Activity 1 – Hands on creating a QnA chatbot

In this activity, we will learn:

- ❑ Create DialogFlow Knowledge Base.
- ❑ Create an Integration to use a knowledge base
- ❑ Chat with the bot to verify the code is working
- ❑ Link Telegram to bot's channel

1. Prerequisites

a) To start using Dialogflow Knowledge, you will need to have a Google account. Head over to here (https://accounts.google.com/signup/v2) to create one if necessary.

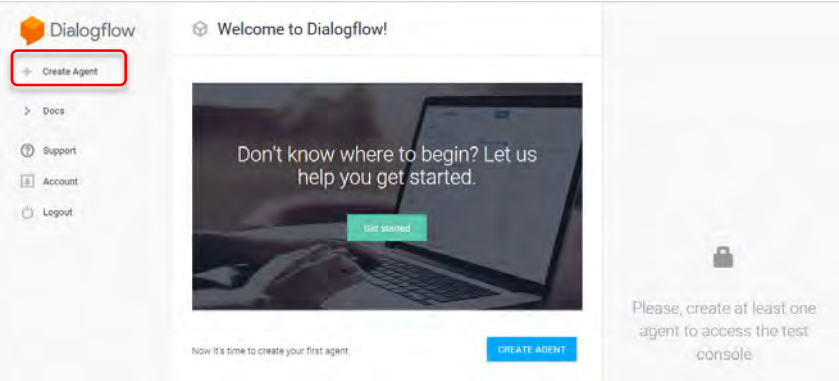
2) Create an Agent

a) Login in to Dialogflow (https://console.dialogflow.com/api-client/#/login).

b) If it is the first time you login to Dialogflow with this account, you will be asked to allow Dialogflow to access your Google account. Click on **Allow** to continue.

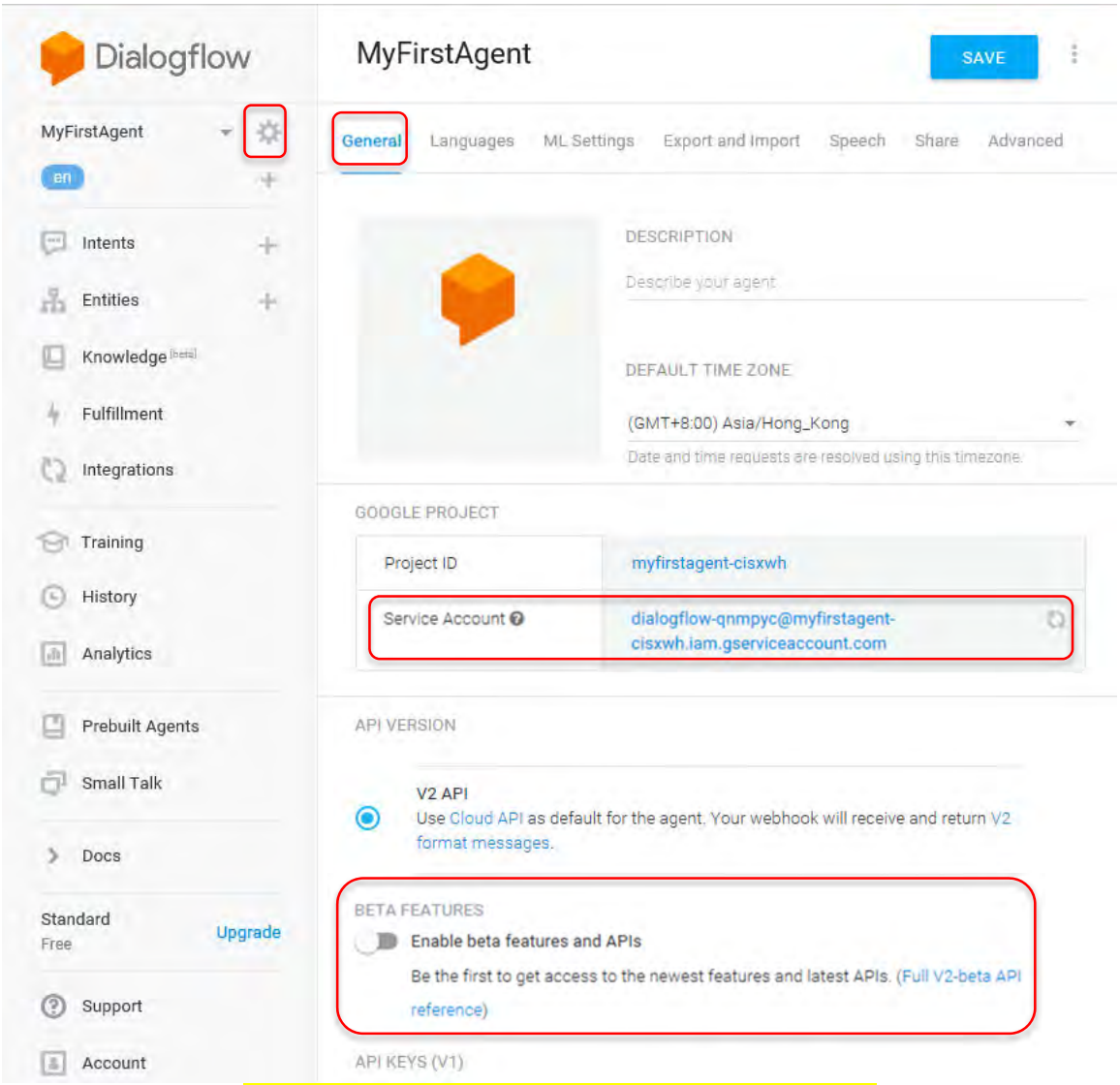
c) In the next screen, review your account settings. The only required action is to check "*Yes, I have read and accept the agreement*". Click **ACCEPT** to continue.

d) You may be asked to authenticate again. If everything is ok, you should see the following screen. Click on **Create Agent**



e) Give your agent an appropriate name. We can leave the DEFAULT LANGUAGE and DEFAULT TIME ZONE settings. Click **CREATE** to continue. After a while, the screen will refresh and bring you to details configuration page.

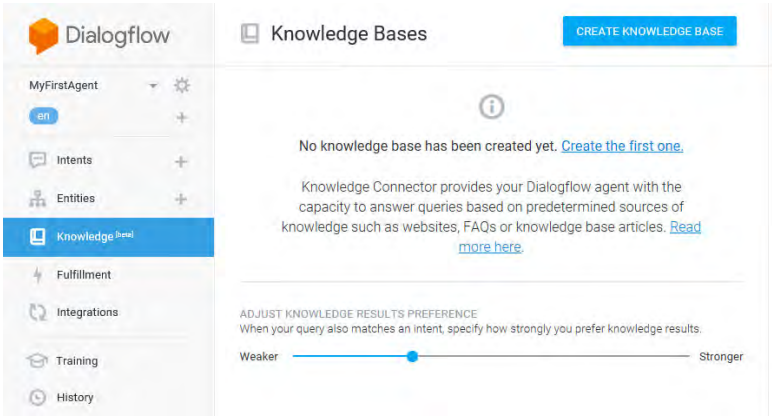
f) Click on **Setting** icon. In the **General** tab, turn on **BETA FEATURES** and click **Save**. Do note the **Dialogflow Knowledge** is still in beta at this point in time.



