naragam@mavs.uta.edu yeshwanthteja.dity@mavs.uta.edu

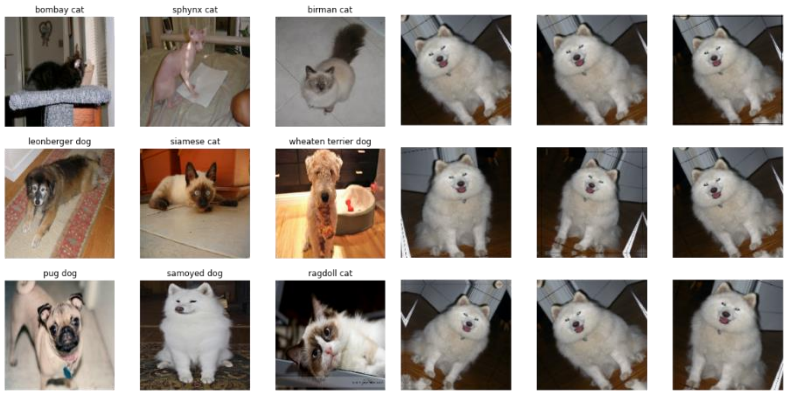
Abstract

This is a TensorFlow project for generic image classification. Multiple ways have been used to classify images using TensorFlow. The project was to try out that pre-trained images like ImageNet, mobilenet_v2 that promise to have significant improvement in training time and the accuracy and the traditional neural network which learns the weights by itself. This project classifies 37 different cat and dog breeds. The project also includes an android app, that classifies images from live camera feed in real time. This is made possible by the portability of TensorFlow images and the library being available in multiple programming languages and availability of mobile devices virtually everywhere.

Dataset

The dataset consists of high-quality images of pets of different breeds of cats and dogs. The dataset was originally collected by Omkar M Parkhi and Andrea Vedaldi and Andrew Zisserman and C. V. Jawahar of The Oxford-IIIT. They have created a 37 category pet dataset with roughly 200 images for each class. The images have a large variation in scale, pose and lighting. All images have an associated ground truth annotation of breed, head ROI, and pixel level trimap segmentation. The dataset is available to download for commercial/research purposes under a Creative Commons Attribution-ShareAlike 4.0 International License. The copyright remains with the original owners of the images, at, Visual Geometry Group - University of Oxford.

The dataset had all images placed in a single tar file and the file names were the actual class labels. There was some amount of pre-processing required to get the data ready for TensorFlow to classify easily, as TensorFlow has libraries that support dataset loading with a certain format. The process was to put each class in a different folder, the task was achieved using python code to operate on the operating system and create and copy relevant data to respective folders and have it ready for the dataset loader to load. The images also needed to be reshaped to same size as neural networks do not really work with images rather treat them as arrays of float like data and process them in batches. The sample of the data is here,



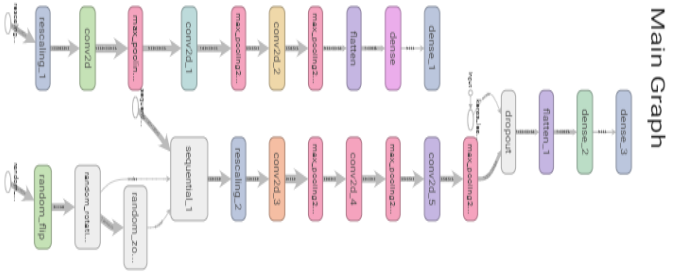
Implementation

To take on this project, the moto was to tryout image classification from scratch using neural networks and TensorFlow framework. The second moto was to use a pretrained popular image classification models available on TensorFlow to classify train the neural network and the classifier would tryout both methods and then select the one with better results to be used in the process of building an android app, that would classify live image data from the camera and show results on screen at real time.

