Final Project Report, ENGR 859, Spring 2023
Gator Vision - Using Machine Learning to Enhance SFSU's Visiting Experience

Emanuel Francis[*]
SFSU

Abstract

This research paper investigates the effectiveness of utilizing a smartphone camera and the a machine learning model to identify specific art pieces on the San Francisco State campus. While the college already implements several strategies to make the information about these art pieces public; students, staff, and visitors still are left ignorant of what these art pieces are actually representing. The goal of this research is to create an application that could help not only identify these pieces but also help expose some of the programs and facilities the university has to offer.

1 Introduction

PI Qin has proposed a deep neural network decoupling method for heterogeneous learning tasks, which can significantly reduce the computation overhead of DNN models [2], and built the very first decentralized DL framework [1]. San Francisco State University is a public institution that utilizes a variety of methods to expose new/existing students to information about art, facilities, events, etc. However, there still seems to be a gap in the information. Tools such as social media, virtual tours, and in-person tours help visitors learn about the campus generally. But certain, resources or key marker landmarks remain a mystery to even graduating seniors. In this paper, I will be exploring the method of using a phone camera and machine learning to identify specific art pieces located throughout the campus.

2 Implementation

This project aims to use MobileNetV2, a convolutional neural network to run a TensorFlow lite image classification model on an Android device. MobileNetV2 is 53 layers deep, using the pre-trained model we are allowed to implement a technique known as transferred learning. Transfer learning allows us to modify any layer of the model to improve the model's accuracy.

2.1 Part 1 - Custom Dataset

Image Classification models require a large and diverse dataset to train the model. For this project, I choose to record videos of 3 different art objects on the San Francisco State campus. Due to the quantity of images needed to properly train a model, it is more efficient to record a 1 min long video and extract 60 frames than to take 60 individual photos.

Using a smartphone camera, I recorded a video of walking around the object. Once satisfied with the angles, I used a software called "FFmpeg" to extract the frames. When extracting the frames it is important to choose a "long" duration between frames. For example, if you choose to take 60 frames per second. This would give you a large dataset, but all the images are too similar. To successfully train the model, the images must contain many different angles, brightness, orientation, etc. Once the frames were extracted, they were separated into groups, trained and validation datasets, and once more into their respective classes.

[*]email:efrancis1@mail.sfsu.edu

2.2 Part 2 - MobileNetV2 Transferred Learning

MobileNetV2 is a convolutional neural network that was designed to perform efficiently on mobile devices. The pre-trained network can classify over 1000 objects. Using transferred learning we can choose any layer of the model and retrain it to recognize the art subjects on the SFSU campus. The code below demonstrates us removing the top layer of the model and replacing it with our own trained layer.

```
base_model=MobileNetV2(weights='imagenet',
    include_top=False, input_shape=input_shape)
```

We are not looking to retrain an entirely new model here but to use the "experience" from the pre-trained model to help identify the objects in our use case. The code below demonstrates us freezing all the other layers so that we are not starting from scratch.

```
# Freeze the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False
```

2.3 Part 3 - Image Augmentation

Before the images can be trained, they must be augmented and reformatted to match the train model's input values. For the model to process the images correctly they must first be their correct resolution which is specified by the creators of MoblieNetV2. The inputs for each layer require a "shape" in this case the shape is (224,224,3). The first two numbers represent the image dimensions and the last number indicates the number of classes.

```
# Set the input shape and number of classes
input_shape = (224, 224, 3)
num_classes = 3
```

As explained previously, we want the model to be able to process images under various conditions. Meaning it should classify an image that is tilted, slightly dark, too bright, etc. Due to the large dataset, it is not feasible to edit each individual image. Instead, we will use a function built within "tensorflow.keras.preprocessing". ImageDataGenerator will help rescale, rotate, flip, etc the images. Below you will find the code that demonstrates some of the settings used to preprocess the images.

```
# Create the data generators for training and
    validation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'


)  # Normalize pixel values between 0 and 1
val_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)
```

2.4 Part 4 - Arduino Nicla Vision

The Arduino Nicla Vision is a powerful microcontroller that can be used to perform image classification tasks in real time. It is ideal for applications such as asset tracking, object recognition, and predictive maintenance. It also has a powerful STM32H747AII6 Dual ARM Cortex M7/M4 IC processor that can be used to train and run a CNN model [6].The Nicla Vision also comes with a 2MP camera that could be used with the real time image classification.

3 Evaluation

The model was completed with about 81% accuracy. However when tested on the android application the app guessed the status correctly less than half of the time. In this next section, I will be reviewing some of the shortcomings that caused the inaccuracy. And try to pinpoint strategies that will improve the model's accuracy in the future

3.1 Computational platform and Software Environment

The CNN model was written and compiled using Jupiter Notebook on a Ubuntu Laptop. This made importing the data training sets simple. That said after increasing the number of training epochs, the model would crash mid-way due to the aging laptops limited hardware.