***If "Service Account" is empty, click on the "+" button.**

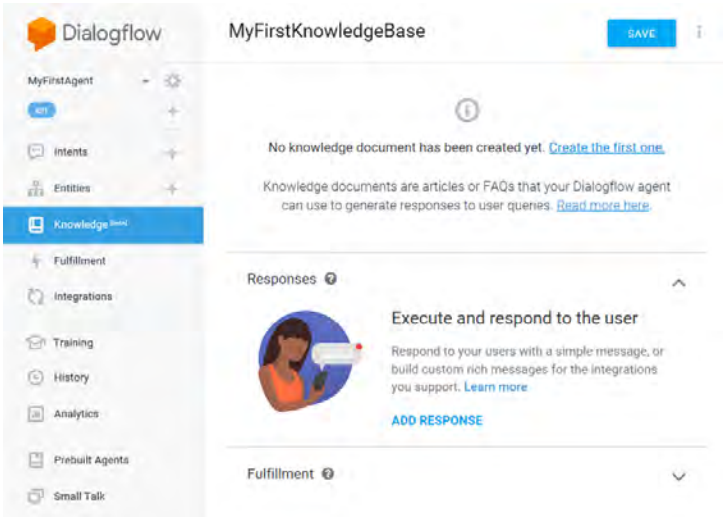
3) Create Knowledge Base

a) On the left panel, click **Knowledge [beta]**, and then **CREATE KNOWLEDGE BASE** to create a new knowledge base.



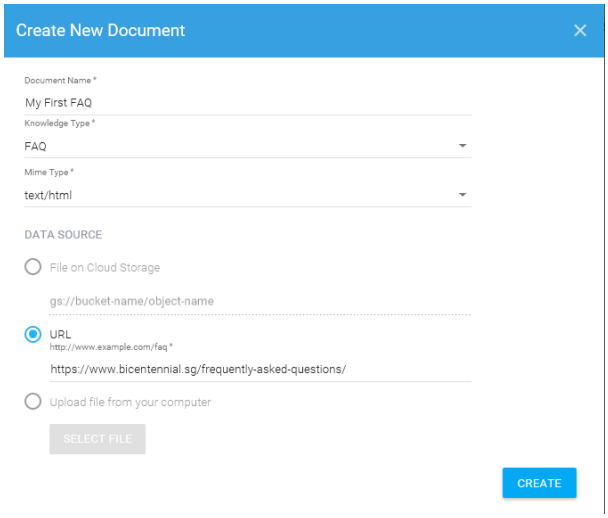
b) Enter a suitable knowledge base name and click on **SAVE**.

c) Click on **Create the first one** to create a knowledge document. Knowledge documents are articles or FAQs that your Dialogflow agent can use to generate responses to user queries.

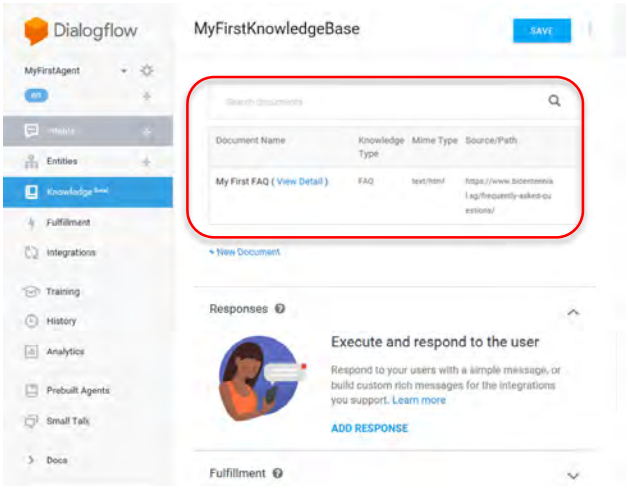


d) Enter values according to your data.
Document Name: this can be anything that you want
Knowledge Type: this can be selected as FAQ. For FAQ, only text/html and text/csv is supported for now.
MIME type: this is the type of data you are going to feed to the bot. It has options like (text/plain, text/html, text/csv, application/pdf)
Data Source: this can be provided as a file on cloud storage or local, you can also give this as a URL to a public page. For our purpose of this activity, we will use a URL. Enter https://www.bicentennial.sg/frequently-asked-questions/. You are free to try with another FAQ URLs.

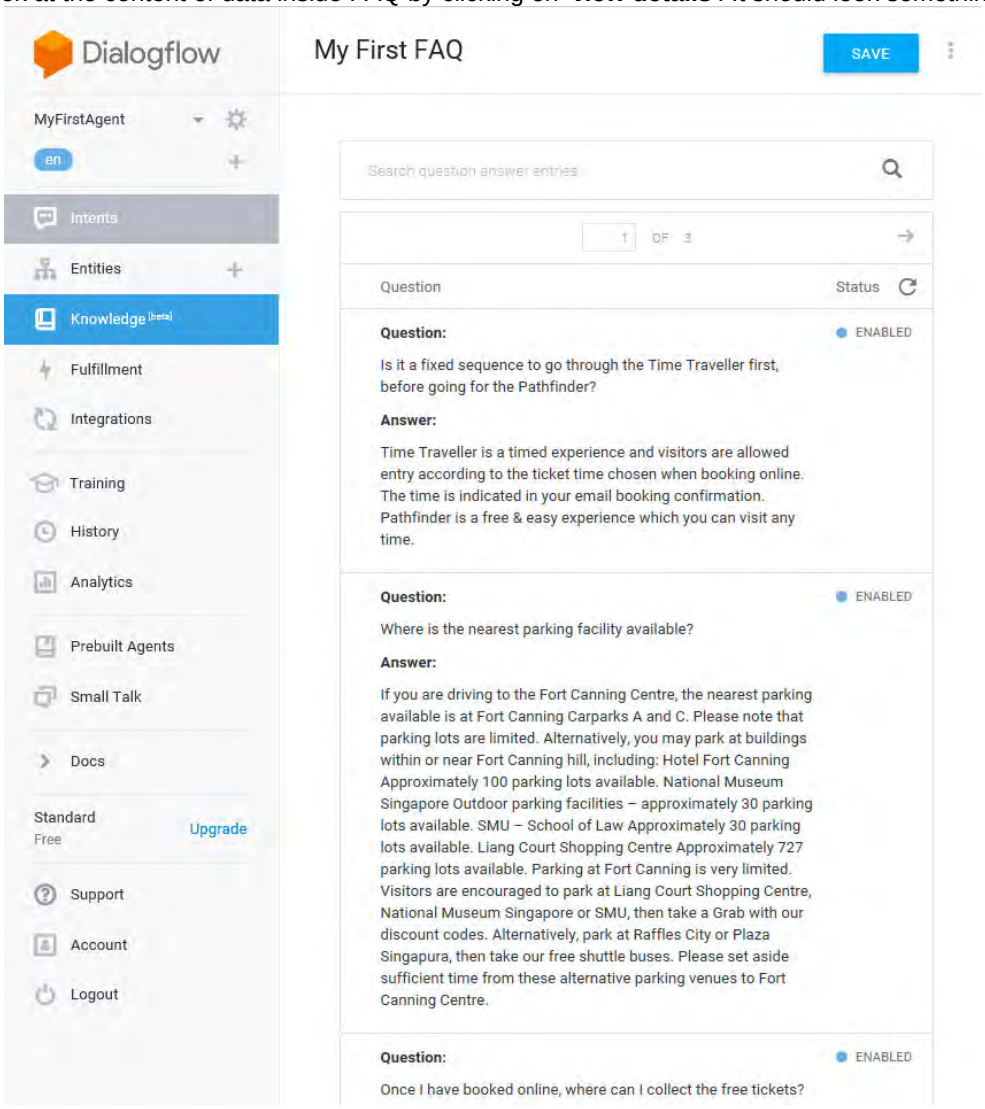
Click **CREATE** to continue.



