



Facial Recognition on Android Using Google Cloud AutoML

Andrew Kelly
Google Developer Expert - Android
[@andrew_ke11y](https://twitter.com/@andrew_ke11y)

Table of contents

Prerequisites	3
Plan of attack	3
Running the template Android App	4
Setup Firebase	5
Setup Google Cloud	7
Google Cloud Image Upload	8
Detect Faces in the Android App	10
Google Cloud Training	12
Classify Faces in Android App	17
Appendix	21

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.firebaseio.google.com/>

Welcome to Firebase!

Tools from Google for developing great apps, engaging with your users and earning more through mobile ads.

Recent projects

- + Add project
- DroidCon Boston 2019 droidcon-boston-2019
- DevNibbles devnibbles

Learn more Documentation Support

Go to docs

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

Add a project

Project name: DroidCon Boston 2019

Tip: Projects span apps across platforms

Project ID: droidcon-boston-2019

Analytics location: Australia

Cloud Firestore location: australia-southeast1

Cloud Functions are not yet available in this location. If you deploy Cloud Firestore and Cloud Functions to different locations, traffic between them will be billed and latency will increase. If you might use these products together, we suggest selecting a different Cloud Firestore location.

Use the default settings for sharing Google Analytics for Firebase data

- ✓ Share your Analytics data with all Firebase features
- ✓ Share your Analytics data with Google to improve Google Products and Services
- ✓ Share your Analytics data with Google to enable technical support
- ✓ Share your Analytics data with Google to enable Benchmarking
- ✓ Share your Analytics data with Google Account Specialists

I accept the [controller-controller terms](#). This is required when sharing Analytics data to improve Google Products and Services. [Learn more](#)

Customise data sharing for your new project

Data you collect, process and store using Google Analytics ('Google Analytics data') is secure and kept confidential. This data is used to maintain and protect the Google Analytics service, to perform system critical operations and in rare exceptions for legal reasons as described in our [privacy policy](#).

The data sharing options give you more control over sharing your Google Analytics data. [Learn more](#)

Share your Analytics data with all Firebase features

Share Analytics data with all Firebase features, including Crashlytics, Predictions, A/B Testing, Remote Config, Cloud Messaging and In-App Messaging. If you disable this option, Firebase will not be able to use your Analytics data for these features. [Learn more](#)

Google products and services

Share Google Analytics data with Google to help improve Google's products and services. If you disable this option, data can still flow to other Google products that are explicitly linked to your Google Analytics property.

I accept the [controller-controller terms](#). [Learn more](#)

Benchmarking

Contribute anonymous data to an aggregate data set to enable features like benchmarking and publications that can help you understand data trends. All identifiable information about your website is removed and combined with other anonymous data before it is shared with others.

Technical support

Get Google technical support and notifications across your Google Analytics data.

Create project

3. Add an Android app to your Firebase project

The screenshot shows the Firebase console interface. The left sidebar has sections for Project Overview, Develop (Authentication, Database, Storage, Hosting, Functions, ML Kit), and messaging. A dropdown menu from the messaging section is open, showing Project settings and Users and permissions. The main area is titled 'Your apps' and displays the message 'There are no apps in your project'. It includes a 'Select a platform to get started' button and icons for iOS, Android, and web development.

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/>

The screenshot shows the 'AutoML Vision' setup page. At the top, there's a dark header bar with the 'AutoML Vision' logo and a 'BETA' badge. To the right is a dropdown menu labeled 'droidcon-boston-2019'. Below the header, a main title says 'Finish setting up your Google Cloud project'. A red callout box contains a warning message: 'Let's grant AutoML Vision access to your project. Due to recent changes on our side, you may notice we also require roles/serviceusage.serviceUsageAdmin. This allows us to check your project is setup ahead of time correctly.' Below the warning, a note states 'You'll only have to do these steps once for your project.' There are two numbered steps: '1. Enable billing' and '2. Enable the required APIs and modify permissions'. Step 1 has a blue button labeled 'GO TO BILLING'. Step 2 has two options: 'SET UP NOW' (blue button) and 'MANUAL SETUP ▾'. At the bottom, there are two links: 'CHECK AGAIN' and 'SELECT DIFFERENT PROJECT'.

