INDIAN INSTITUTE OF INFORMATION TECHNOLOGY VADODARA
GOVERNMENT ENGINEERING COLLEGE, SECTOR-28, GANDHINAGAR
GUJARAT - 382028

DESIGN PROJECT REPORT

ON

NXPico MX7 Based System for Object Classification and Recognition for Blind

Submitted By

Aashutosh Rathi (201651001)
Anshuman Verma (201651060)
Pritha Upadhyay (201651058)
Vaaibhavi Singh (201651051)

Under the supervision of

Dr. Kamal Kishor Jha

Abstract

In the present era, almost every new technology aims to make human life more comfortable. Vision is an NXPico MX7 based system for object classification and recognition for blinds. This device would help them to gratify their inquisitiveness about the beauty of the world and their excitement to explore it. Firstly, the image is captured using a camera module and send to NXPico MX7, from there it is sent to TensorFlow for classification. Subsequently, after processing the image a text to speech API is used to produce the output through the audio jack. Thus this smart system will ensure that knowing the object in front is just a button away for the blind people.

Index Terms *- NXPico MX7, TensorFlow, Transfer Learning*

1. INTRODUCTION

Blind people have an immeasurable curiosity about the world around them and one of the major obstacle faced by them in their daily life is identifying what is present in front of them. Vision aims to become their sight.

1.1. Problem Statement

Vision aims to help the blind people identify objects by producing output in the form of audio signals. The project's approach lies in developing a system based on NXPico MX7 which is capable of labeling objects with the help of TensorFlow libraries and converting the labeled text to speech using an API and producing output in the form of audio signals. Vision if connected by a power supply would also be accessible by any android device connected to it using Bluetooth. The scope also includes the code to be developed such that it could be used on any platform without any changes be it Raspberry Pi 3, NXPico MX7 or any other platform. It would be a system successfully performing real-time object detection with decent accuracy.

1.2. Major Challenges

- Collection of data-set for Transfer learning. Batch downloaded images were not sufficient and some were even not suitable which led to collection of images through self clicked pictures.
- Optimization of the Trained Model.
- Conversion of graph file *.pb* file to *.tflite* using TOCO(TensorFlow Lite Optimizing Converter). TOCO is not compatible with Windows OS therefore for this conversion Linux was used.
- The images captured by the camera module are processed in float format. Initially, the Android code was processing the UNIT8 format of images but later Standard Mean and Standard Deviation were introduced to convert UNIT8 parameters to float format.

1.3. Hardware and Software Used

- NXPico MX7
- NXPico MX7 Camera Module
- Micro Push Button
- 1K Resistor
- 15K Resistor
- General Purpose PCB
- Android Things
- Android Studio
- Tensor Flow

2. LITERATURE SURVEY

Google Developers Group Ahmedabad meet-up on Android Things: Things matter. The event report discuss about building professional and mass market products on Android Things without prior knowledge of embedded system designing. It includes the usage of MQTT protocols for controlling devices, including Raspberry Pi 3 Model B, breadboard and LED. It emphasizes on the usage Google Assistant for controlling devices and hence involved a brief introduction of Google Home, DialogFlow application, Firebase Real time database and cloud functions[1].

Android Things Hardware Manual is a long description of the NXPico MX7. This report contains a descriptive explanation about the Pico-IMX7 device which is a part of the Pico SOM series. It is an ultra-compact form factor SOM(System on module) optimized for mobile devices and IoT. It contains information about all the internally embedded components.It began with the overview of all the major components followed by the MX7 processor blocks. It combines the working technicalities of the various ports and pins. Pico is equipped with high-speed connectivity options engineered to support IoT endpoints and mobile terminals[2].

Transfer Learning and retraining of MobileNet model was studied through its official website. This tensorflow website contains information about retraining an Image Classifier using transfer learning. The article took a small data-set of flowers and added this data-set to the existing trained model by the process of Transfer Learning. They reused the feature extraction capabilities from powerful image classifier trained on ImageNet and simply trained a new classification layer on top[3].

Training a new model through transfer learning was not enough to make it work with mobile phones and hardware. A document based on TFLite was considered next. This document explains the conversion of a trained model to a TFLite format. It includes optimization of a re-trained model using TFLite converter and briefly explains the usage of TFLite interpreter. It mainly focuses on the program named as TOCO (TensorFlow Lite Optimizing Converter)[4].