e) Wait for it to generate all the data from the source that you have provided. This usually takes about 2–5 minutes depending upon the size of your data. You can see 'My First FAQ' knowledge base has been created.



f) You can look at the content of data inside FAQ by clicking on '**view details**'. It should look something like this



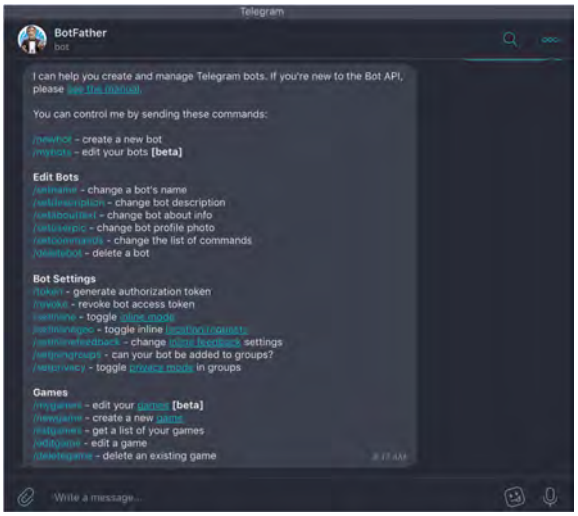
g) Click the menu button (three vertical dots) beside the SAVE button and select **back** in the dropdown menu to return to the Knowledge base view.

h) Select "ADD RESPONSE"

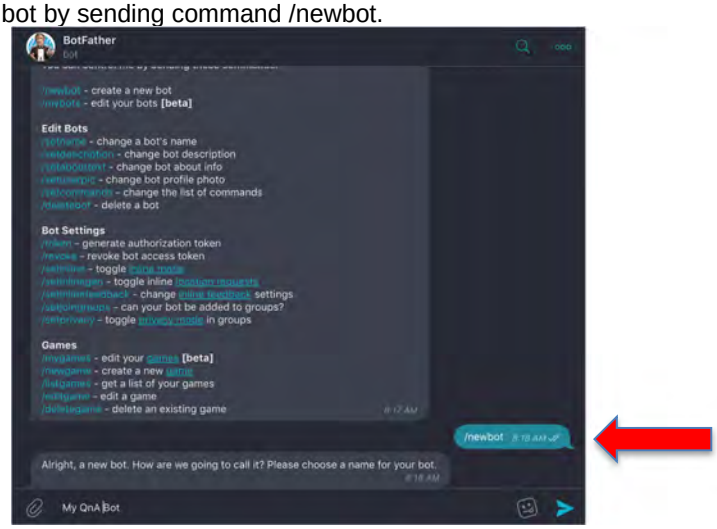
4) Create telegram bot

For most integration, you must provide configuration information to run your bot with Dialogflow. Most channels require that your bot have an account on the channel, and others, like Facebook Messenger, require your bot to have an application registered with the channel also.

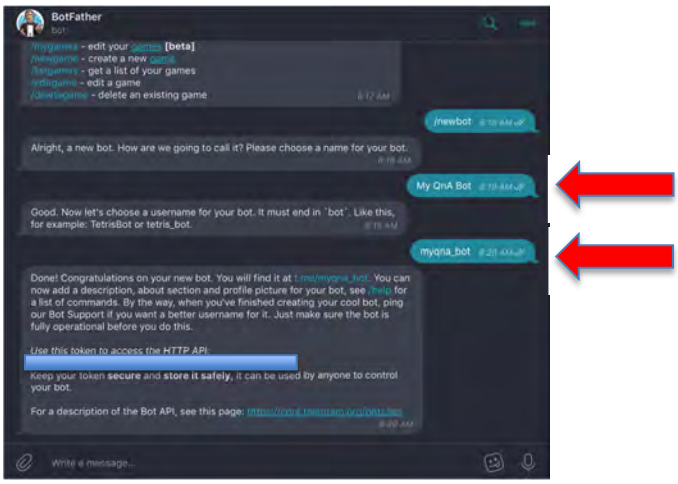
a) If you do not have telegram installed on your mobile phone, please install via Google Play store or Apple App Store.
b) If you have telegram installed on your laptop, use this link to connect to Bot Father (https://telegram.me/botfather) or add botfather. Otherwise, if you are using telegram on your mobile, start telegram and search for botfather.
c) Click on **Start** to start a conversation with botfather.



d) Create a new Telegram bot by sending command /newbot.



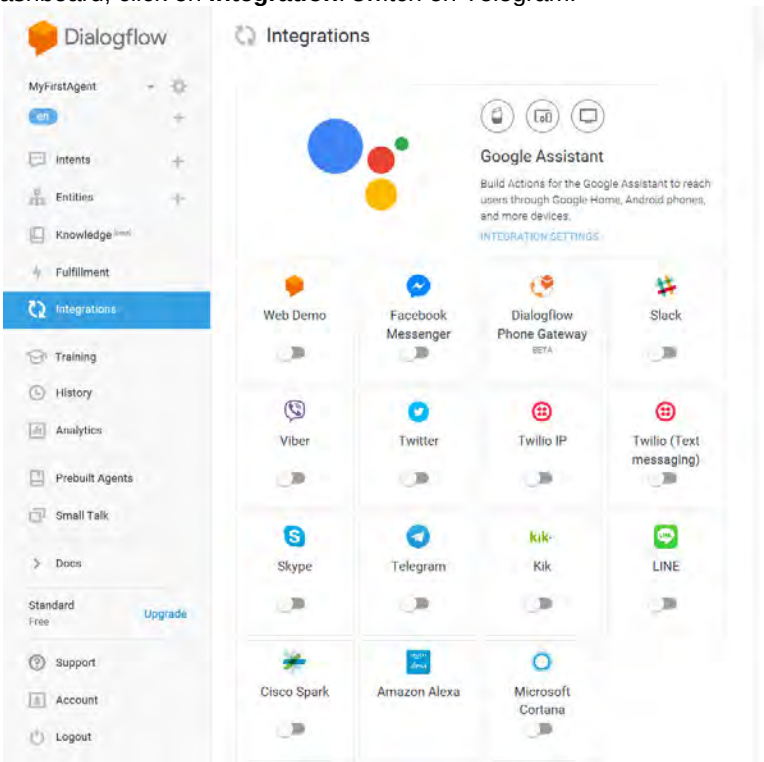
e) Give the bot a friendly name.
f) Specify a username.



