Facial Recognition on Android Using Google Cloud AutoML

Andrew Kelly
Google Developer Expert - Android
@andrew_ke11y

Table of contents

Prerequisites

To complete this workshop you'll need the following things.

1. Computer
2. Internet Connection
3. Android Studio (installed and able to run a sample project)
4. Your favourite Git client
5. A Firebase Project

Plan of attack

1. Checkout base Android app - make sure it runs.
2. Setup Firebase and Google Cloud project
3. Upload photos to Google Cloud AutoML
4. Add face detection to Android app
5. Train model in Google Cloud AutoML
6. Use Google Cloud AutoML REST or SDK
   from the Android app

Hopefully all this will take less than 90 mins, feel free to work at your own pace through this workshop, if you don't finish during the allotted time keep this sheet and finish at home.

I hope you enjoy this workshop and any feedback you have on the content is greatly appreciated.

Running the template Android App

We have a GitHub starter repository that's done some of the boring work for you.
- Requesting Permissions
- Basic Architecture

https://github.com/apkelly/DroidCon-Boston-2019

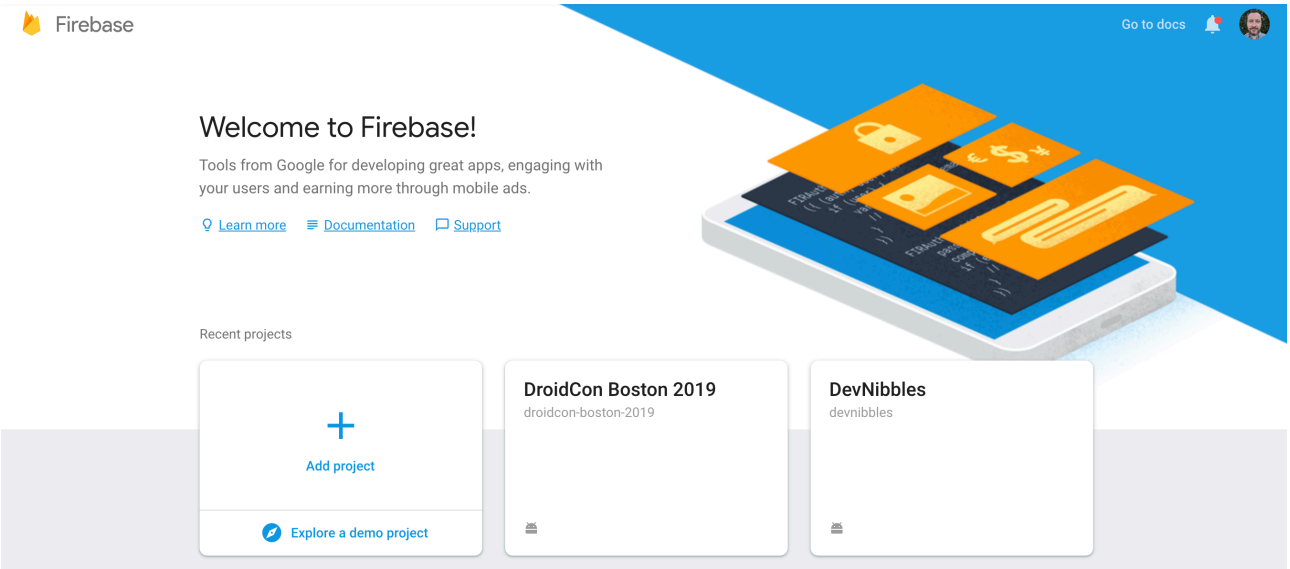
You'll want to start with the **checkpoint/1-start** branch

Let's open Android Studio and explore this starter project together.

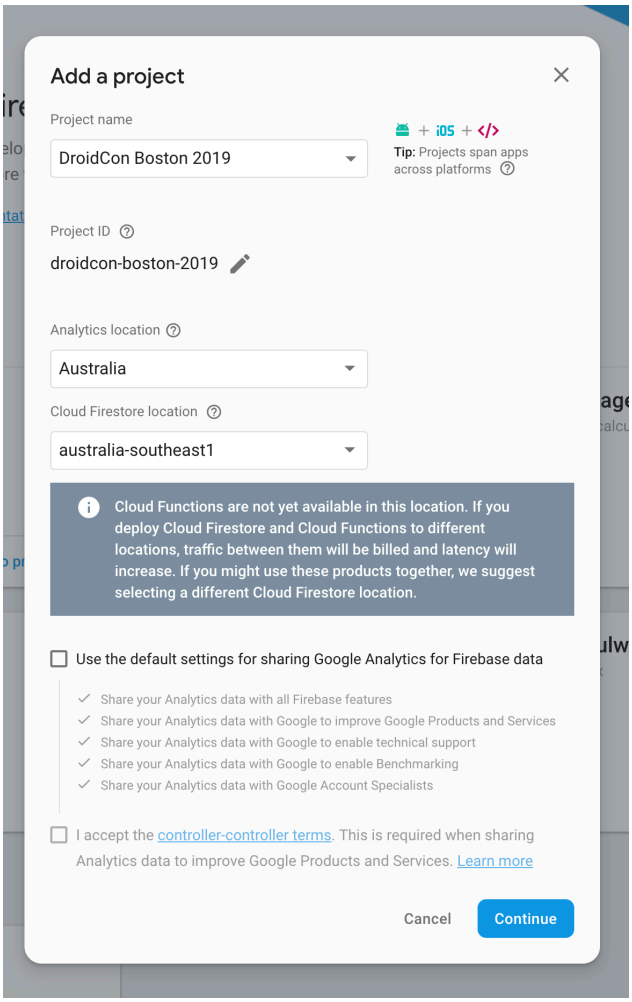
When setup successfully you should see a camera preview inside the app.