Finish setting up your Google Cloud project

1 Let's grant AutoML Vision access to your project. Due to recent changes on our side, you may notice we also require roles/serviceusage.serviceUsageAdmin. This allows us to check your project is setup ahead of time correctly.

You'll only have to do these steps once for your project.

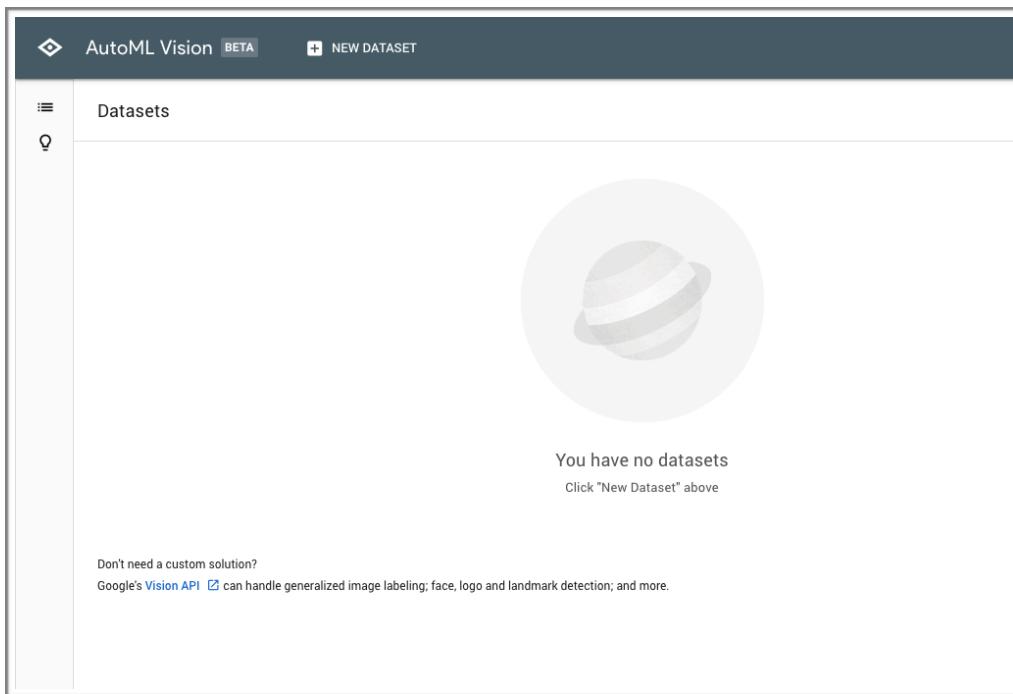
1. **Enable billing**
You'll need to enable billing for your Google Cloud project to create custom models.
[GO TO BILLING](#)

2. **Enable the required APIs and modify permissions**
Clicking "Set up now" will also create a bucket on Google Cloud Storage to store your models' images. You can also do this process manually.
[SET UP NOW](#) [MANUAL SETUP ▾](#)

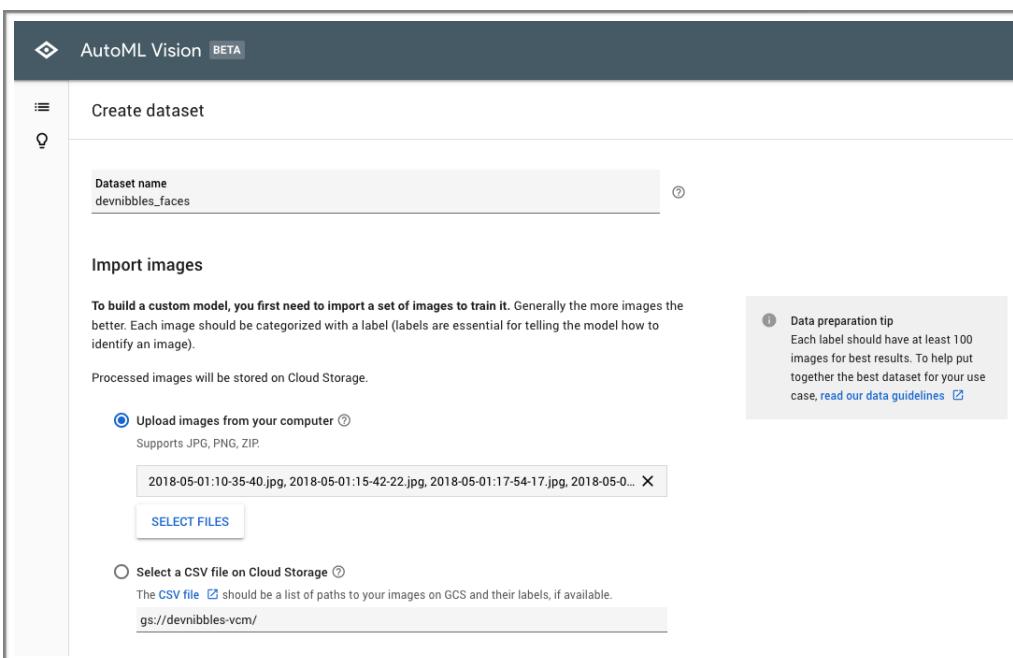
[CHECK AGAIN](#) [SELECT DIFFERENT PROJECT](#)