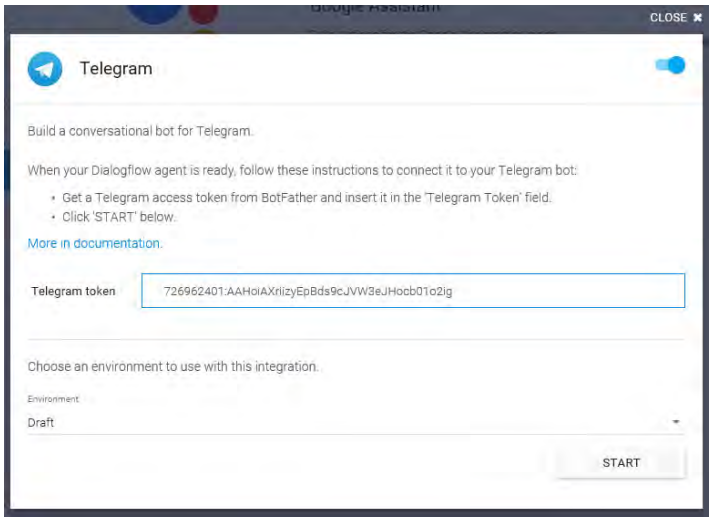
g) Copy the Telegram bot's access token provide in the screen above. You will need this info to configure integration for the Agent.

5) Integrate telegram with Dialogflow

a) Back to Dialogflow dashboard, click on **Integration**. switch on Telegram.



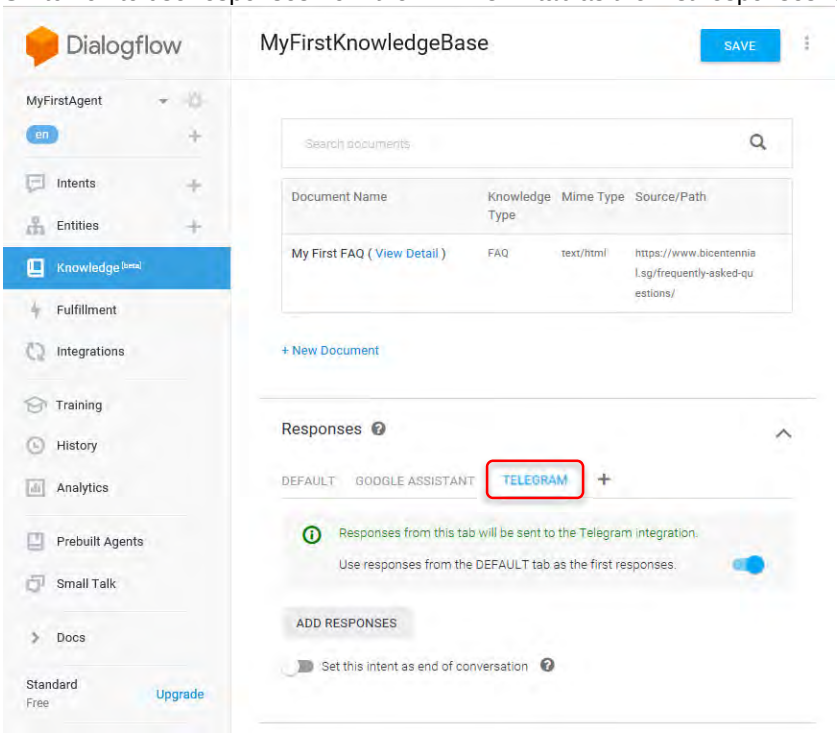
b) In the Telegram configuration pop up, enter the telegram bot's access token you obtained in Step 4. Click **START** and after a while, you should see a flash message saying the bot is started.



c) Click on Close (top right) to finish the configuration.

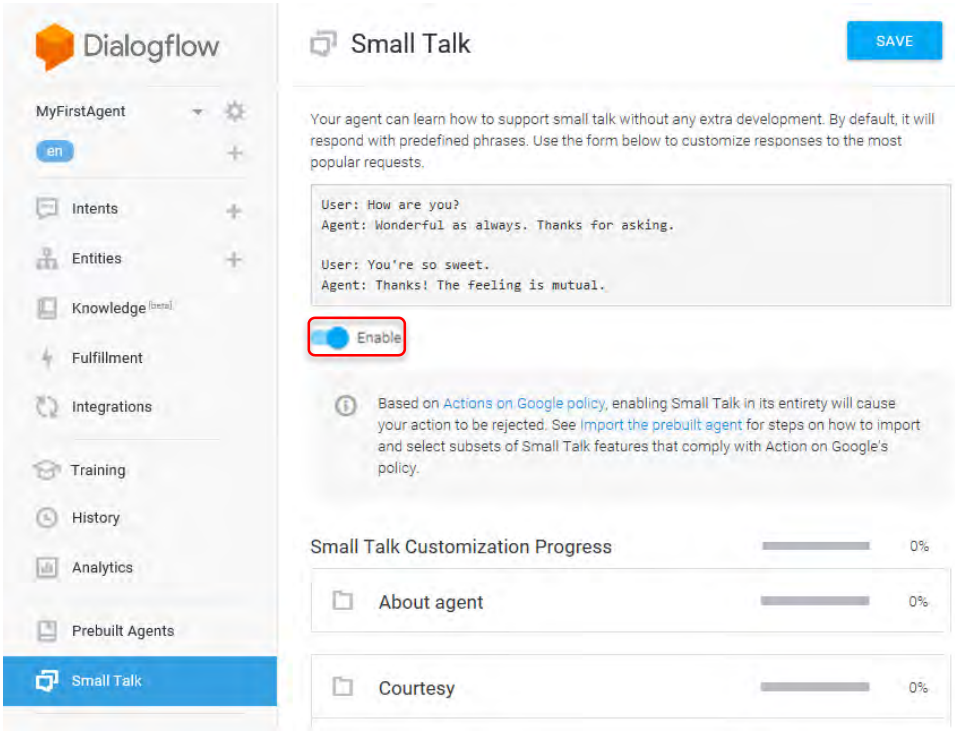
6) Add Responses

a) Switch on to use responses from the DEFAULT tab as the first responses for telegram. Click **Save**.



7) Small Talk

a) Your agent can learn how to support small talk without any extra development. By default, it will respond with predefined phrases. (By default, if you send “How are you?”, agent will respond with “Sorry, can you say that again?)



8) Test your agent from Telegram

a) Launch telegram and add your bot. You can search for your bot by adding a @ to the username you used in step 3.
b) Click start to start a conversation with the bot.
c) Start asking your bot!