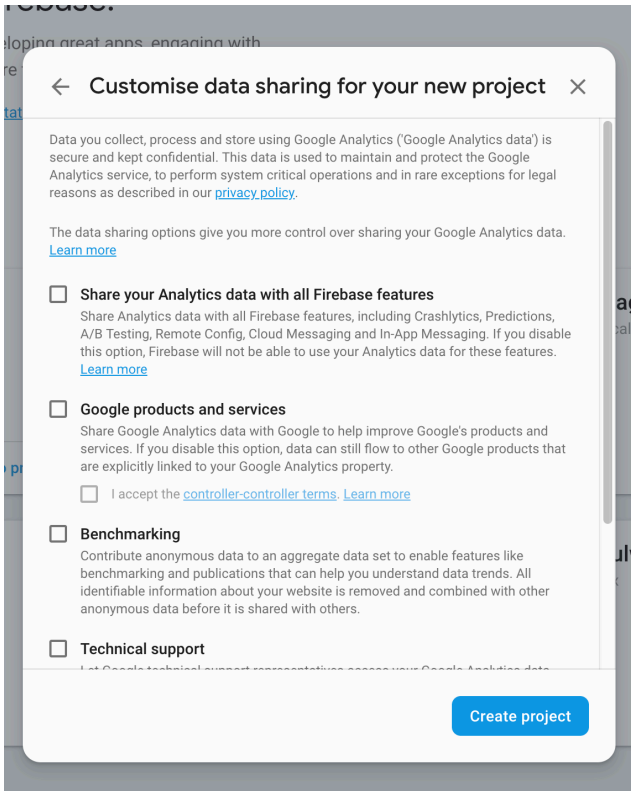
Setup Firebase

1. Open the Firebase Console in your browser - https://console.firebase.google.com/

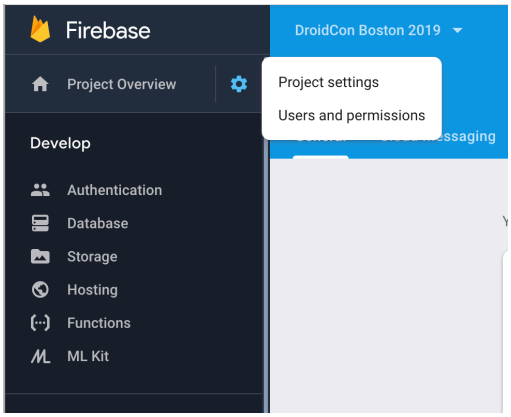


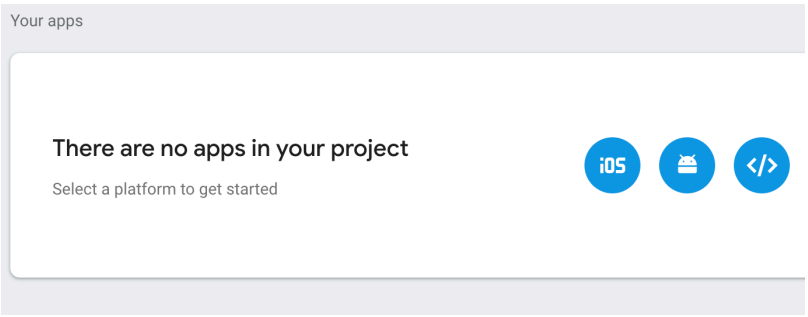
2. Add a new Project - I've called my DroidCon Boston 2019

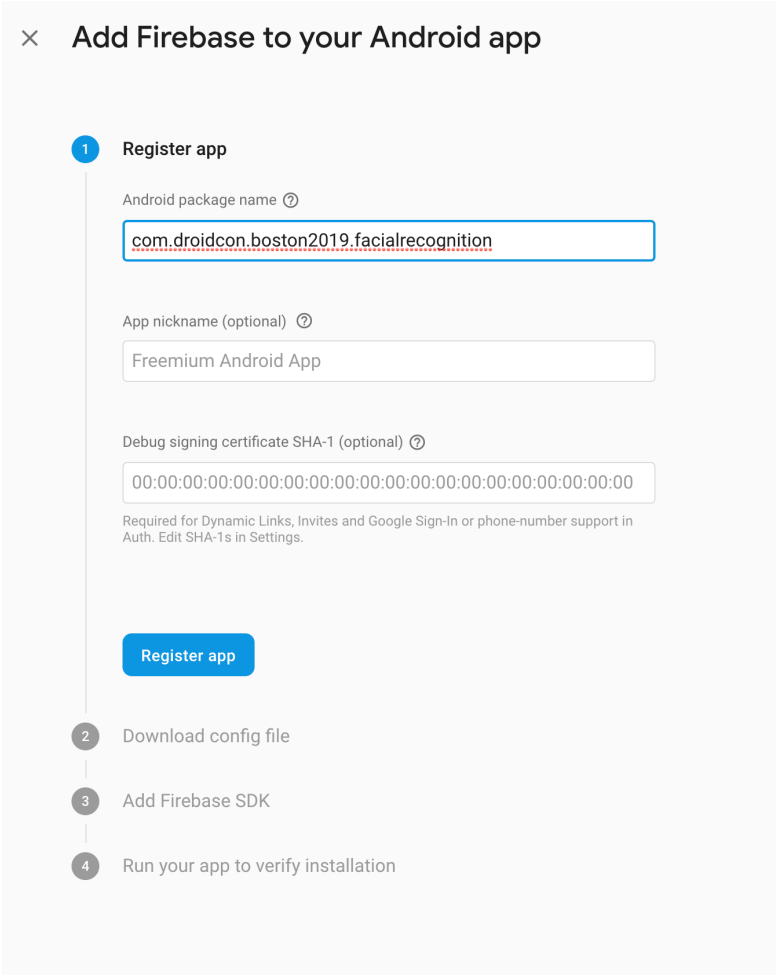




3. Add an Android app to your Firebase project







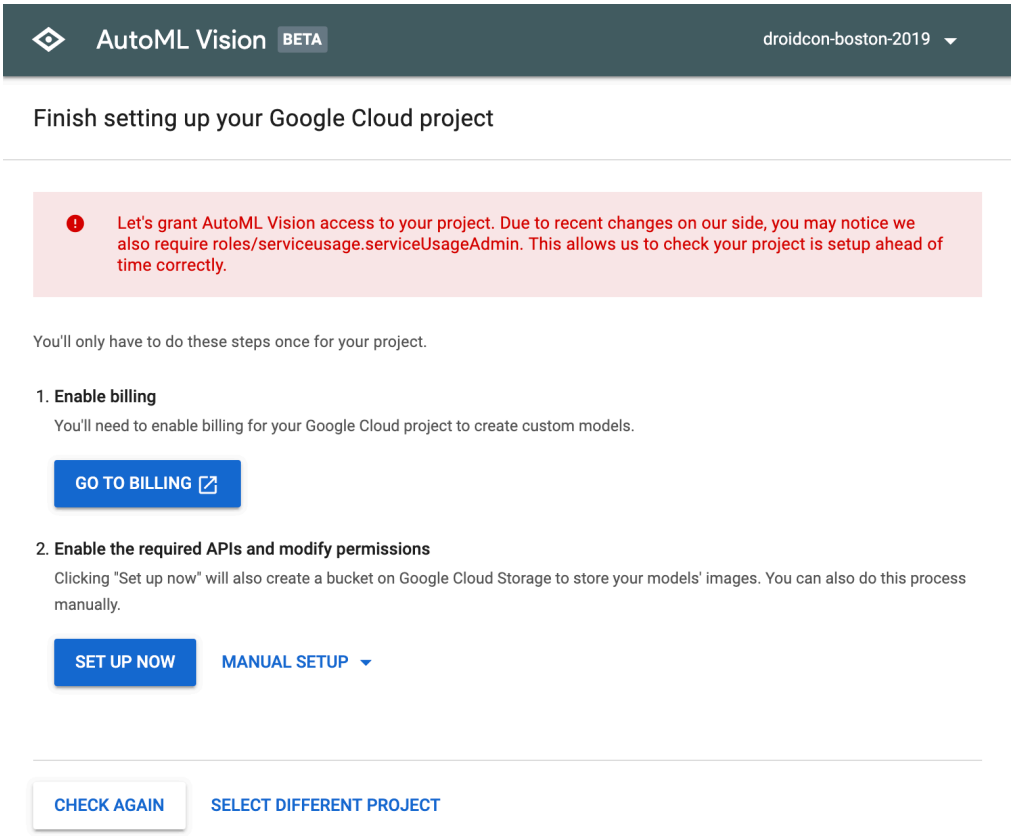
The package name you choose *must* match the Android project from earlier

com.droidcon.boston2019.facialrecognition

Remember to save the google-services.json config file into your android app project, you'll need this later!