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

Supports JPG, PNG, ZIP.

2018-05-01:10-35-40.jpg, 2018-05-01:15-42-22.jpg, 2018-05-01:17-54-17.jpg, 2018-05-0... X

[SELECT FILES](#)

Select a CSV file on Cloud Storage ⓘ
The CSV file should be a list of paths to your images on GCS and their labels, if available.
gs://devnibbles-vcm/

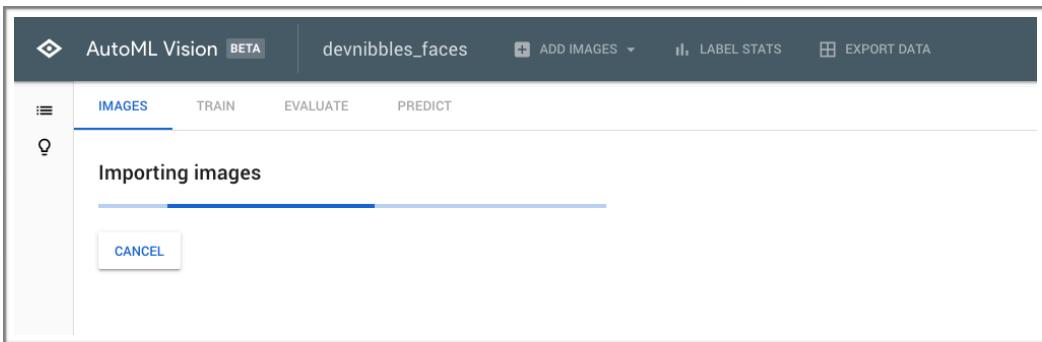
Import images later
In the next step, you can add images and label them

Classification type

Enable multi-label classification
If you have images that may require multiple labels, enable this setting now. Typically requires more training images per label to get good model results.

[CREATE DATASET](#) [CANCEL](#)

