Deep Learning Image Classification using Quantized Neural Networks

Krista Chanaritthichai
Erik Jonsson School of Engineering and Computer Science/Natural Sciences and Mathematics
The University of Texas at Dallas
Richardson, TX
kxc162230@utdallas.edu

Daniel Roberts
Erik Jonsson School of Engineering and Computer Science/Natural Sciences and Mathematics
The University of Texas at Dallas
Richardson, TX
dir170130@utdallas.edu

Guillermo (Alan) Romero
Erik Jonsson School of Engineering and Computer Science/Natural Sciences and Mathematics
The University of Texas at Dallas
Richardson, TX
gar180000@utdallas.edu

Dat Tran
Erik Jonsson School of Engineering and Computer Science/Natural Sciences and Mathematics
The University of Texas at Dallas
Richardson, TX
dmt170030@utdallas.edu

Alex Turk
Erik Jonsson School of Engineering and Computer Science/Natural Sciences and Mathematics
The University of Texas at Dallas
Richardson, TX
att170630@utdallas.edu

Sneha Venkatesh
Erik Jonsson School of Engineering and Computer Science/Natural Sciences and Mathematics
The University of Texas at Dallas
Richardson, TX
ssv170030@utdallas.edu

Abstract

The task of classifying images is a growing field in machine learning with the potential to be applied to many real world problems, but its resource intense nature has limited its usage on computationally limited devices such as smartphones. The objective of this project is to harness the power of deep learning but deploy it into a form factor that enables an Android mobile application to accurately and quickly classify images of routine objects. By using quantized neural networks and transfer learning in Tensorflow Lite, we are able to create a powerful yet lightweight deep learning image classification model that can be run locally on an Android device. This architecture can be similarly used on other resource-limited hardware such as IoT or embedded systems and can be easily applied to many other image classification problems.

.

Table of Contents

INTRODUCTION

For the first month of the semester, we worked through "CS231n: Convolutional Neural Networks for Visual Recognition" offered by Stanford University. This course began with a review of the fundamentals of supervised machine learning through linear regression. We revisited the traditional method of solving linear regression problems through the normal equations which provide a closed form analytic solution to find the line of best fit, but saw the limitations of the scalability of this process when working with many input features and its inability to detect highly non-linear patterns. To deal with this, we covered gradient descent, which provides an iterative process of minimizing a defined cost function that can be applied to a broad class of machine learning problems and scale to datasets with many input features.

After reviewing the basics of linear regression, we moved to supervised classification using support vector machines (SVMs). This process defines a number of hyperplanes to separate the data into distinct regions, which can allow for classification. To detect non-linear patterns within the dataset, we reviewed the topic of applying a kernel transformation to the input features, which then allows the hyper planes to separate the data points in this transformed feature space. In addition to learning the theory behind these traditional machine-learning models, we worked through the homework assignments of this course in which we implemented both iterative and vectorized approaches of linear regression and SVMs in Python using NumPy.

Following this, we moved into neural networks where we learned of their key advantages over the traditional machine learning algorithms with their capacity to learn the highly nonlinear patterns present within many real world problems. They are built off from the basic unit of the perceptron that takes in a weighted sum of inputs and then passes it through an activation function. This activation function being non-linear is crucial in networks being able to detect complex patterns and we learned of the advantages and disadvantages of various activation functions such as sigmoid or ReLU.

As powerful as neural networks can be, their flexibility comes at a cost and when used in practice they are highly apt to overfit and memorize the data that they have been trained on. To combat this, we learned about several regularization techniques to prevent over fitting such as weight decay (also known as L2 regularization) in which the model is penalized for having too large of parameter values. Another highly effective yet simple regularization technique specific to neural networks is dropout. Dropout involves randomly shutting off neurons within a neural network during the training process. It essentially distributes the process of learning across a variety of different nodes within the network.

The process of training these neural networks is quite complex, so we learned about the various descent optimization algorithms, the most simple of which being batch gradient descent. This involves passing the entire dataset through the network, finding the average cost, computing the gradient via back propagation and then updating the model parameters. In practice however, datasets are often very large and memory and computational constraints make this process infeasible, lending us to learn about stochastic gradient descent (SGD) in which the model is trained on a small random subset

of the data before updating its parameters. SGD allows for much faster model training and much less memory requirements. In addition to this, we also learned about the concept of momentum. Momentum involves adding a portion of the previous gradient to the current parameter update step and can allow for faster training times and smoother convergence to optima.