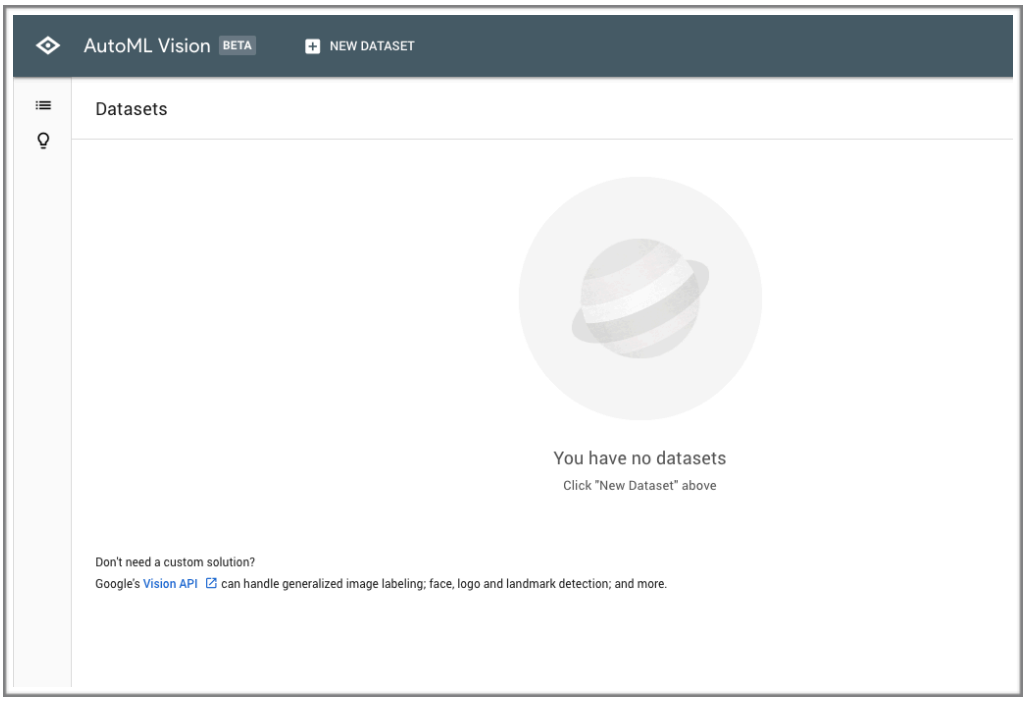
Setup Google Cloud

1. Open the Google Cloud site in your browser - https://cloud.google.com/automl/ui/vision/

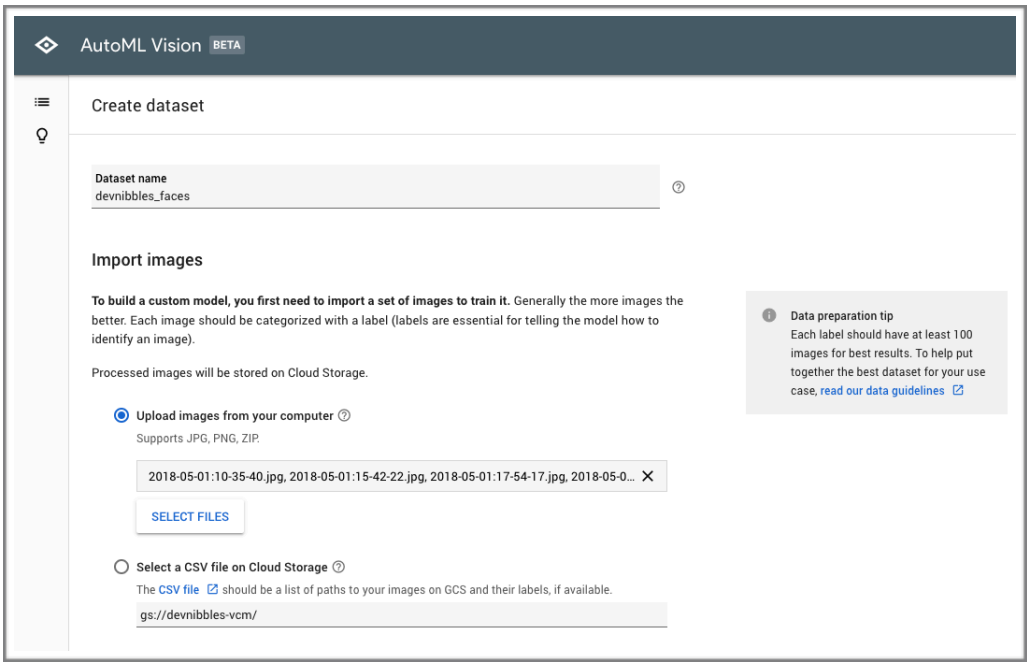


Google Cloud Image Upload

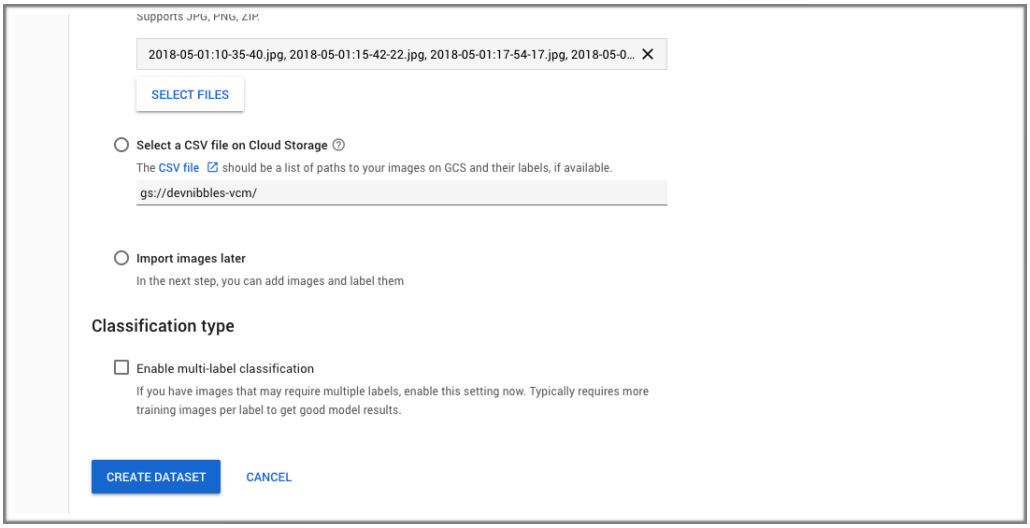
1. Choose a project from your list of firebase projects.



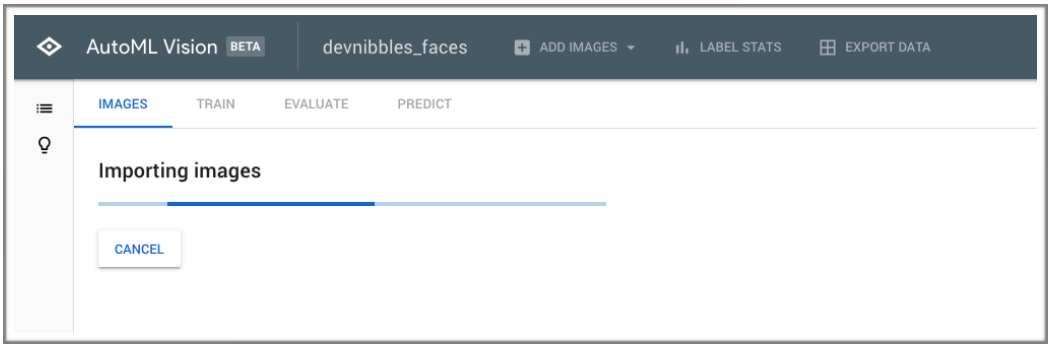
2. Create a New Dataset using the button at the top of the screen.