Activity wrap-up:

We learn how to

- ❑ Create a Dialogflow agent.
- ❑ Create a Dialogflow knowledge base
- ❑ Use Small Talk to add personality to agent
- ❑ Integrate Telegram to agent

Activity 2 – Training an Image Classifier

In this activity, we will:

- ❑ Learn how to build a classifier through Clarifai AI service.
- ❑ Learn how to train a (supervised learning) image classification model
- ❑ Learn to evaluate the model
- ❑ Deploy your model to a chatbot

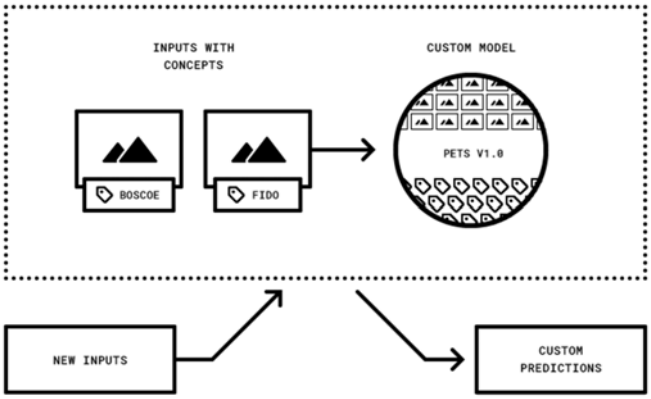
NOTE:

- ❑ **Same security and network settings may disable proper registration on Clarifai. Please check that you are using Guest@RP wireless network!**

Custom Training

Custom Training is about teaching computers to see the world in a way that is specific to your own content and context. You can think of Custom Training as a series of inputs where you ultimately teach a neural network what concept_1 is and what it is not. We define the training data, taxonomy (the model's 'concepts', this is equivalent to "labelling") and other performance characteristics.

If your content is visually distinct and easy to identify, 25-50 positive examples per concept will provide robust and accurate predictions. ***A 'concept' is synonymous with 'tag', 'category' or 'keyword'.*** Concepts are your business' world view as to what an object, visual pattern, or style may represent.



What makes for a well-built concept?

- **Accurate labels**. Mis-labelled images introduces noise into your model and can lead to weak or confusing predictions.
- **Balanced training data**. Skewed training sets where several concepts have 5-20x as many positive training images as others may affect model performance.
- **Matching training and prediction context**. It's crucial that your training images for your concepts resemble the conditions and context of imagery you'll be making predictions on.

As an example, training a flower identification model solely with stock photography and then attempting to predict on user generated smartphone photos will not be ideal.

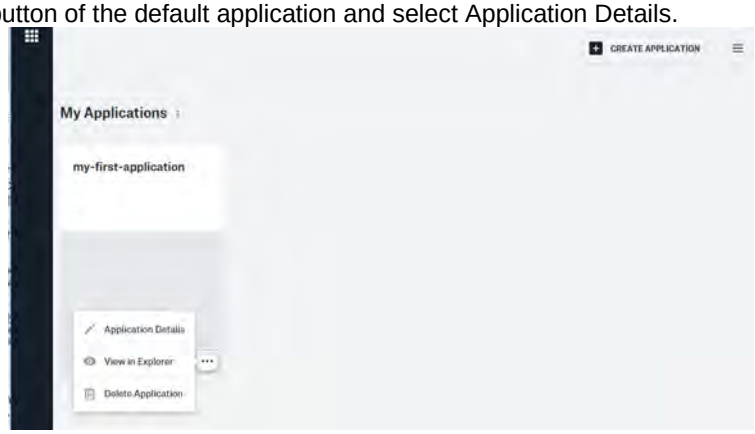
1. Create an account with Clarifai

a) Sign up for an account at clarifai.com (https://portal.clarifai.com/signup)

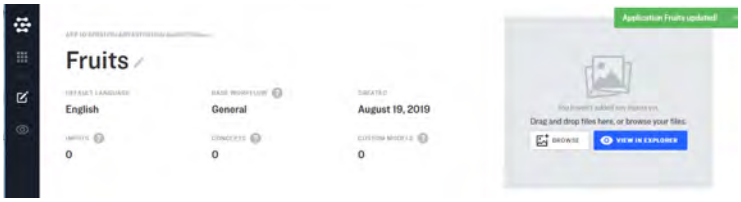
2. Edit your application

Operations are tied to an account and application, and any model that you create and add images to will be contained within a specific application. By default, you'll have an application in your account already so let's change that one's name to whatever you'd like. We are going to name ours "Fruits" for practical reasons

a) Click on the details button of the default application and select Application Details.

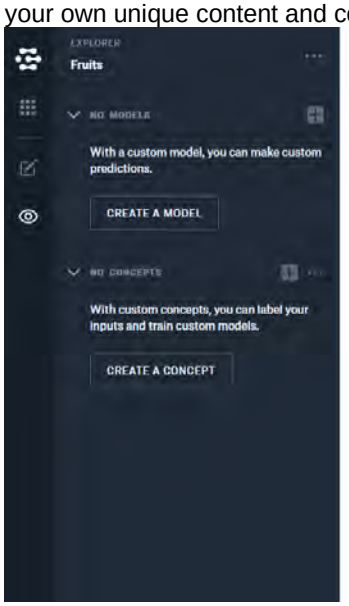


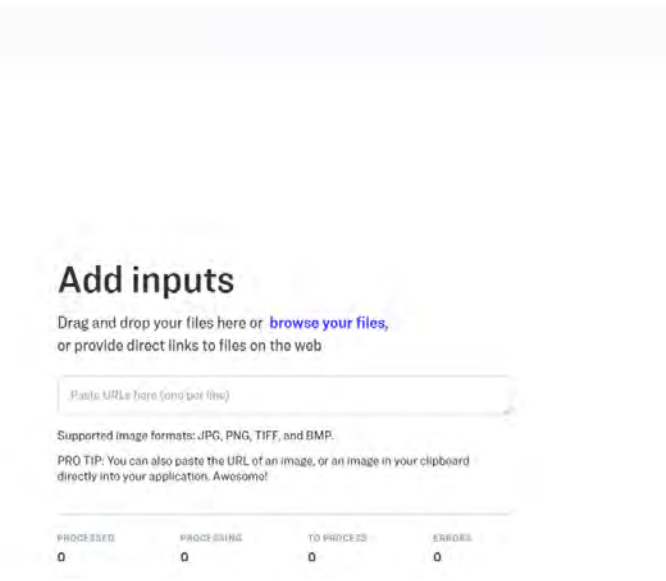
b) Change the application name to Fruits.



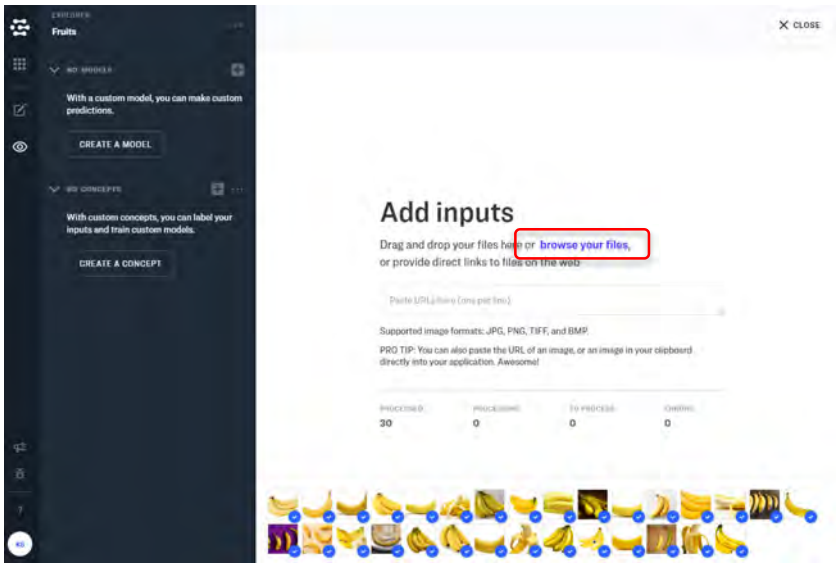
3. Add images to your application

a. Click **on VIEW IN EXPLORER** button to start adding images to your application. Click **browse your files**. Custom models are built by training on your own data, and they will be able to make predictions specific to your own unique content and context.





b. Start uploading your images, say, those banana images. It may take a while to upload the images. Once uploaded, you will see tick mark with the uploaded images.



