Project 1 - Object Detection
ECGR 4127
Lauren Bourque, Adam Shigapov

Introduction

The purpose of Project 1 is for students to create and experiment with an object detection system using a machine learning model deployed to an Arduino board. In this project, the team intends to create a machine learning model in Tensorflow that can be uploaded to the Arduino TinyML kit (with Nano33 BLE and camera) with the goal of predicting/detecting when a target object is the intended image. For this project, our intended class of image is a frog. The model will predict whether or not an image it detects is a frog.

This report contains an in depth breakdown of the model architecture and design. Dataset creation and training strategies are discussed, as well as the results and performance of the model. Our report includes a summary table of the model's results and architectures containing accuracy, false rejection rate, false positive rate, number of parameters, MACs, input tensor shape, and sampling rate. Additionally, this is followed by a discussion on the results of the model, as well as a summary of the project's key takeaways.

Model Architecture

The input shape was chosen to be 32x32x3 because images were chosen from the CIFAR-10 dataset, which contains images of size 32 by 32 with 3 channels for red, green, and blue. The model was broken into sections that contained two convolutional layers before a max pooling and dropout were performed. The two convolutional layers were chosen before the max pooling to recognize patterns better and extract features. The ReLU activation function was chosen for these convolutional layers because of its computational efficiency. Batch normalization was added after each convolutional layer to normalize the inputs, improving the speed and stability of training and keeping the inputs at a consistent scale to address the vanishing gradient problem. Dropout was added at the end of this section to prevent overfitting, especially due to the large number of parameters in the model.

Three of these sections were created with two convolutional layers. Models were tested that contained four or five of these sections, but training time was significantly slower and the accuracy was reduced significantly. These designs added unnecessary complexity to the model. After these three sections, the flatten layer was added to transfer the output from the multi-dimensional convolutional layers into the one-dimensional dense layers. Two dense layers were added to extract more features. 512 and 256 neurons were chosen for these layers to allow the model to have a better capacity to learn from patterns in the data. Dropout layers were added again to prevent overfitting. A sigmoid function was chosen to classify the output between 0 and 1. The model summary can be seen in Figures 1 and 2 below.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 batch_normalization (Batch  (None, 30, 30, 32)        128
 Normalization)

 conv2d_1 (Conv2D)           (None, 30, 30, 32)        9248

 batch_normalization_1 (Bat  (None, 30, 30, 32)        128
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 dropout (Dropout)           (None, 15, 15, 32)        0

 conv2d_2 (Conv2D)           (None, 15, 15, 64)        18496

 batch_normalization_2 (Bat  (None, 15, 15, 64)        256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 13, 13, 64)        36928

 batch_normalization_3 (Bat  (None, 13, 13, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)
```

Figure 1: Model Summary Part 1

```
 dropout_1 (Dropout)         (None, 6, 6, 64)          0

 conv2d_4 (Conv2D)           (None, 6, 6, 128)         73856

 batch_normalization_4 (Bat  (None, 6, 6, 128)         512
 chNormalization)

 conv2d_5 (Conv2D)           (None, 4, 4, 128)         147584

 batch_normalization_5 (Bat  (None, 4, 4, 128)         512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 128)         0
 g2D)

 dropout_2 (Dropout)         (None, 2, 2, 128)         0

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 512)               262656

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_4 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 1)                 257