3. Give the dataset a meaningful name, from here we'll upload the images that make up our training data.
4. Select Files from your computer for upload.
5. Choose Create Dataset



The upload process can take a while depending on the number of images, and the speed of your internet connection. We'll be notified by email once the upload is complete, while we wait we'll start working on the Android App.



Detect Faces in the Android App

Switch back to Android Studio and open the MLActivity.kt file, add the following imports.

```
import com.droidcon.boston2019.facialrecognition.detect.mlkit.FaceDetector
import com.droidcon.boston2019.facialrecognition.detect.mlkit.FaceGraphic
import com.droidcon.boston2019.facialrecognition.detect.mlkit.FrameMetadata
import com.droidcon.boston2019.facialrecognition.detect.mlkit.MLCameraSource
import com.google.firebase.ml.vision.face.FirebaseVisionFace
import java.nio.ByteBuffer
```

You should see an empty createCameraSource function like this one

```
override fun createCameraSource() {
    mCameraSource = MLCameraSource(this, mGraphicOverlay)
}
```

And add the following code.

```
override fun createCameraSource() {
    mCameraSource = MLCameraSource(this, mGraphicOverlay)
    mCameraSource?.apply {
        setFrameDetector(
            FaceDetector(object : FaceDetector.DetectorCallback {
                override fun onSuccess(
                    frameData: ByteBuffer,
                    results: List<FirebaseVisionFace>,
                    frameMetadata: FrameMetadata) {

                    if (results.isEmpty()) {
                        // No faces in frame, so clear frame of any previous faces.
                        mGraphicOverlay.clear()
                    } else {
                        // We have faces
                        results.forEach { face ->
                            val existingFace = mGraphicOverlay
                                .find(face.trackingId) as FaceGraphic?

                            if (existingFace == null) {
                                // A new face has been detected.
                                val faceGraphic = FaceGraphic(
                                    face.trackingId,
                                    mGraphicOverlay
                                )

                                mGraphicOverlay.add(faceGraphic)

                            } else {
                                // We have an existing face, update its position.
                                existingFace.updateFace(face)
                            }
                        }

                        mGraphicOverlay.postInvalidate()

                    }
                }

                override fun onFailure(exception: Exception) {
                    exception.printStackTrace()
                }
            })
        )
    }
}
```

Next we'll need to configure the use of the MLKit Firebase library, you've already downloaded the google-services.json file.

Open the top level build.gradle file and add the following plugin

```
classpath 'com.google.gms:google-services:4.2.0'
```

Now open the app level build.gradle and add the following to the *bottom* of the file.

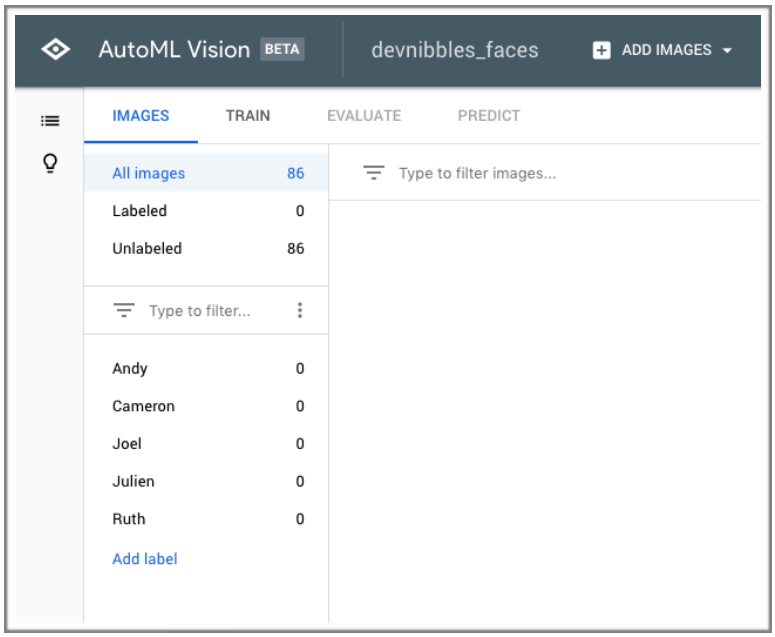
```
apply plugin: 'com.google.gms.google-services'
```

You should now be able to run the app again and see that your face is detected (some dots will be placed over your eyes).

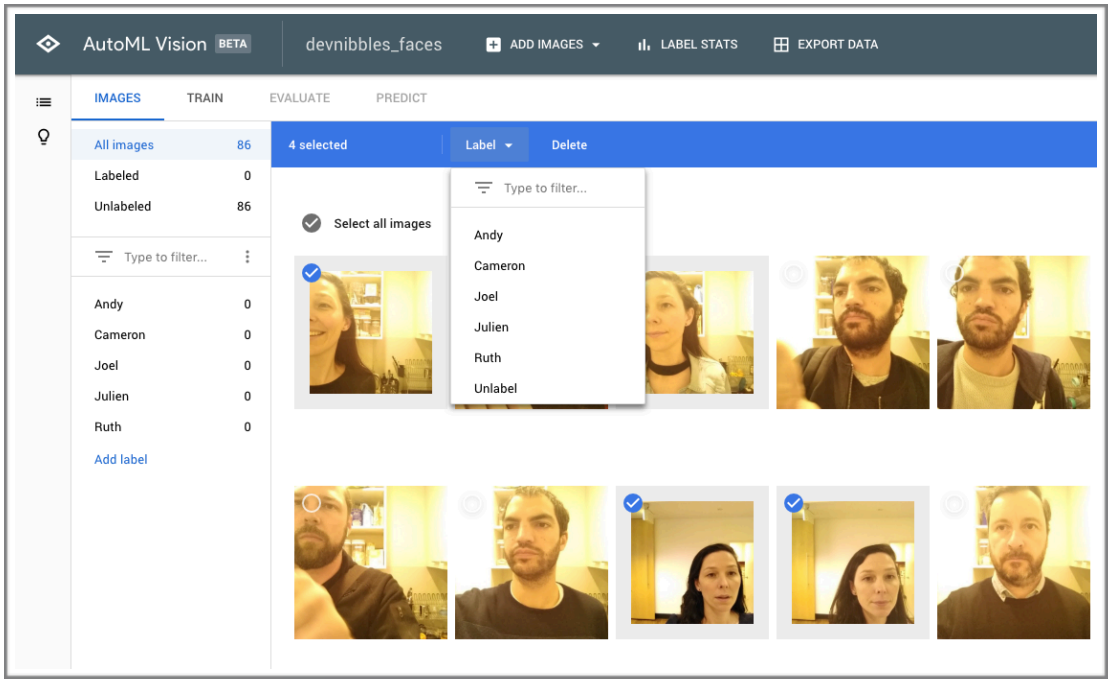
Google Cloud Training

Switch back to the browser, your image upload should now be complete.