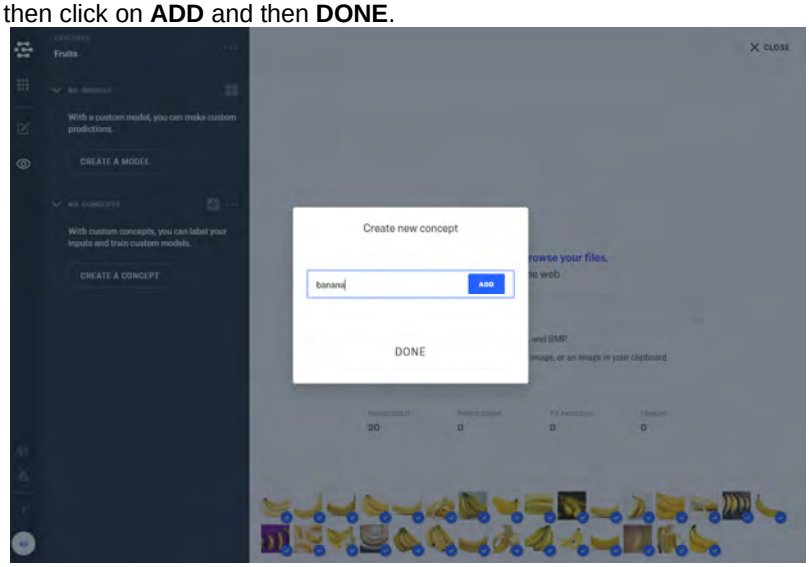
*Note: It may take a while to upload the images.

4. Add concept(s) and create a model

A model is created as soon as you create your first concept. The model name inherits the name of your Application (though you can always change that later).

a) Click on **CREATE A CONCEPT**.

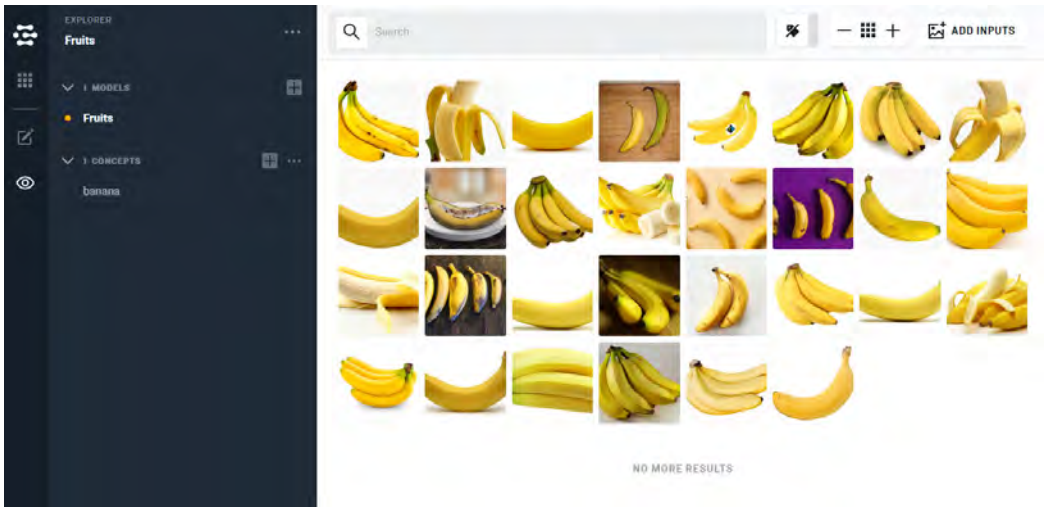
b) Enter banana and then click on **ADD** and then **DONE**.



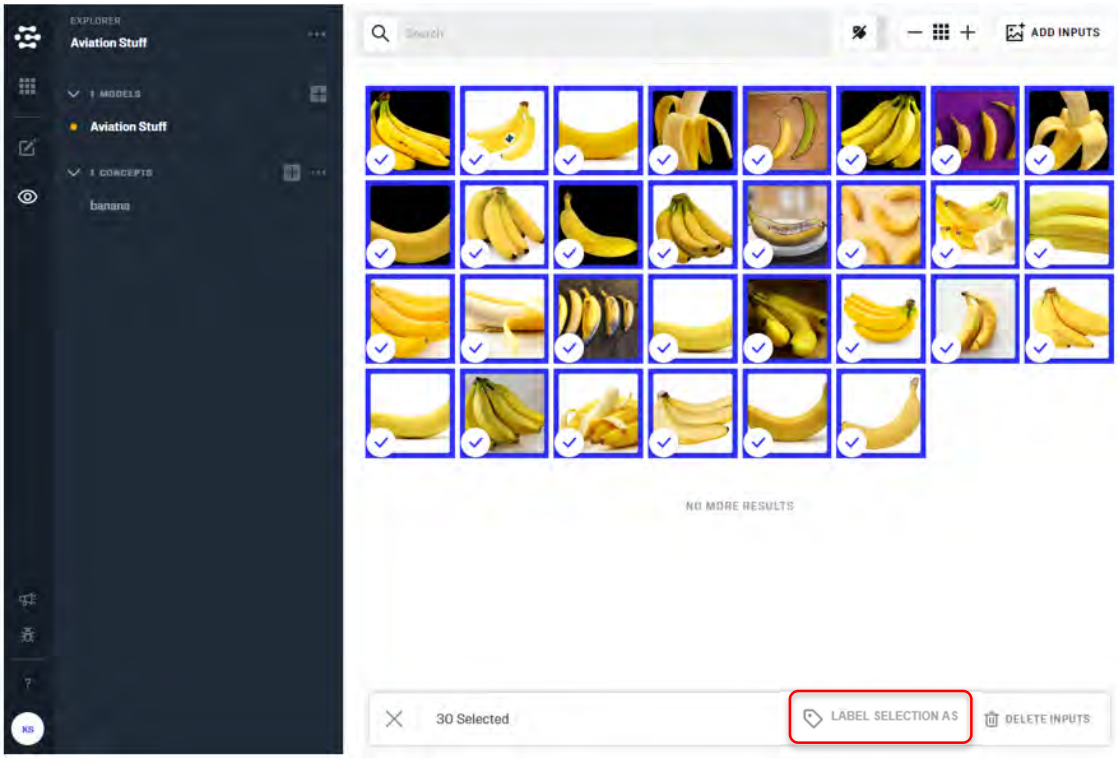
5. Label your images with those concepts

Now that you have some concepts, let's add them to your images! All images are in your application are referred to as inputs so if you ever see that lingo, the names are essentially interchangeable. As you add concepts to images, you'll notice the counts in the Concept Panel on the left showing the updated total of how many images are labelled with each, respectively.