3. THE PRESENT INVESTIGATION

The primary aim of this project is to create a hardware capable of identifying the objects around it based on the captured images and finally producing an audio output. Following is the descriptive explanation of our work while crossing each milestone.

3.1. Flashing the Android Things SOM onto NXPico MX7

The Android things kit contains a screen which is already built with Android kernel. We want this device to display the captured images and to print the interpreted output. For this functional behaviour, we need to first flash the Android Things Operating System to this device. We treated the screen as a simple Android device and flashed the Operating System in the Fast Boot Mode through a Linux platform. To flash Android Things on our screen, we began with downloading and installing Android studio and updated the Android SDK Platform Tool to version 25.0.3 from the SDK Manager. After this we booted out device in the Fast Boot Mode followed by the execution of a shell script flash-all.sh which installs the necessary bootloaders, baseband firmware(s) and operating system.

3.2. Android Code Development for Image Classification

The code development included the building of code for various component transitions. We began with creating a simple Hello World app and displaying it on the screen. We created a simple UI for the screen followed by the method for capturing the images through the camera module. We imported the image handler for dealing with the captured images. For the purpose of image classification, we created the Image Classifier Activity that classifies the objects based on the pre-trained model. We have used the MobileNet model $(MobileNet\_v1\_1.0\_224)$ trained on ImageNet data set. The captured images are sent to the model for classification purpose and then the result is printed on the screen along with the captured image. We displayed the 3 object names with highest percentage identification. We tested and verified the percentages by printing them on the terminal. Finally, we combined all the individual elements including camera handler, image classifier and image pre-processor.

3.3. Assembling the Devices and Building the Circuits

The major purpose for building a circuit was to include a push button in the hardware. It will help in clicking the images. The circuit is built on a General purpose PCB. We are supplying 5V power to the PCB through the 2nd pin of NXPico while the 9th pin is serving as the ground. We have included a LED in our circuit which will check the completeness of the circuit. The LED will blink each time we will click the image because one part of the circuit will break every time giving rise to another complete circuit. The completed circuit will send the captured image through the 31st pin GPIO. The fourth pin that we have used here is the 40th pin which is used for Control Area Network receive signal. CAN provides an inexpensive, durable network that helps multiple CAN devices communicate with one another. An advantage to this is that Electronic Control Units (ECUs) can have a single CAN interface rather than analog and digital inputs to every device in the system. The circuit includes a flex cable and a Type C data cable. The flex cable is used here to connect the camera module to the NXPico chip. We have also attached a screen in the present structure to have a look at the captured images and the output is also printed on that screen. The Type C data cable was used to connect the circuit and the whole hardware part with the Android Code. We used Android Studio and ran the code through that while the USB cable is connected to the NXPico chip through the type C data cable. The captured image is passed through the code using the GPIO pin. The images are sent to the Image Classifier activity, which then searches for the maximum percentage match. Top three objects are selected on the basis of

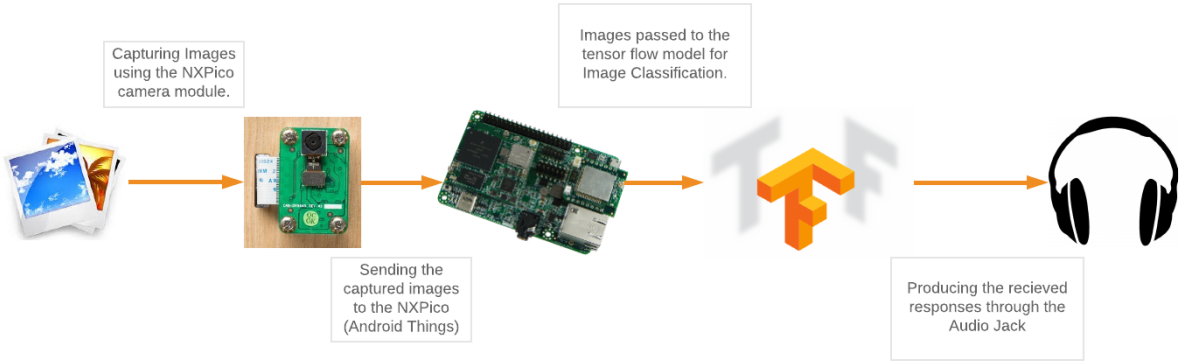


Figure 1: Flow of Vision

their confidence level.