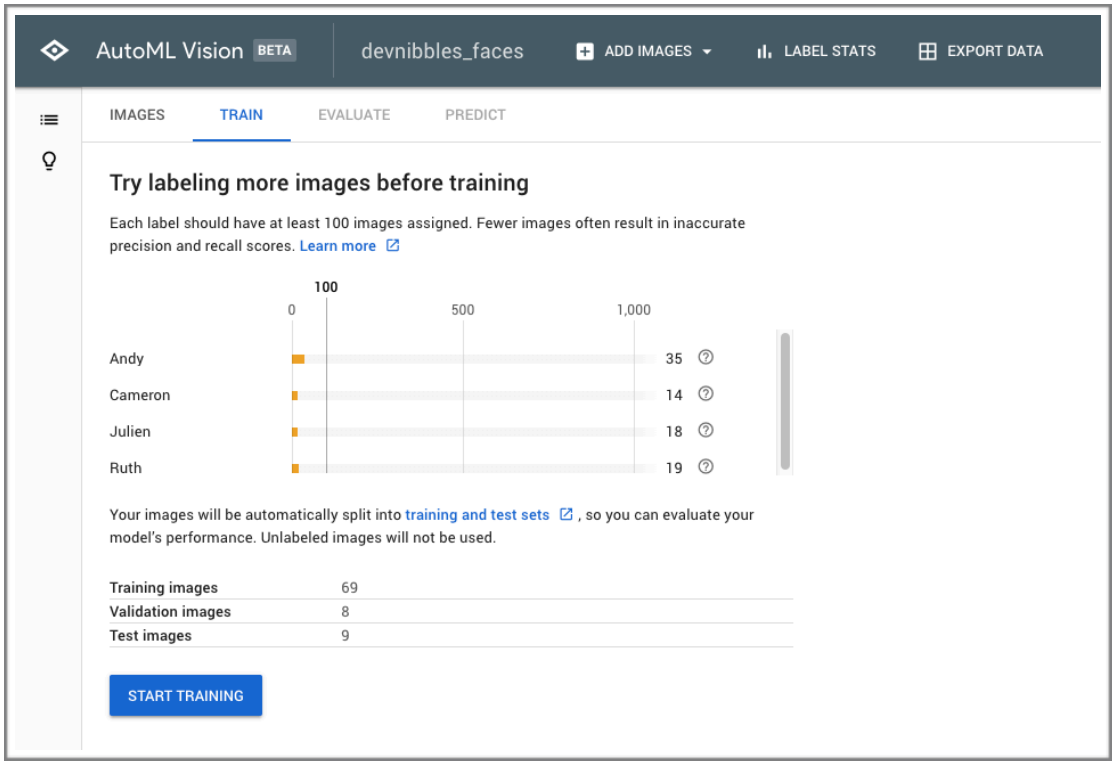
1. Add some labels for our images, these labels will be the names of the people.



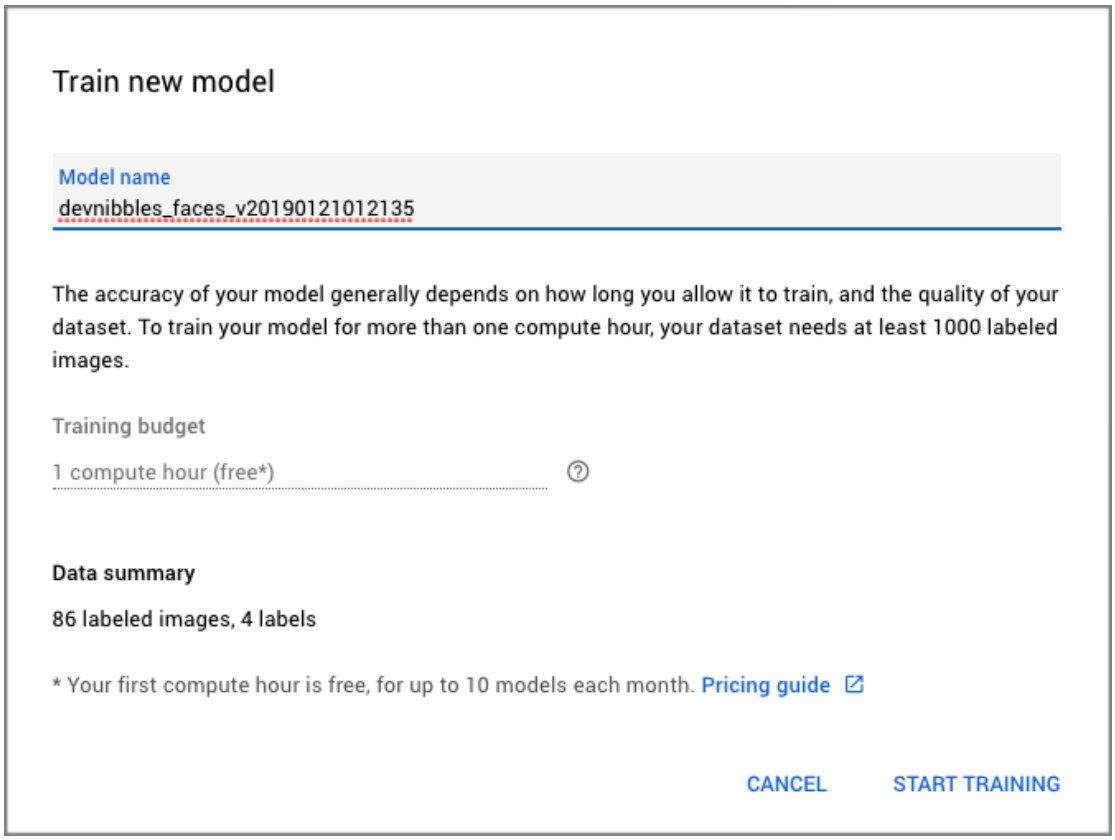
2. Select the photos of the people you want to label and choose the correct label.



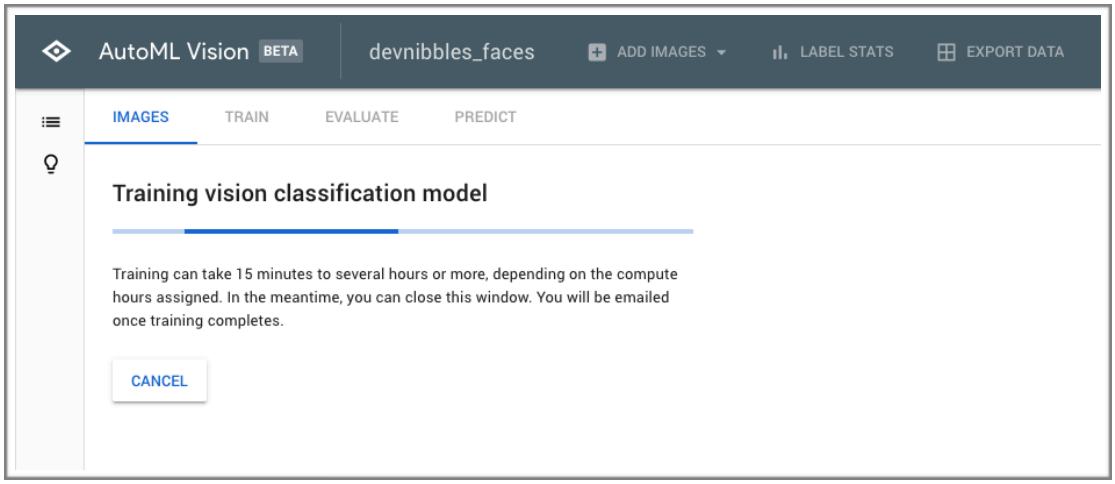
3. In the Train tab, select “Start Training”, you may be warned that there aren’t enough images, this is fine.