After covering the basics of neural networks, we then moved into learning about convolutional neural networks (CNNs), which are very commonly used when working with datasets consisting of images. Many attribute the great interest in deep learning to the AlexNet model outlined in Krizhevsky *et al.* 2012 which vastly outperformed the traditional image classification algorithms on the ImageNet dataset. CNNs consist of stacking convolutional and pooling layers atop each other. This allows the model to understand images in an increasingly abstract and hierarchical manner. The first layers of the network may be identifying edges, with following layers identifying shapes and textures, and the last layers identifying more abstract features of objects. Although the field of CNNs has progressed much since this seminal paper was published, Krizhevsky *et al.* 2012 set the standard for training CNNs and we largely followed many of the steps for our own implementation of an image classification model.

In practice while training a CNN, the images must be preprocessed before training the dataset. One component of preprocessing involves scaling the images such that they all are of the same dimensionality. We also learned about the process of data augmentation, which involves making small random class-preserving transformations to the images in the training set. These transformations often involve a combination of rotation, cropping, reflection, brightness adjustment, etc. and serve to increase the variety of data for the model to train on. This decreases the likelihood of overfitting (and thus can be thought of as a type of regularization) which can help increase the generalized performance of the model.

After gaining this thorough theoretical and practical understanding of the fundamentals of machine learning, neural networks, and CNNs from working through the Stanford CNN course, we set out to implement an image classification model to classify images of flowers as a proof of concept. For our choice of model, we used the MobileNet V2 model. This model had already been trained on the ImageNet dataset. By using transfer learning, we were able to take advantage of all the patterns that this pretrained model had already learned without having to bear the computational cost of training a model with more than 1 million images. For the training procedure, this involved retraining and fine-tuning the last few layers of the model to our particular dataset. This gave us a very powerful yet lightweight model that could be deployed on computationally limited hardware such as a smartphone with the aid of TensorFlow Lite.

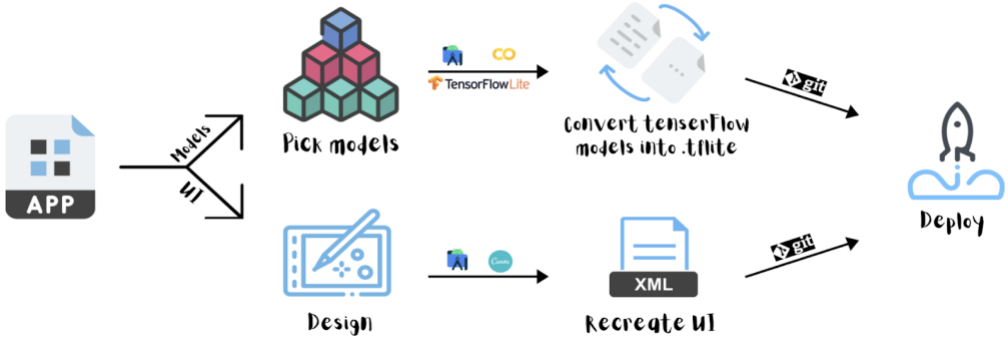
IMPLEMENTATION DETAILS

Model

The objective of the project was to classify routine objects using a mobile device using a proven and effective technology stack. It was imperative to avoid adding too many features and flavor-of-the-month frameworks. Our team agreed on developing for an android phone using Java, Gradle, and XML ecosystem, which are well supported on Android Studio. The creation of our model would need to be created using popular data science programming languages such as Python, R, and Stata. Given our great experience using the Google Collab (similar to Jupyter Notebook), the team developed our models with Python and Tensorflow Lite's TFlite model maker library.

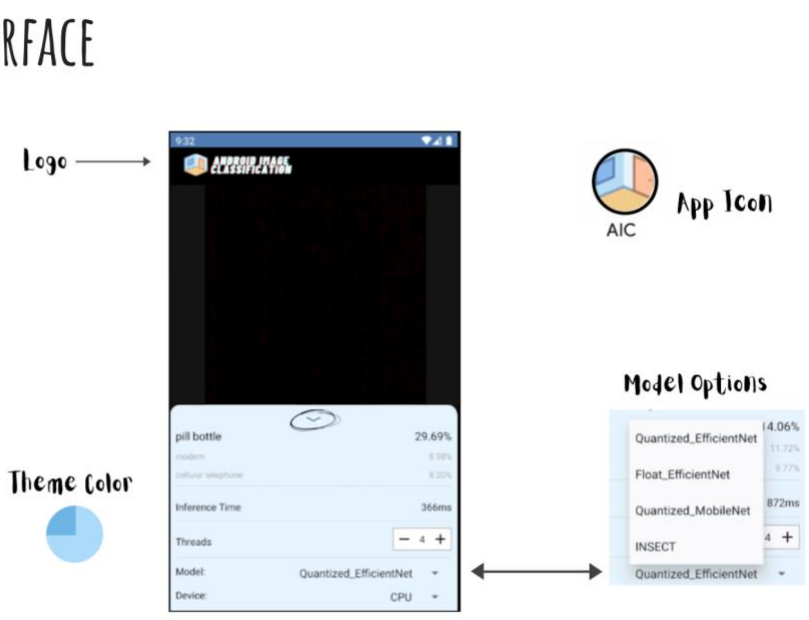
Our models are trained on either MobileNet or EfficientNet for transfer learning. We chose to focus on both of those options since both are trained on the popular ImageNet dataset. Our models focused on image identification of routine household items and insect classification obtained on Kaggle. We developed several models with varying degrees of accuracy. In order to increase the accuracy, we tried many methods. One method was using the Keras library for data augmentation. We artificially implement horizontal rotation and flipping to reduce overfitting. Another method was that we increased the performance on some models by fine-tuning the weights of the top layers of the pre-trained model (for example our MobileNet) alongside with the training of the classifier that was added. The training process enabled the weights to be tuned from generic features mapped to features associated specifically to our dataset.