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()
                    }
                }
            })
        )
    }
}
```

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.

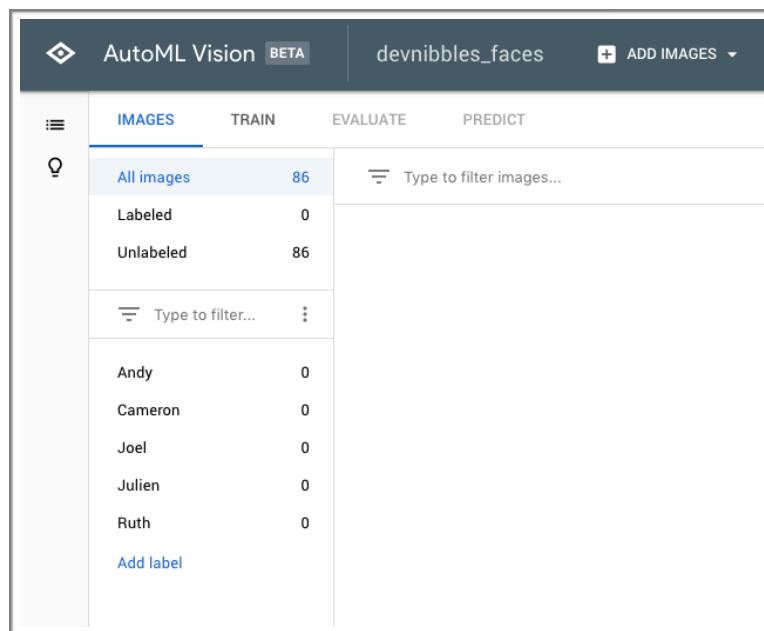
```
apply plugin: 'com.google.gms.google-services'
```

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.

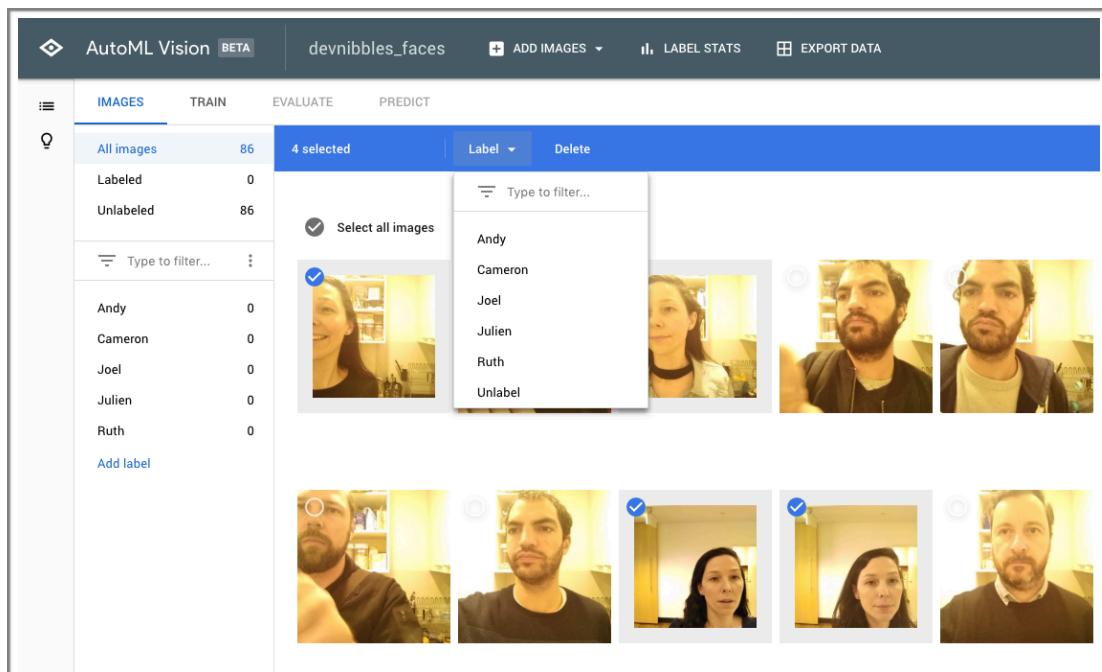
1. Add some labels for our images, these labels will be the names of the people.



AutoML Vision BETA devnibbles_faces + ADD IMAGES ▾

	IMAGES	TRAIN	EVALUATE	PREDICT
All images	86			Type to filter images...
Labeled	0			
Unlabeled	86			
Type to filter...		⋮		
Andy	0			
Cameron	0			
Joel	0			
Julien	0			
Ruth	0			
Add label				

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



AutoML Vision BETA devnibbles_faces + ADD IMAGES ▾ ⚡ LABEL STATS ⚡ EXPORT DATA

IMAGES TRAIN EVALUATE PREDICT

	IMAGES	TRAIN	EVALUATE	PREDICT
All images	86			
Labeled	0			
Unlabeled	86			
Type to filter...		⋮		
Andy	0			
Cameron	0			
Joel	0			
Julien	0			
Ruth	0			
Add label				

4 selected Label Delete

Select all images Type to filter...

Andy Cameron Joel Julien Ruth Unlabel

Andy Cameron Joel Julien Ruth Unlabel

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

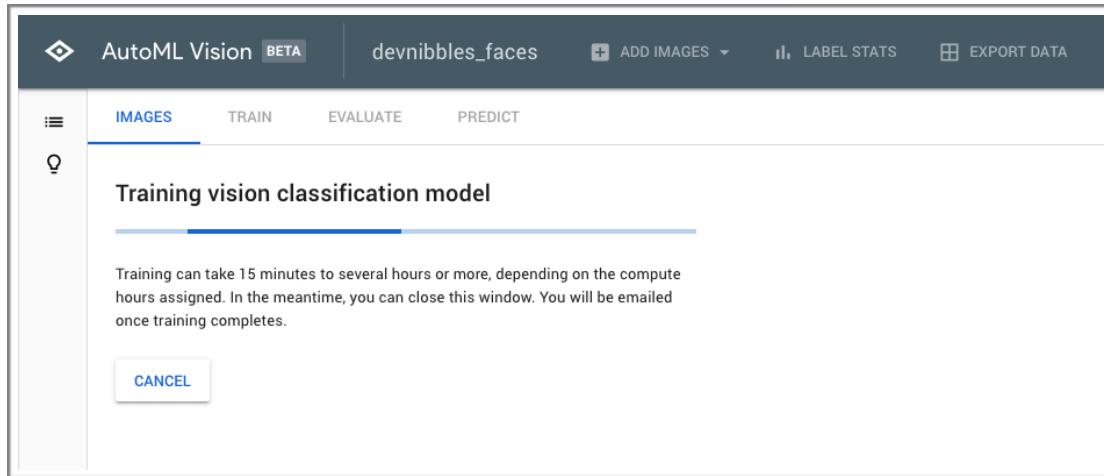
The screenshot shows the AutoML Vision interface with the project name "devnibbles_faces". The "TRAIN" tab is selected. A message at the top says "Try labeling more images before training" and notes that each label should have at least 100 images assigned. Below this is a chart showing labeled images for four people: Andy (35), Cameron (14), Julien (18), and Ruth (19). A note below the chart states that images will be split into training and test sets. At the bottom, a table shows the distribution of images: Training images (69), Validation images (8), and Test images (9). A large blue "START TRAINING" button is at the bottom.

	Training images	Validation images	Test images
Andy	35		
Cameron	14		
Julien	18		
Ruth	19		

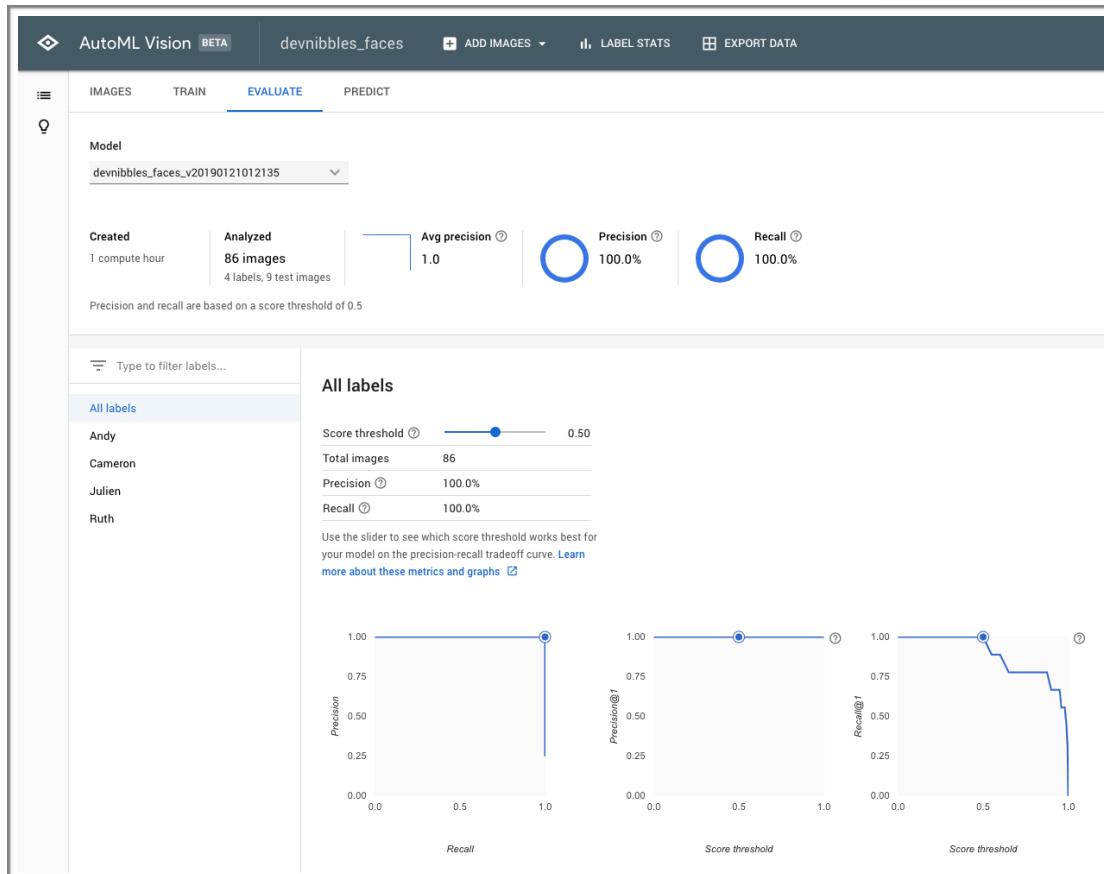