To train the neural network we used several layers of network, the first implementation was to train the network from scratch in this we used various layers like, Conv2D, MaxPooling2D, Dropout, Flatten and Dense layers of the neural network models to help train the network. The images were also rotated for nuance in the data. All the images were rotated slightly and inserted back to the learning rules as the network needs to identify dogs or cats at any orientation of the mobile device. The network then looked like this,

Layer (type)	Output Shape	Param #
sequential_1 (Sequential)	(None, 180, 180, 3)	0
rescaling_2 (Rescaling)	(None, 180, 180, 3)	0
conv2d_3 (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d_3 (MaxPooling2	(None, 90, 90, 16)	0
conv2d_4 (Conv2D)	(None, 90, 90, 32)	4640
max_pooling2d_4 (MaxPooling2	(None, 45, 45, 32)	0
conv2d_5 (Conv2D)	(None, 45, 45, 64)	18496
max_pooling2d_5 (MaxPooling2	(None, 22, 22, 64)	0
dropout (Dropout)	(None, 22, 22, 64)	0
flatten_1 (Flatten)	(None, 30976)	0
dense_2 (Dense)	(None, 128)	3965056
dense_3 (Dense)	(None, 37)	4773

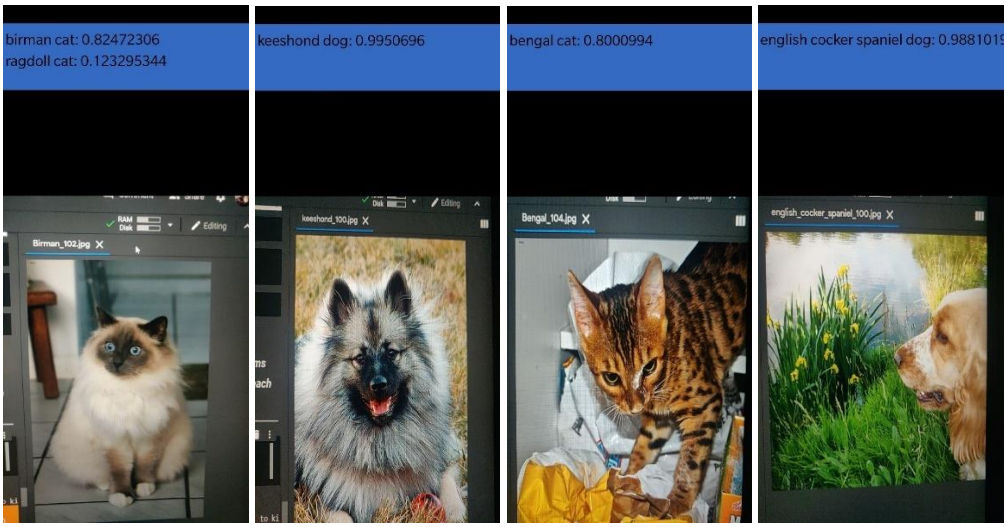
Total params: 3,993,413
Trainable params: 3,993,413
Non-trainable params: 0



The second method used a pre-trained network and used Transfer Learning to retrain the last layer using a Dense layer in TensorFlow and then use that to fit the given data. There are various choices in the pre-trained models where each network is good at some parts by itself. Some of the popular choices include, ImageNet, Inception, resnet, mobilenet. For this project, we chose mobilenet as the intention was to run the final classifier on mobile devices. The output was a file with a size of around 2MB that could be used on a mobile device easily and used to render results of classification from the neural network.