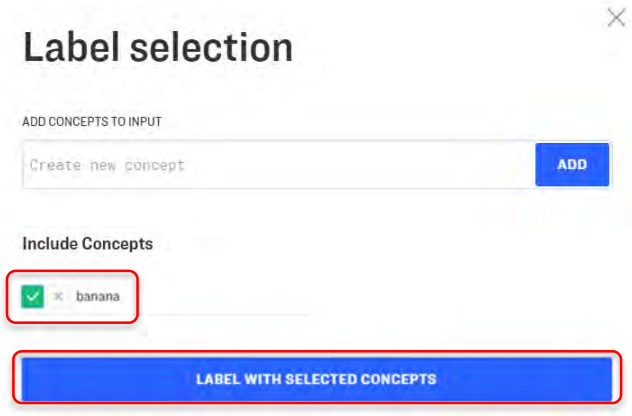
a) Click on CLOSE (if necessary) to bring you to the following screen.



b) Select all the banana images and select **ADD CONCEPTS**.



c) Select **banana**



d) Click **LABEL WITH SELECTED CONCEPTS.**

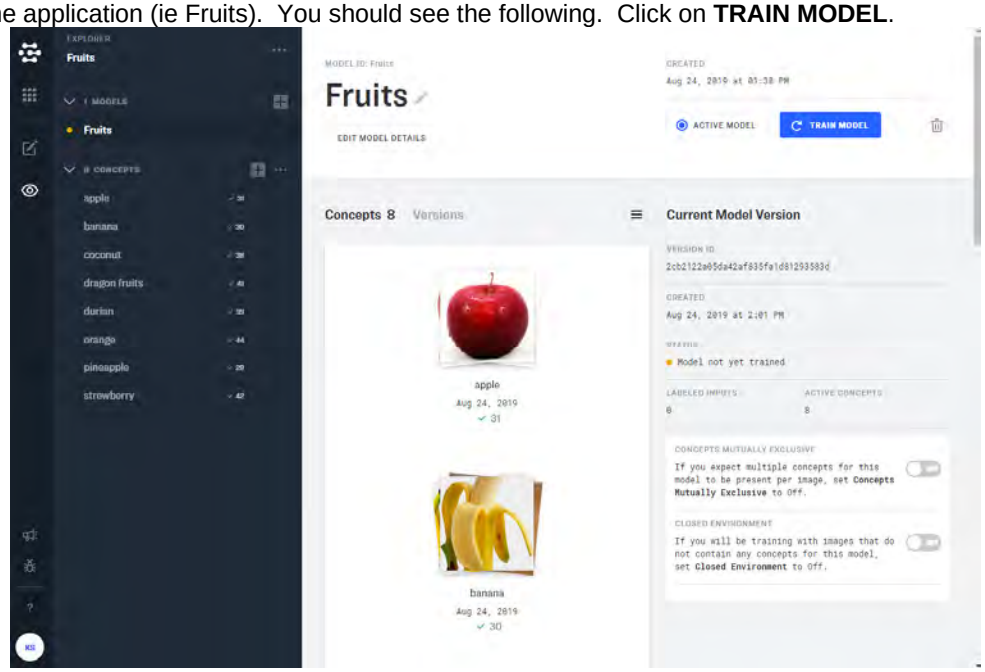
6. Add more images and concepts

Repeat step 3 to 5 for the rest of the fruits folders (except test_images)

7. Train the model

Once all of your images are labelled, let's go ahead and train your first model! You can either do this via the little 3-dot menu next to the model name or you can click on the model name and then click the Train Model button on the ensuing page.

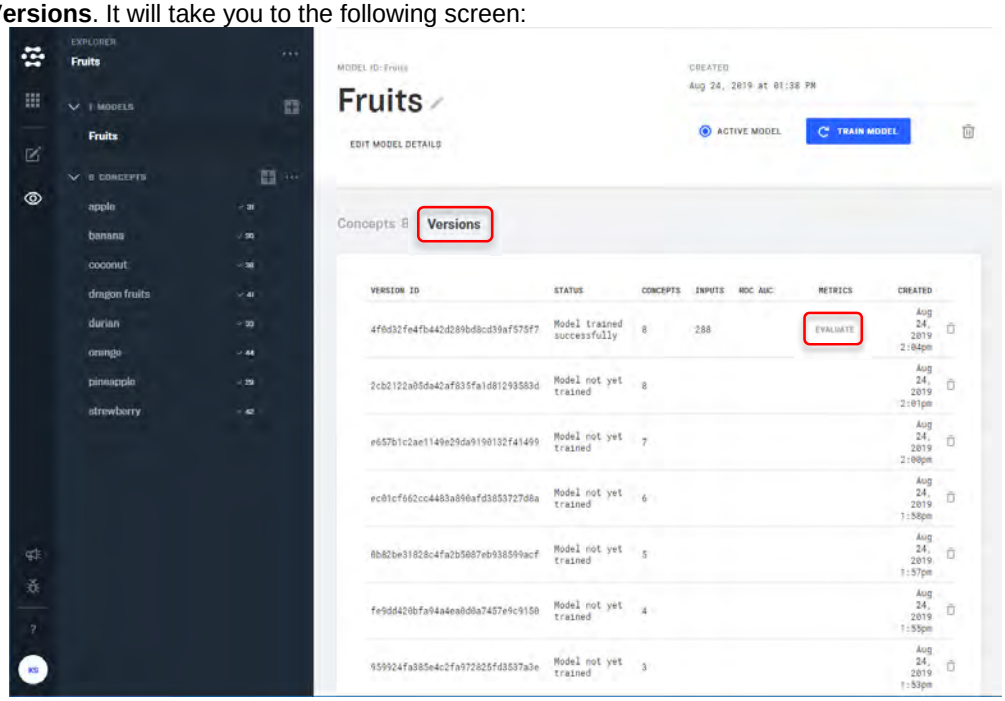
a) Click on the application (ie Fruits). You should see the following. Click on **TRAIN MODEL**.