4. You’ll be prompted to create a Model Name, choosing the default is fine. You’ll note that we get 1hr of training time for free, this should be enough for our workshop, but if you’re going to train large amounts of data after the first hour is \$20/hr.

The dialog is titled "Train new model". It shows a "Model name" field containing "devnibbles_faces_v20190121012135". Below it is a note about training accuracy depending on dataset quality and size. The "Training budget" section shows "1 compute hour (free*)". The "Data summary" section shows "86 labeled images, 4 labels". At the bottom, a note states "* Your first compute hour is free, for up to 10 models each month." and includes a "Pricing guide" link. The dialog has "CANCEL" and "START TRAINING" buttons at the bottom.

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 it was 100% accurate at classifying the faces with the labels.

Confusion matrix					
True label	Predicted label	Andy	Cameron	Julien	Ruth
		Andy	Cameron	Julien	Ruth
Andy	Andy	100.0%	-	-	-
Cameron	Cameron	-	100.0%	-	-
Julien	Julien	-	-	100.0%	-
Ruth	Ruth	-	-	-	100.0%

- 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 screenshot shows the AutoML Vision web interface. At the top, there is a navigation bar with the title "AutoML Vision BETA", a "devnibbles_faces" project name, and buttons for "ADD IMAGES", "LABEL STATS", and "EXPORT DATA". Below the navigation bar, there is a sidebar with "IMAGES", "TRAIN", "EVALUATE", and "PREDICT" buttons, with "PREDICT" being the active tab. In the main area, there is a section titled "Model" with a dropdown menu set to "devnibbles_faces_v20190121012135". Below this, there is a section titled "Test your model on new images" with a note about testing on diverse images and a "UPLOAD IMAGES" button. Further down, there is a section titled "Implement your custom model" with instructions and code examples for "REST API" and "PYTHON". The Python example shows a "request.json" file template and a curl command for executing the request.