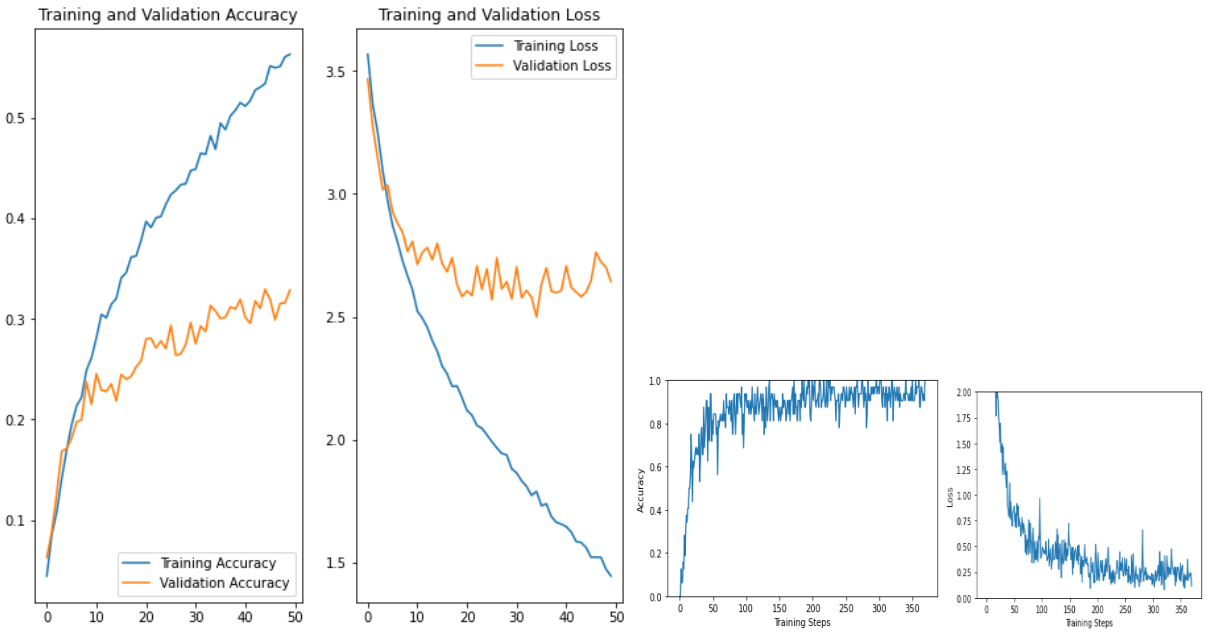
The prediction on mobile device was done using an opensource project that is supported by Google and TensorFlow community to build android apps that helps classify images using TensorFlow models. The model was reloaded and the built .apk file is included alongside the project submission archive.

The observation was that the Neural network that used weights from scratch was accurate to 80%. neural network that used the pre trained models from mobile net was accurate and 95% and took much less training time. in fact, the mobilenet only took two epochs to train network well similar performance was not achieved by the traditional neural network with hundred epochs. With this considerable difference in accuracies for retrain models and Motors that were built from scratch it is an obvious choice to choose the pre trained model as it was already ready with the ways that could detect shapes, sizes, colors, objects, and various other things. This made us select the pre trained model from mobilenet inside the Android app that was being built. Here are some screenshots form the android application running on a mobile phone,



Conclusion

The project concludes that the pre trained models are better when it comes to larger range of classes. well for smaller classes like, three to four classes did you run into that restaurant from scratch was good enough or was like that of a pretrained network. A pre-trained network adds hundreds of layers to the network which is not useful while classifying the data if the same data could be classified using a smaller network and achieve similar accuracies. here the task was to classify 37 different classes, so it was obviously a choice of choosing a network like image net or mobile net That was pre trained to classify shapes sizes colors on images. While traditional networks are still a thing, pre-trained models are taking over tasks of larger class data, as that would be a hectic and time taking task to build a similar network as they are well trained and available for free. The Training and Validation accuracy and loss for both the models is given below.



Self-Trained Model

Pre-Trained Model

References

1. Visual Geometry Group - University of Oxford
2. Examining the TensorFlow Graph | TensorBoard
3. examples/lite/examples/image_classification/android at master · tensorflow/examples (github.com)
4. Image classification | TensorFlow Lite
5. Object detection | TensorFlow Lite