=================================================================
Total params: 683041 (2.61 MB)
Trainable params: 682145 (2.60 MB)
Non-trainable params: 896 (3.50 KB)
```

Figure 2: Model Summary Part 2

Data and Training

Our data was pulled from the CIFAR-10 dataset and classified into frog and non-drog images. We chose this dataset to have easy access to a large dataset of images without having to manually create a dataset. We deleted 30,000 images from the non-frog class to better balance the dataset and provide a better prediction. An ImageDataGenerator was used for data augmentation to skew the images and provide better training. We performed rotation, width shift, height shift, shear, zoom, horizontal flip, and fill mode. We added early stopping to our training to prevent overfitting during training.

Results

The final training accuracy was found to be 0.8830, the validation accuracy was 0.8967, and the test accuracy was 0.9108. The model does exhibit some false positives such as occasionally predicting a human to be a frog, but it does respond to the target. When the camera was shown an image of a frog, it was able to detect it. The model took about 18-20 seconds to train each epoch and took about 18 seconds to upload the code and run. A plot of the accuracy and confusion matrix can be seen in Figures 3 and 4 below.

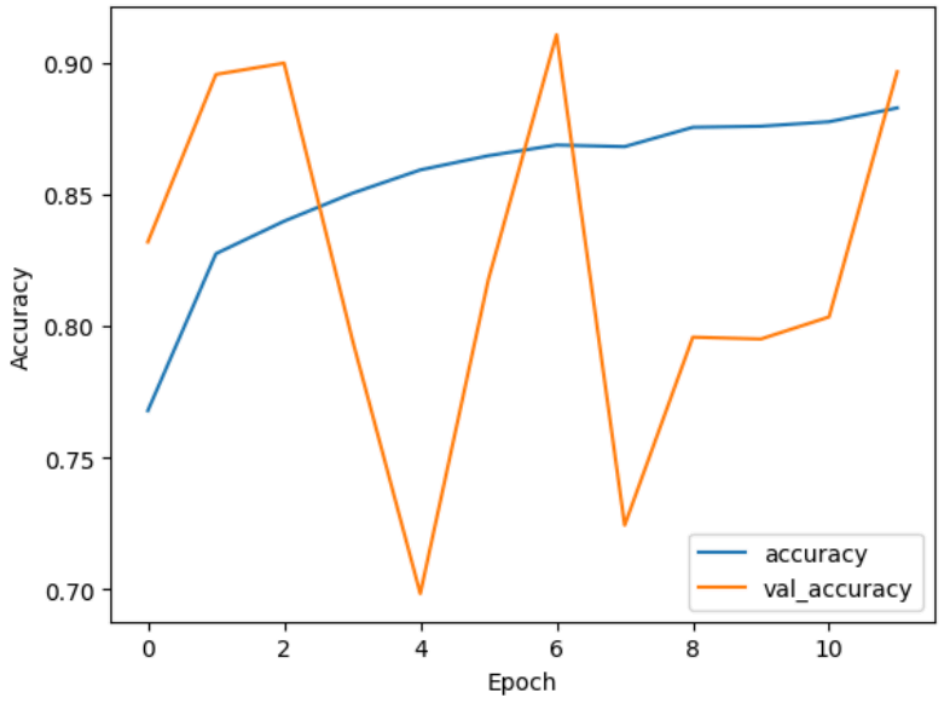

Figure 3: Model's Validation and Training Accuracy

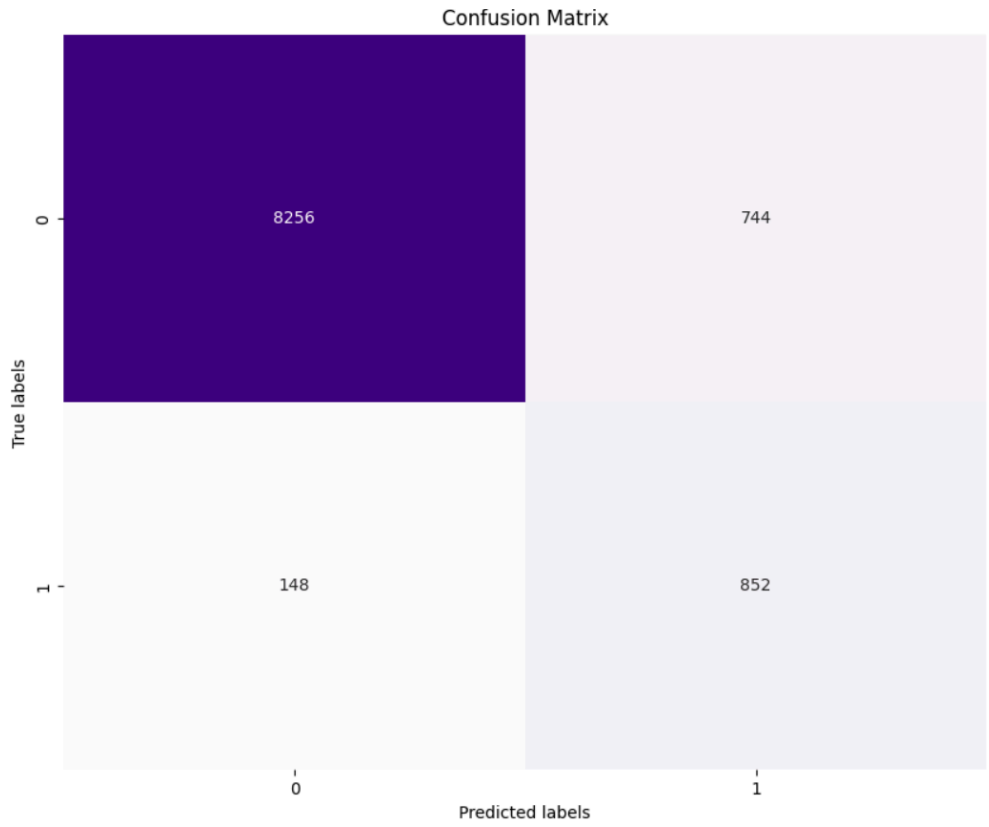

Figure 4: Confusion Matrix

Summary Table

Training Accuracy	Validation Accuracy	Test Accuracy	False Rejection Rate	False Positive Rate	Num. of Params.	Num. of MACs	Input Tensor Shape
88.3%	88.3%	91.1%	1.48%	7.44%	683041	14,717,928	32x32x3

Discussion

Overall, the model performed well and had a high test accuracy. It does seem useful, but could be improved. There were several instances where the model falsely predicted the ceiling or a human face as a frog. A larger dataset could be used to train the model and provide a wider range of non-frog images, but more images of frogs would be needed to better balance the dataset. At first, creating the dataset was a large problem, but the target image was changed to one of the classes in the CIFAR-10 dataset so that a larger, easily accessible dataset could be used.

Conclusion

From this project, we learned how to design a model for image recognition and classification and deploy it into the real world. We also learned about the importance of creating a model with enough complexity to produce a high accuracy, but not too much to cause a long training time and be able to fit on the memory constraint of the Arduino board. We also learned about troubleshooting the Arduino board and how to get the environment set up to deploy a fully trained model. We gained more familiarity with developing and saving an h5 file, converting it to a TFLite model, and deploying it to the Arduino board.

References

Person Detection Model:
https://github.com/mlcommons/tiny/blob/master/benchmark/training/visual_wake_words/vww_model.py