```
request.json
{
  "payload": {
    "image": {
      "imageBytes": "YOUR_IMAGE_BYTE"
    }
  }
}

Execute the request
curl -X POST -H "Content-Type: application/json" \
-H "Authorization: Bearer $(gcloud auth application-default print-access-token)" \
https://automl.googleapis.com/v1beta1/projects/devnibbles/locations/us-central1/models/ICN3704829353327390855:predict
```

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

AutoML Vision BETA

devnibbles_faces + ADD IMAGES LABEL STATS EXPORT DATA

IMAGES TRAIN EVALUATE PREDICT

Model
devnibbles_faces_v20190121012135

Test your model on new images
If your model will be used to make predictions on people, test your model on images that capture the diversity of your userbase. [Learn more](#)



Predictions
Only top 4 labels are shown.

Andy	0.803
Julien	0.135
Cameron	0.031
Ruth	0.031

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(mServiceCredentials)).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 = "ICN163646692449999978"
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.

The screenshot shows two parts of the Firebase console. On the left, the 'Project Overview' page for 'DroidCon Boston 2019' is shown with links for 'Project settings', 'Users and permissions', 'Managed in Google Cloud Conso', and 'Billing'. On the right, the 'Project settings' page is open, showing the 'Service account' section. It displays a JSON string for the service account, with the 'private_key' field highlighted in yellow. Below the JSON is a blue button labeled 'Generate new private key' and a note indicating the last key was downloaded on April 9, 2019, at 06:44:23.

```
credential: admin.credential.cert({
  databaseURL: "https://droidcon-boston.firebaseio.com"
});
```

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.