PIN	Signal	V	I/O	Description
JP8_2	5V Power	5V	P	5V Power Supply
JP8_9	GND		P	Ground
JP8_31	GPIO	3V3	I/O	General Purpose Input Output
JP8_40	CAN_RX	3V3	I/O	Control Area Network receive signal

Figure 2: NXPico MX7 Pins

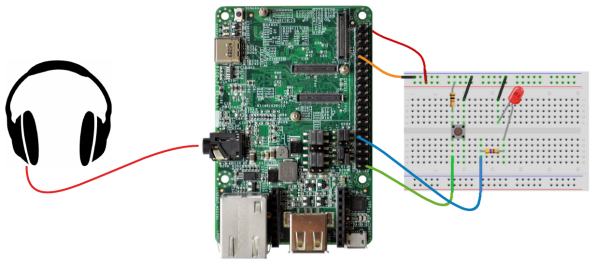


Figure 3: Producing Audio through the circuit

3.4. Producing Audio Output

Vision is built with a motive of bringing ease in this society. Visions works as a support system for the Blind. The only way by which we can communicate with blind people is through verbal communication. We are producing audio outputs as the names of the classified objects and are passing the audio through the audio jack which will help the blind to recognize the world around them. For the purpose of producing audio outputs, we have created separate methods. We have imported the TextToSpeech API and have created TTS speaker Activity. After processing the images and classifying them, the output labels are further sent for text to speech conversion which are then passed through the audio jack.

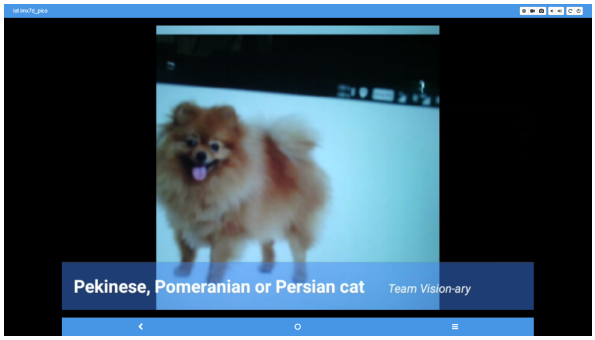
3.5. Retrain the Image Classifier Using Transfer Learning

Now we were ready with a working project. The next task was to create a better model with respect to a blind person. We focused on the major difficulties faced by them such as identifying the currency notes, chair, table, road, zebra crossing and many more. We also considered dangerous weapons like knife and guns. This task of building a new model began with the collection of large sets of images and exploring the concept of Transfer Learning. We collected large number of images through batch downloading Google Chrome Extension. But collection of images through online sources were not enough and hence we also clicked real life pictures. We also collected large number of images by creating short videos and then converting them to image format. We collected around 500 images of each category and arranged them in separate folders. Modern image recognition models have millions of parameters. Training them from scratch requires a lot of labeled training data and a lot of computing power (hundreds of GPU-hours or more).Transfer learning is a technique that shortcuts much of this by taking a piece of a model that has already been trained on a related task and reusing it in a new model. We have used the MobileNet model $(MobileNet\_v1\_1.0\_224)$ for Transfer Learning. We reused the feature extraction capabilities from this pre-trained model and trained a new classification layer on its top.

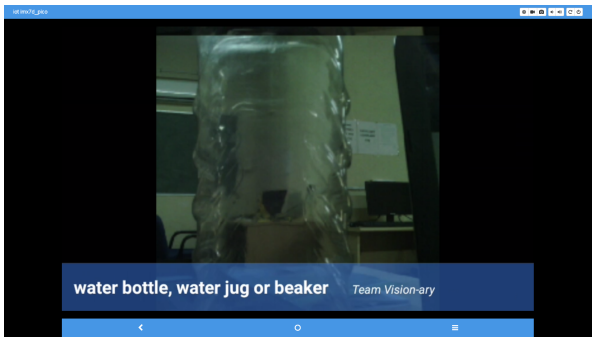
This process involves many small phases. The first phase analyzes all the images on disk and calculates and caches the bottleneck values for each of them. TensorFlow Hub is used to ingest the pre-trained pieces of models. In TensorFlow Hub, Bottleneck values are referred as image feature vector. This layer has been trained to output a set of values that's good enough for the classifier to use to distinguish between all the classes it's been asked to recognize. It has to contain enough information for the classifier to make a good choice in a very small set of values. The reason our final layer retraining can work on new classes is that it turns out the kind of information needed to distinguish between all the 1,000 classes in ImageNet is often also useful to distinguish between new kinds of objects. It is preferred to cache these bottleneck values on the disk because every image is processed multiple times during training and calculating each bottleneck takes a significant amount of time.