3.2 Methodology

- Use TensorFlow to train our dataset on the first layer of the MobileNetV2 model and freeze the other lower layers.
- Convert the trained model into a model and test its accuracy.
- Implement more rigorous fine-tuning methods or add more variety to the training/validation set.
- Create a TensorFlow Lite file that could be used in Android Studios and test real-life accuracy.

3.3 Experiment 1 - EfficientNet

When creating the model, I for some reason was confused about the training and validation sets. When I finally got all the settings down and the file compiled with no issues, the accuracy graphs made no sense. After 1 epoch, the accuracy would hit 100% immediately. The only way this is possible is that the training images and the validation images match each other. This was indeed the case, the training and validation sets had the same path, which caused the issue.

At this time I was using the EfficientNet model. What cause me to switch models was that after the models were converted to a Tensor flow Lite file, the input shapes would not match. This would cause the application to crash whenever the model was in use.

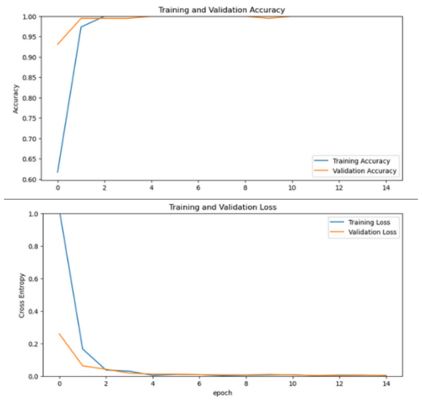

Figure 1: Accuracy of the model hitting 100%

```
    \cfun main() {
    // Load the model
    val modelPath = "path/to/model.tflite"
    val modelBuffer = FileUtil.loadMappedFile(modelPath)
    val interpreter = Interpreter(modelBuffer)

    // Prepare input tensor
    val inputShape = interpreter.getInputTensor(0).shape()
    val inputBuffer = Array(inputShape[0]) { FloatArray(inputShape[1]) { 0f } }
    // TODO: Fill the inputBuffer with your input data

    // Run inference
    val outputShape = interpreter.getOutputTensor(0).shape()
    val outputBuffer = Array(outputShape[0]) { FloatArray(outputShape[1]) }
    interpreter.run(inputBuffer, outputBuffer)

    // Process the output
    // TODO: Add your post-processing logic here

    // Clean up
    interpreter.close()
}
```

Figure 2: Sample code provided by Tensorflow Lite

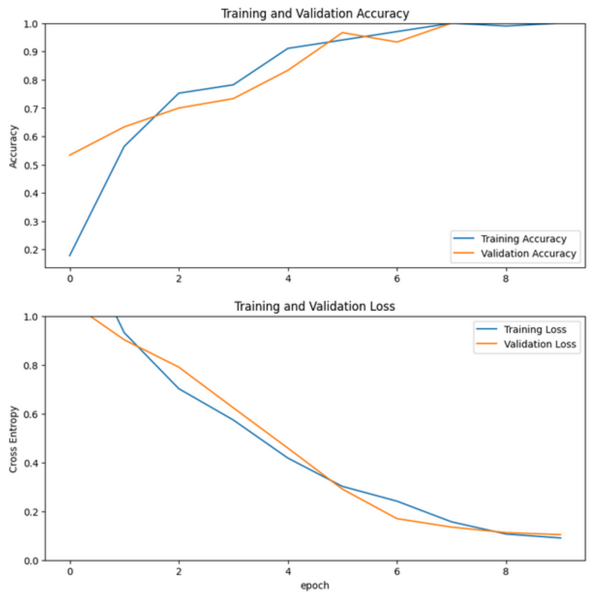

Figure 3: In this case, the number of batches are increased and the validation/training set are now correctly organized. As you can see, there is now a more gradual curve in the learning process.

3.4 Findings and Discussion

In the end, I realized that the training datasets were not diverse enough. I recorded only about 30 seconds of video and exported frames every 1-2 seconds. When recording I needed to make the images as different as possible. I also used an iPhone camera instead of an android with also changes the way the model interpreters the photos.

ACKNOWLEDGMENTS

The authors wish to thank A, B, C. This work was supported in part by a grant from XYZ. *Comment:* you don't need this section in final project report, but you will need it when you create a final, camera-ready version of a technical paper for publication.

REFERENCES

[1] San francisco state university. *U.S. News & World Report*.
[2] G. Grinstein, D. Keim, and M. Ward. Information visualization, visual data mining, and its application to drug design. IEEE Visualization Course #1 Notes, October 2002.