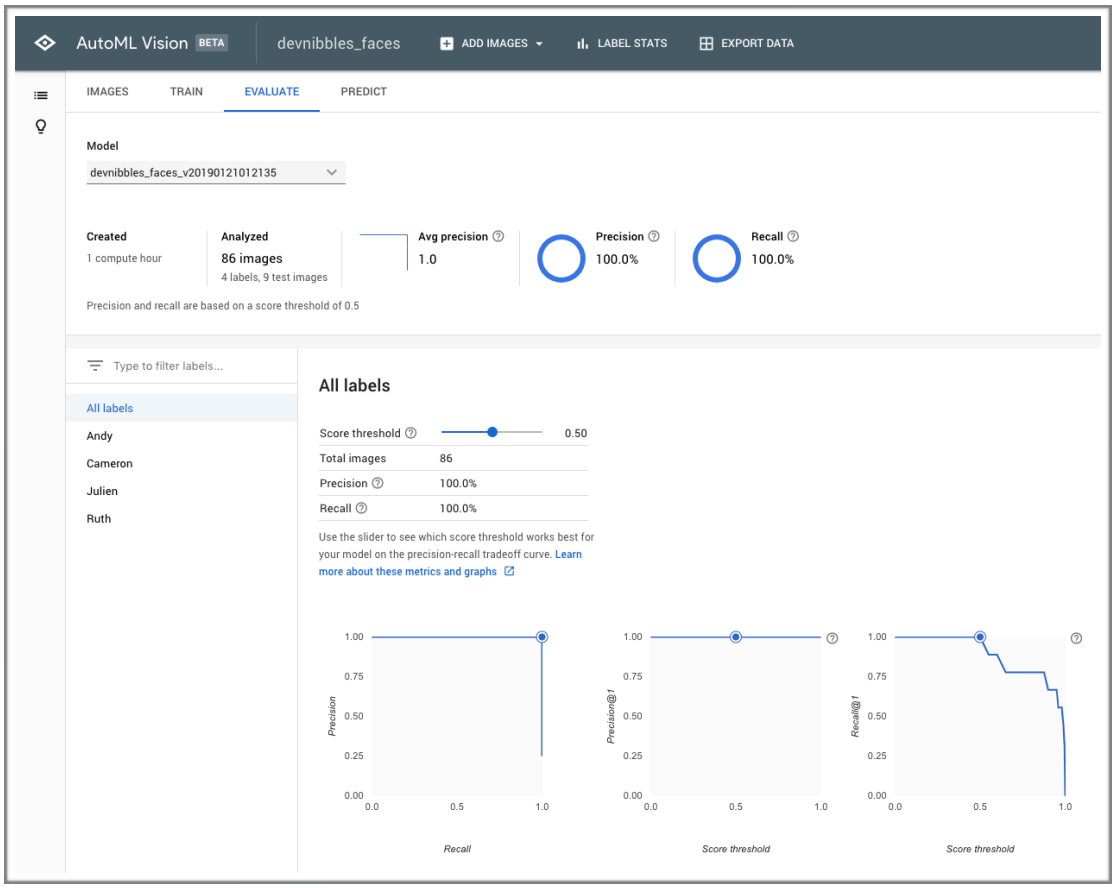
4. You’ll be prompted to create a Model Name, choosing the default is fine. You’ll note that we get 1hr of training time for free, this should be enough for our workshop, but if you’re going to train large amounts of data after the first hour is $20/hr.



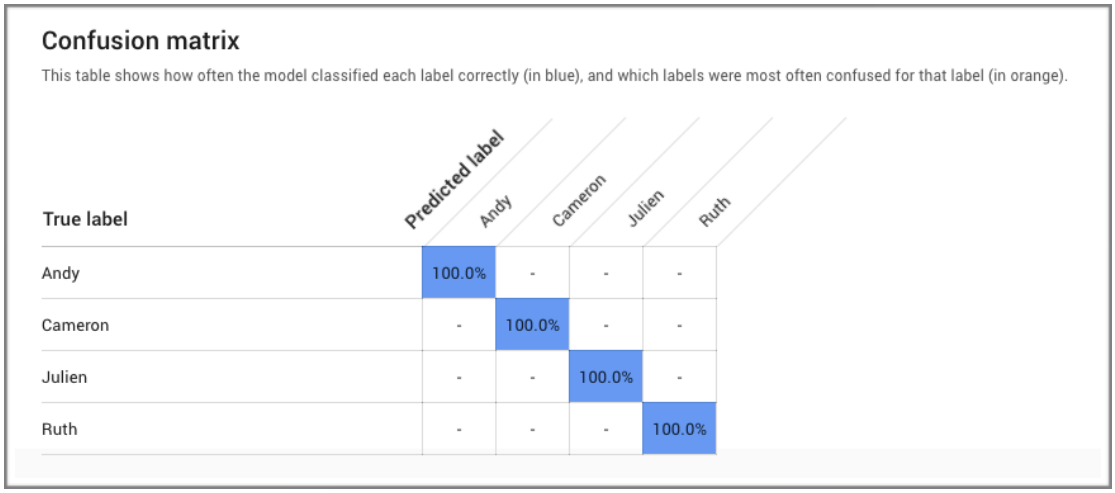
5. Training can take a while, don't worry we'll be notified by email once training is complete.