After completing this phase of bottlenecks, actual training of the top layer begins. The script that we have used here for retraining runs 4000 training steps by default. Each step chooses ten images at random from the training set, finds their bottlenecks from the cache, and feeds them into the final layer to get predictions. Those predictions are then compared against the actual labels to update the final layer's weights through the back-propagation process. As the process continued, we observed improvement in the reported accuracy. After completing all the steps, a final test accuracy evaluation was run on a set of images kept separate from the training and validation pictures. This test evaluation is the best estimate of how the trained model will perform on the classification task. The TensorBoard visualization helped us in observing and understanding the retraining process. Now, we have our new trained model with our categories stored as output_graph.pb and a text file storing the labels was also generated. The new model contains both the TF-Hub module inlined into it, and the new classification layer.

Now, our next target was to deploy this trained model into our hardware. So, we wanted to make it compatible for mobile devices. For using a trained model in mobile device we need to create an optimized and redundancy free model foramt, one that already exist is '.tflite'. To convert our graph.pb file to model.tflite, we used TOCO. But before getting started with toco we need to freeze graph to remove redundancy. TOCO is not compatible with windows architecture and hence we used a separate OS for this conversion. While toco has advanced capabilities for dealing with quantized graphs, it also applies several optimization that are still useful for our graph, (which does not use quantization). These include pruning unused graph-nodes, and performance improvements by joining operations into more efficient composite operations. The pruning is especially helpful given that TFLite does not support training operations yet, so these should not be included in the graph. TensorFlow uses Protocol Buffers while TFLite uses Flat Buffers. The primary benefit of Flat-Buffers comes from the fact that they can be memory-mapped, and used directly from disk without being loaded and parsed. This gives much faster start-up times, and gives the operating system the option of loading and unloading the required pages from the model file, instead of killing the app when it is low on memory.



(a)



(b)

Figure 4: Custom objects testing

3.6. Testing for our own Custom Objects

After the training comes testing of the model which was done by us by clicking images for a plenty of real-time objects. We are sharing some results here, for these objects we have trained the model on our own. First is a water bottle, that is detected as a water bottle, water jug or beaker by the device as shown in Figure 4.

Second one is a dog detected as a pekinese,

Pomeranian or Persian cat by the device as shown in Figure 5. Both of the detection made by vision are accurate.

4. RESULTS AND DISCUSSIONS

The training accuracy shows what percent of the images used in the current training batch were labeled with the correct class. On the other hand, the validation accuracy is the precision on a randomly-selected group of images from a different set. The key difference is that the training accuracy is based on images that the network has been able to learn from, so the network can over-fit to the noise in the training data. A true measure of the performance of the network is to measure its performance on a data set not contained in the training data, this is measured by the validation accuracy. If the training accuracy is high but the validation accuracy remains low, that means the network is over-fitting and memorizing particular features in the training images that aren't helpful more generally. In our case Figure 6 shows the graph of training and validation accuracy. The difference between resultant training and validation accuracy is 0.025, which is within an acceptable margin.

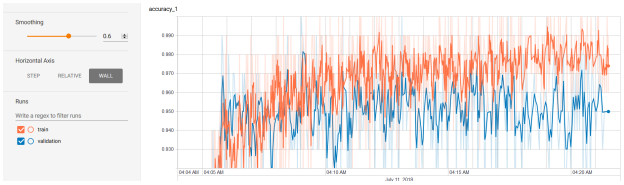


Figure 5: Training vs Validation Accuracy

Cross-entropy is a loss function which gives a glimpse into how well the learning process is progressing. The training's objective is to make the loss as small as possible, so that we can tell if the learning is working by keeping an eye on whether the loss keeps trending downwards, ignoring the short-term noise. The resultant cross-entropy graph is shown in Figure 7 where the difference between loss during training and validation is 0.1, that is within an acceptable margin.

Bias is changed continuously to reduce the error due to it and this error is measured by taking the difference between the expected prediction of our model and the correct value which we are trying to predict. Bias measures how far off, in general, the model's predictions are from the correct value. The summary of biases for training and validation are shown in Figure 8 and 9.