The results indicated the Insect model had the fastest inference time and the best accuracy. That was expected because it was our smallest model. For the other three models, MobileNet which had a .tflite file size of 4.1MB had the smallest inference time compared to EfficientNet. EfficientNet had a .tflite file size of 18.1MB and had the longest inference time but produced a slightly higher accuracy. Our models are called by Java classes and are used in conjunction with XML to display the results.



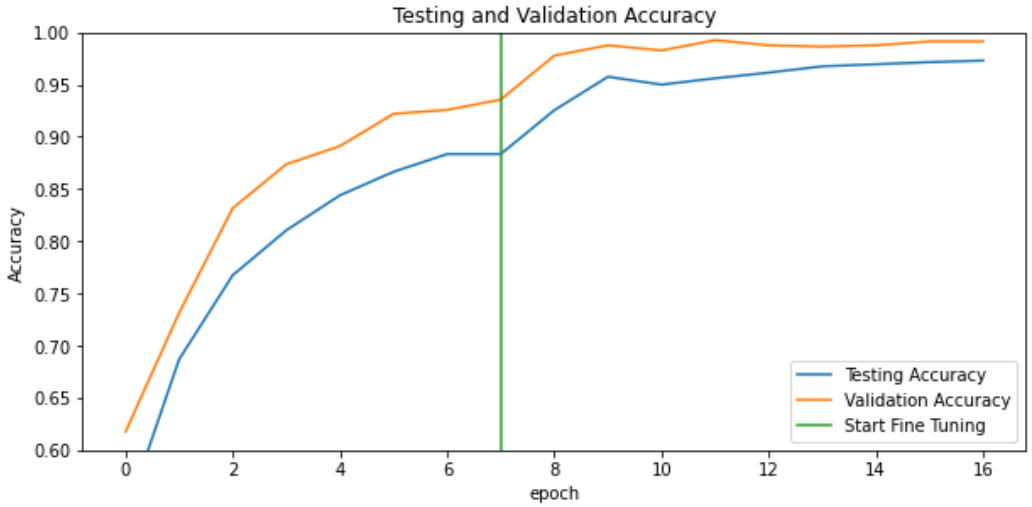
UI

Android Studio allowed the team to seamlessly add features that our app used such as the head banner, bottom banner and additional row creation. The user interface for the android application is structured in XML files. Our team used Android Widget and Camera library with Java that called the XML components that would be displayed. The layout editor in Android Studio also allowed us to modify dimension from XML layout using a drag-and-drop interface. With the help of Canva, graphic design platform software, we were able to create and customize our unique app logo app, alter the theme color for buttons and the bottom navigation drop-down. In order to make our models presentable, our team enabled the scroll down to have a menu of options. These options allowed the user to increase the thread count (which can go up to four), select one of our four models and switch between CPU and GPU (if the mobile device permission is configured). The output of the accuracy and inference are displayed in real time after toggling options.