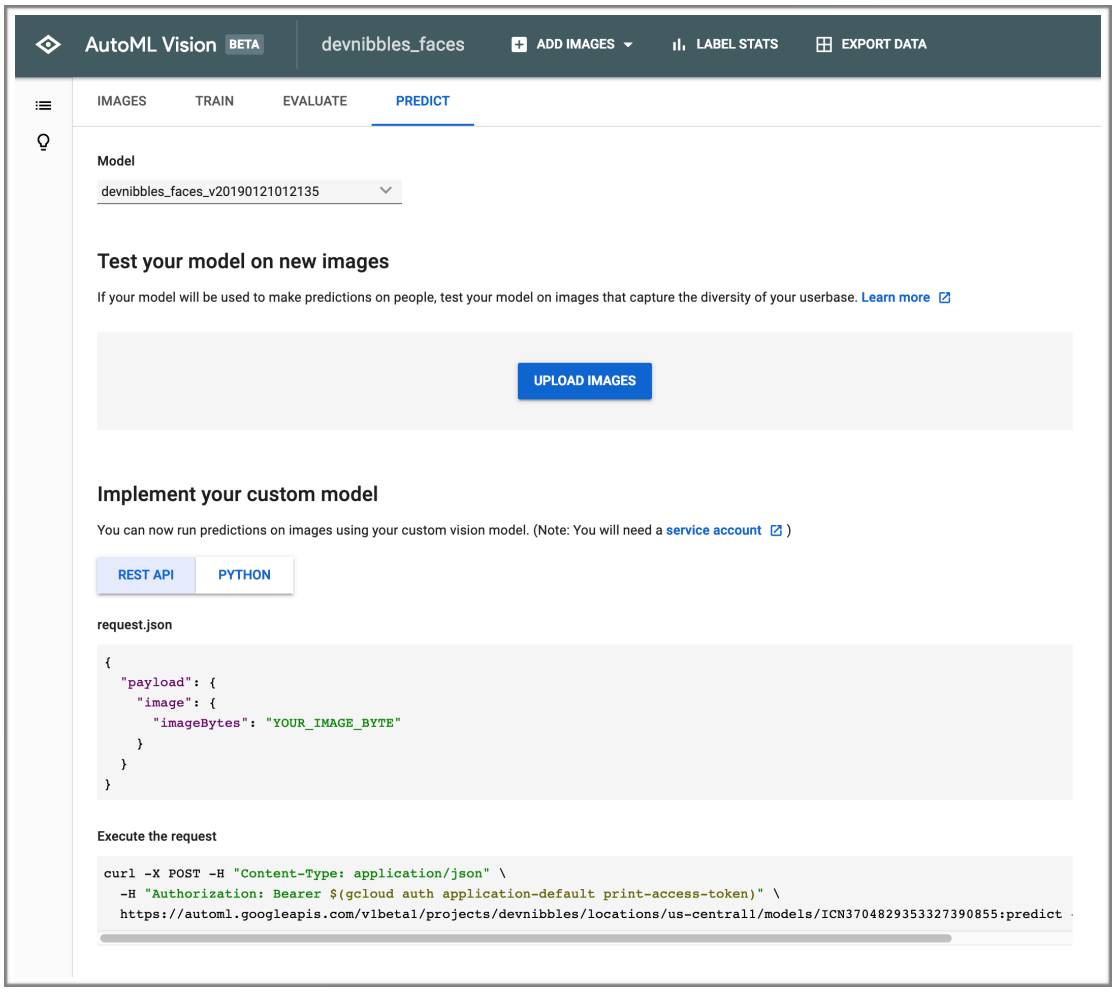
6. Once training is complete you'll be presented with an evaluation report, similar to this one.



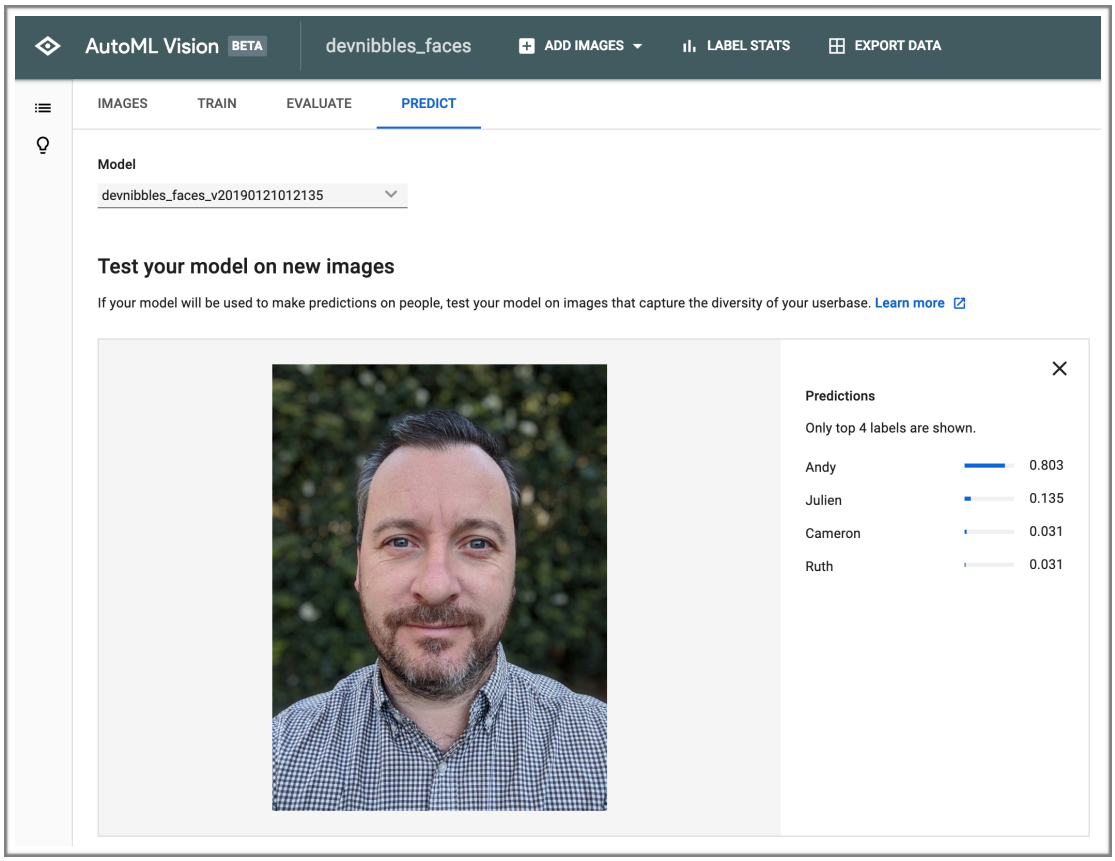
The confusion matrix shows how well the model classified the test images against the predictions, here we can see that if was 100% accurate at classifying the faces with the labels.



7. Finally, we can test the model through the web interface, so grab an image that the model hasn't seen before and let's try it out. It is this endpoint that our Android app will be calling later.



The results are in, this never before seen image was classified correctly with a confidence score of 80%



Classify Faces in Android App

Switch back to Android Studio and open MLActivity.kt again

Add the following to subscribe to our CloudAutoMLViewModel

```
private lateinit var mViewModel: CloudAutoMLViewModel

override fun onCreate(icicle: Bundle?) {
    super.onCreate(icicle)

    mViewModel = ViewModelProviders
        .of(this)
        .get(CloudAutoMLViewModel::class.java)

    mViewModel.subscribeClassifications()
        .observe(this, Observer<Resource<FaceClassification, Throwable>> { resource ->
            when (resource) {
                is LoadingResource -> {
                    // Show loading indicator for given face.
                    val faceId = resource.data?.faceId
                    faceId?.let {id ->
                        (mGraphicOverlay.find(id) as?
FaceGraphic)?.setName("Classifying...")
                    }
                }
                is SuccessResource -> {
                    val faceId = resource.data.faceId
                    val name = resource.data.name
                    val score = resource.data.confidence
                    (mGraphicOverlay.find(faceId) as? FaceGraphic)?.setName("$name (${"%.
2f".format(score)})")
                }
                is ErrorResource -> {
                    // Just print the stack trace, ideally show a dialog.
                    resource.errorData?.printStackTrace()
                }
            }
        })
}
```

We also need to call the classifier when we detect a face, so find the createCameraSource method that we added to earlier.