Neural networks are trained iteratively using optimization techniques like gradient descent. After each cycle of training, an error metric is calculated based

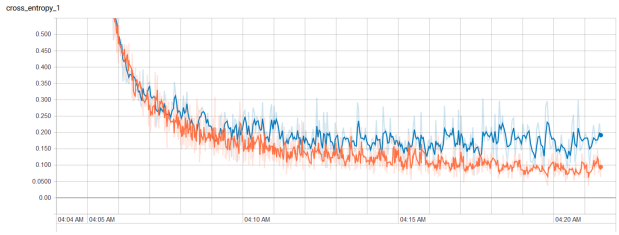
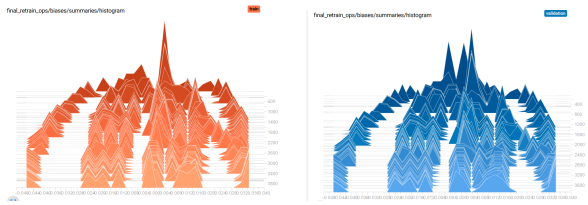


Figure 6: Training vs Validation Cross-Entropy



(a)

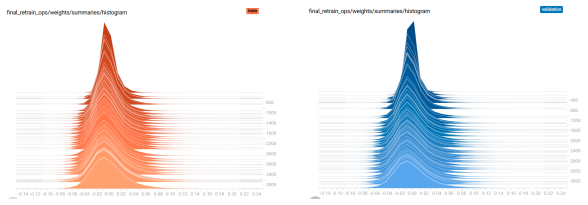
Figure 7: Biases Summary (Training vs Validation)

on the difference between prediction and target. The derivatives of this error metric are calculated and propagated back through the network using a technique called back-propagation. Each neurons coefficients (weights) are then adjusted relative to how much they contributed to the total error. This process is repeated iteratively until the network error drops below an acceptable threshold. The summary of biases for training and validation are shown in form of histograms in Figure 10 and 11.

The above discussion leads to an inference that we were able to make Vision as a real-world application that is successfully performing real-time object classification with decent accuracy, but at the same time their lies scope for improvement in our model whether it is it's portability, accuracy, reliability, bulkiness or speed.

5. CONCLUSIONS AND FUTURE SCOPE

Vision is simple in design, easy to assemble and reliable to a great extent. If the data-set used during



(a)

Figure 8: Weights Summary (Training vs Validation)

training is increased then the accuracy of the system while testing will also increase. Moreover, the usage of NXPico MX7 has optimized the system as compared to conventional systems. A blind person will be able to determine the object in front of him through audio signals simply by clicking a button on the system and if the system is connected by a power supply then it could be accessed through any Android device connected to it using Bluetooth. The Android code developed is such that it can be used on any platform without any changes in it, including Raspberry Pi 3, NXPico MX7 or any such platform. As the results have demonstrated, it is a system successfully performing real-time object classification with decent accuracy.

In the near future, the bulkiness of the system can be reduced. Also, the speed, accuracy and the reliability of the system can be improved. Integrated Circuits can be used for its conversion into a portable device that the blind person can carry with him/her all the time. Vision can also be modified to respond to the instructions given by a blind person. This modification can lead to the development of features like navigation, calling or passing a message to some person and many more. The future of this device is full of new possibilities.

ACKNOWLEDGMENT

We are thankful to Indian Institute of Information Technology Vadodara, for providing us the computational and infrastructure facilities.

This project could not have been completed without the guidance of our mentor- Dr.Kamal Kishor Jha. We are extremely thankful for his support and constructive comments. It would have been very difficult for us to achieve our goal in the given duration without his effective directions.

A special thanks to Mr. Jitu Sharma for his efficient guidance and eminent support while working with hardware components.

References

[1] Android Things: Things matter, https://www.meetup.com/GDG-Ahmedabad/events/249860953/

[2] Pico i.MX7 Development Kit for Android Things Hardware Manual, https://www.nxp.com/docs/en/user-guide/PICO-IMX7D-USG.pdf

[3] How to Retrain an Image Classifier for New Categories, https://www.tensorflow.org/tutorials/image_retraining

[4] TensorFlow for Poets 2: TFLite Android, https://codelabs.developers.google.com/codelabs/tensorflow-for-poets-2-tflite/#0