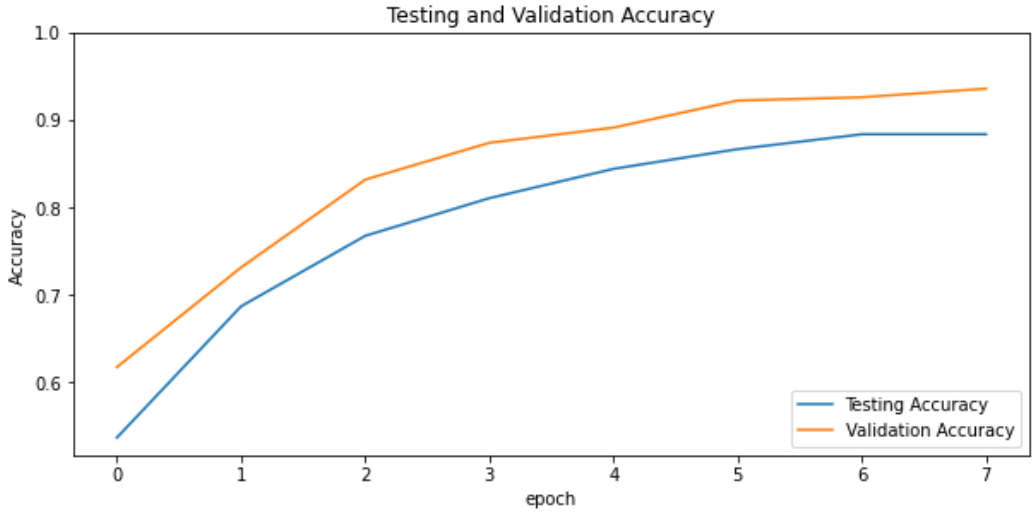


Results

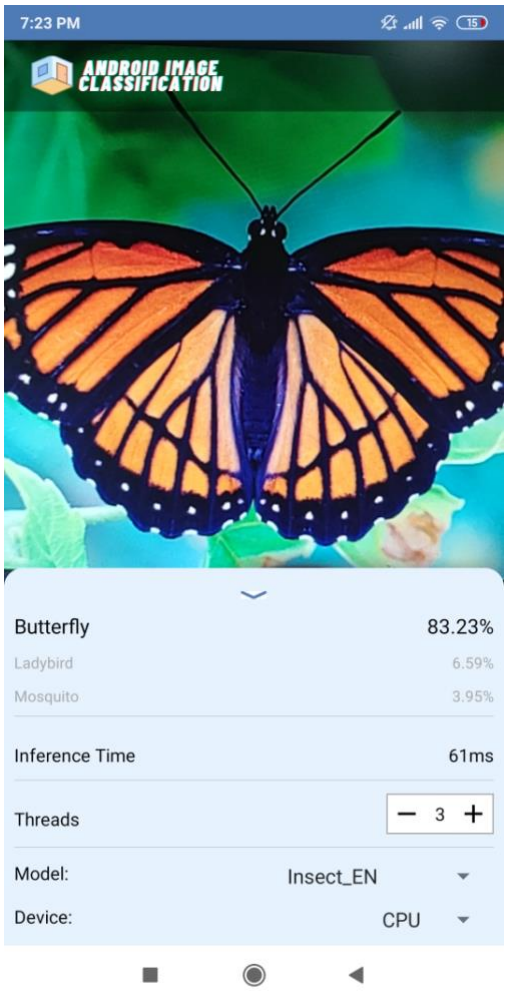
ImageNet



Insect



App pictures



FUTURE WORK

While our Android Image Classification (AIC) app could be a completed and deployable product, there are some additional features that could have been added. Due to time constraints we were unable to add certain features to our app that we had planned. Some of them include an in-app button to screenshot the current view, a capture button to save the current image being looked at, and access the device's camera roll. The ability to take a screenshot quickly or take a picture through the app could be incredibly useful in saving the information from the classification or just a picture of the object you are looking at. The ability to classify objects from your camera roll could also come in handy if you download the app if you have something from the past you want to find out more about. All of these features would give sight improvements to the overall functionality and accessibility of the app, but were not prioritized under the time constraints.

Along with these quality of life features there are base app improvements that we could add over time. The first of these features would be increasing the number of predictions shown. This feature could potentially provide the user with more information about the target object if there are many different predictions with similar levels of certainty. We would also implement additional models to select from, specifically some with shallower networks to increase speed and performance on older devices. Along with simpler models for increased performance, we will also incorporate more specified models similar to our insect model.

REFERENCES

Ali, H. (2020, September 16). Insects Recognition. Kaggle. https://www.kaggle.com/hammaadali/insects-recognition.

CS231n: Convolutional Neural Networks for Visual Recognition. Stanford University CS231n:Convolutional Neural Networks for Visual Recognition. (n.d.). http://cs231n.stanford.edu/.

Google. (n.d.). Recognize Flowers with TensorFlow Lite on Android | Google Codelabs. Google. https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android/#0.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (n.d.). ImageNet Classification with Deep Convolutional Neural Networks. Retrieved from https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

APPENDIX

Project: https://github.com/GARom/DS_project4

APK file:
https://drive.google.com/file/d/1LojQSEtgrWVpIqLkleUVPuToW3heFSfP/view