Find this bit of code, and add in the lines in bold to call our classify method.

```
if (existingFace == null) {
    // A new face has been detected.
    val faceGraphic = FaceGraphic(
        face.trackingId,
        mGraphicOverlay
    )

    mGraphicOverlay.add(faceGraphic)

    // Lets try and find out who this face belongs to
    mViewModel.classify(face.trackingId, frameData.convertToByteArray(frameMetadata))
} else {
    // We have an existing face, update its position.
    existingFace.updateFace(face)
}
```

Finally switch over to the CloudAutoMLViewModel.kt file and add the replace the empty classifier methods with these ones.

```
private fun classifyUsingRetrofit(faceId: Int, imageBytes: ByteArray) {
    launch(errorHandler) {
        // Show loading indicator while we wait for the request.
        mResult.value = LoadingResource(FaceClassification(faceId, "", 0.0))

        // Build the body of our request, essentially the image to be classified.
        val body = CloudAutoMLModel(
            Payload(
                MlImage(
                    String(
                        Base64.encodeBase64(imageBytes)
                    )
                )
            )
        )

        // Define the authentication credentials and make the API request
        val response = getRESTService().classify(
            "Bearer ${accessToken?.tokenValue}",
            PROJECT, LOCATION, MODEL, body
        ).await()

        if (response.payload?.isNotEmpty() == true) {
            // We have a prediction!
            var predictedName: String? = null
            var predictedConfidence: Double? = null

            response.payload.forEach { entry ->
                if (entry.displayName != null) {
                    predictedName = entry.displayName
                    predictedConfidence = entry.classification?.score
                }
            }

            if (predictedName != null && predictedConfidence != null) {
                // We had an actual name returned
                mResult.postValue(
                    SuccessResource(
                        FaceClassification(
                            faceId,
                            predictedName!!,
                            predictedConfidence!!
                        )
                    )
                )
            } else {
                // No name was returned, this is an unknown face.
                mResult.postValue(ErrorResource(null))
            }
        } else {
            // There were no payloads returned, possible error or unknown face.
            mResult.postValue(ErrorResource(null))
        }
    }
}
```

```
private fun classifyUsingCloudSDK(faceId: Int, imageBytes: ByteArray) {
    launch(errorHandler) {
        // Show loading indicator while we wait for the request.
        mResult.value = LoadingResource(FaceClassification(faceId, "", 0.0))

        withContext(Dispatchers.IO) {
            // Define the authentication credentials
            val settings = PredictionServiceSettings.newBuilder()
                .setCredentialsProvider(FixedCredentialsProvider.create(mServiceCredentia
ls)).build()

            val predictionServiceClient = PredictionServiceClient.create(settings)
            predictionServiceClient.use { client ->
                // Build the body of our request, essentially the image to be classified.
                val name = ModelName.of(PROJECT, LOCATION, MODEL)
                val image =
Image.newBuilder().setImageBytes(ByteString.copyFrom(imageBytes)).build()
                val payload = ExamplePayload.newBuilder().setImage(image).build()
                val params = HashMap<String, String>()

                // Make the API request.
                val response = client.predict(name, payload, params)

                if (response.payloadCount > 0) {
                    // We have a prediction!
                    var predictedName: String? = null
                    var predictedConfidence: Double? = null

                    response.getPayload(0).allFields.entries.forEach { entry ->

                        if (entry.key.jsonName == "displayName") {
                            predictedName = entry.value as String
                        } else if (entry.key.jsonName == "classification") {
                            val classification = entry.value as
ClassificationProto.ClassificationAnnotation
                            predictedConfidence= classification.score.toDouble()
                        }
                    }

                    if (predictedName != null && predictedConfidence != null) {
                        // We had an actual name returned
                        mResult.postValue(
                            SuccessResource(
                                FaceClassification(
                                    faceId,
                                    predictedName!!,
                                    predictedConfidence!!
                                )
                            )
                        )
                    } else {
                        // No name was returned, this is an unknown face.
                        mResult.postValue(ErrorResource(null))
                    }
                } else {
                    // There were no payloads returned, possible error or unknown face.
                    mResult.postValue(ErrorResource(null))
                }
            }
        }
    }
}
```

Then once you have all your code in place you lastly need to configure the endpoint for your Firebase project.

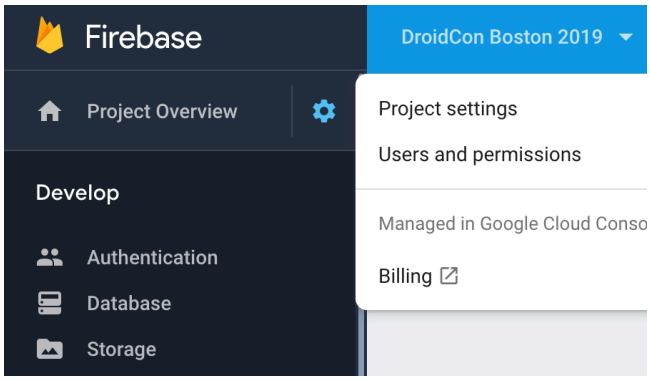
```
private const val PROJECT = "droidcon-boston-2019"
private const val LOCATION = "us-central1"
private const val MODEL = "ICN1636466924499999978"
private const val SERVICE_ACCOUNT_JSON = "{\n" +
        "  \"type\": \"service_account\",\n" +
        "  \"project_id\": \"\",\n" +
        "  \"private_key_id\": \"\",\n" +
        "  \"private_key\": \"\",\n" +
        "  \"client_email\": \"\",\n" +
        "  \"client_id\": \"\",\n" +
        "  \"auth_uri\": \"https://accounts.google.com/o/oauth2/auth\",\n" +
        "  \"token_uri\": \"https://oauth2.googleapis.com/token\",\n" +
        "  \"auth_provider_x509_cert_url\": \"\",\n" +
        "  \"client_x509_cert_url\": \"\"\n" +
        "}\n"
```

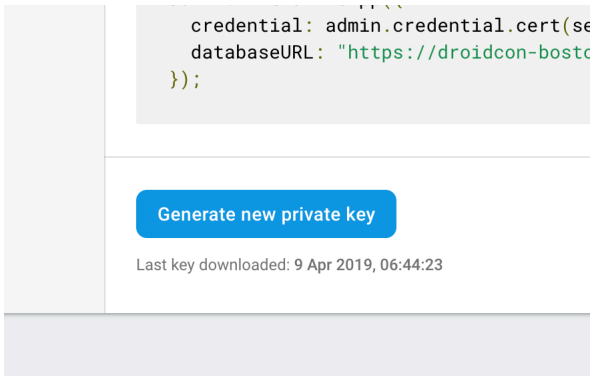
These values should match the ones you entered earlier when you setup your Firebase project, and when you trained your model.

You can switch back to the browser to fetch these values from the Predict tab of the Google Cloud AutoML website. Look for the CURL command at the bottom of the web page, and extract the bits marked in bold.

curl -X POST -H "Content-Type: application/json" \
  -H "Authorization: Bearer $(gcloud auth application-default print-access-token)" \
  https://automl.googleapis.com/v1beta1/projects/**droidcon-boston-2019**/locations/**us-central1**/models/**ICN1636466924481055678**:predict -d @request.json

The SERVICE_ACCOUNT_JSON is the details of the Service Account you need to use your service securely. Switch back into your Firebase project, and go to Settings, from there you'll want to Generate a new private key. Just copy the contents of the generated file into the String.





Appendix

If you want to generate large amounts of image data quickly take a video of your friends, then convert that video to a series of jpgs using FFMPEG

The command I used over a 10 second video was

```
./ffmpeg -i andy.mp4 -vf "scale=640:-1,fps=24" andy_%d.jpg
```

Where ruth.mp4 was my input video, and I wanted to capture 24 images for each second of footage (fps). If you have longer videos, then you may want to reduce this to fps=12 or fps=8 to keep the number of images manageable. Finally we specify a filename template for the resulting images.