8. Evaluate your model

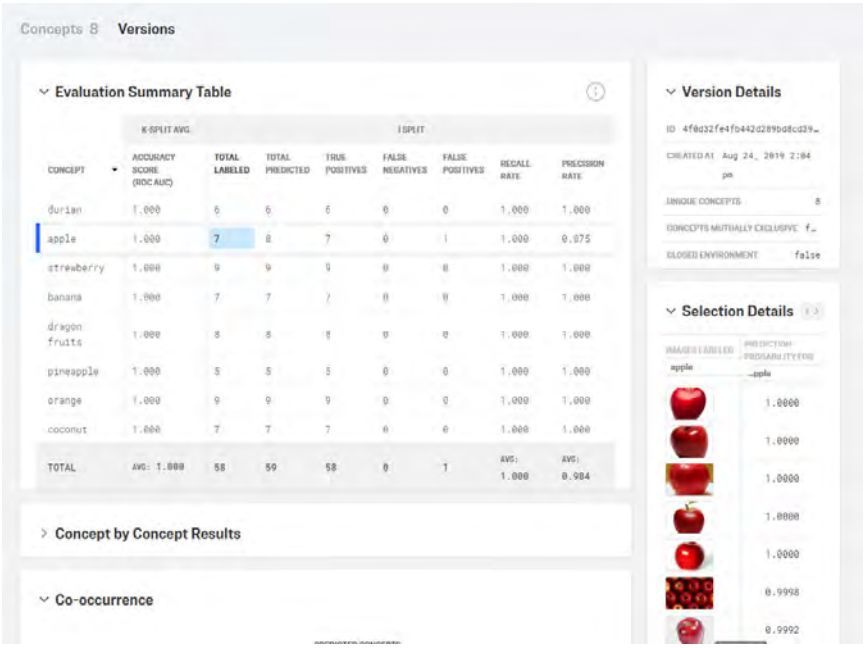
a) Click on the model name that appears on the left panel to bring you to the screen above.

b) Click on **Versions**. It will take you to the following screen:



c) To evaluate your model, click the **evaluate** option under Metrics. This will take a short amount of time

depending on the number of images added to your model. Our simple model should be evaluated in seconds. Once the evaluation is completed, the "Evaluate" option will be changed to a “View” option. Click it, and you will see the evaluation results.

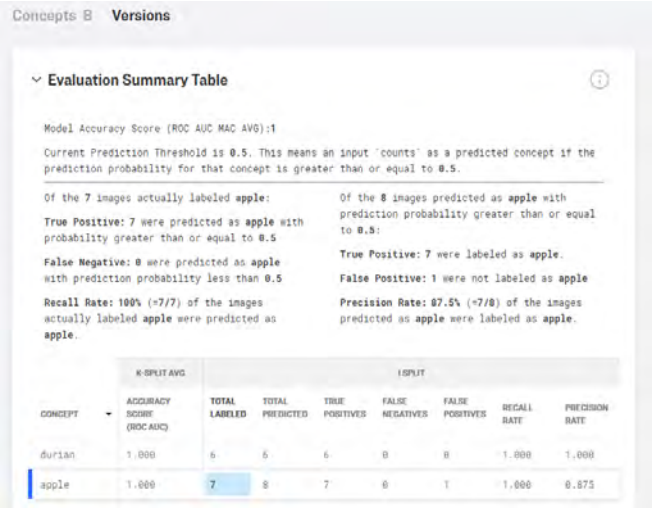


d) Evaluation results will be categorized under 4 headings: **Evaluation Summary Table**, **Concept by Concept Results**, **Co-occurrence**, **Precision-Recall**, and **ROC Curves**. Clarifai does the evaluation using a method called K-split cross-validation on the data.

Clarifai uses a random subset of our input data as test data to test against a new model with the remaining data. Then, it predicts the test data against the new model. After that, it compares the predicted labels with the real labels. After performing this multiple times, Clarifai gives each concept a score.

There is a value called ***probability threshold***, which determines the point at which concepts will be classified as either positive or negative. For example, an image is counted as belonging to a particular concept, such as a pineapple, only if its prediction probability of that image for the pineapple is higher than the threshold value. The default threshold value is 0.5. You can change it as you want.

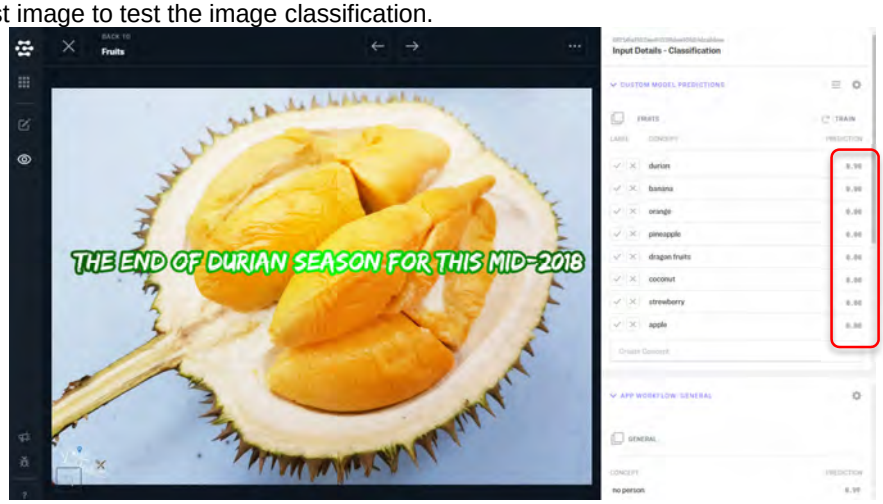
e) Click on the “i" icon to see the evaluation details of a particular concept.



9. Test your model (Upload some external images and see how the model is performing)

a) Click on the application, **ADD INPUTS** to upload a test image. (for this exercise, use an image from the test_images directory)

b) Click on the test image to test the image classification.

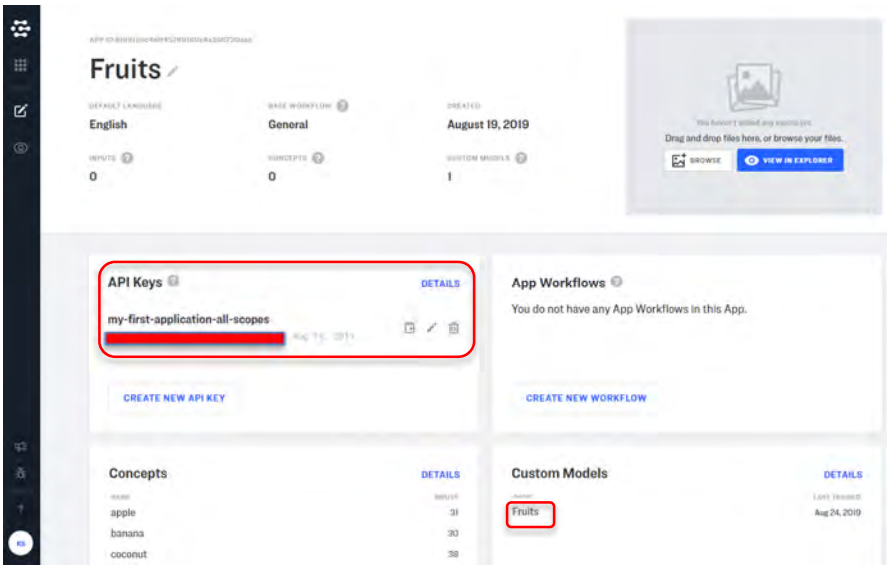


10. Adding negative examples when the model is getting confused (optional)

Up until this point we have only added positive examples to the model (e.g. saying "this is definitely concept X"), but we haven't added any negatives that say the opposite (i.e. "this is not concept X"). A robust and well performing concept is typically made up of both positive and negative examples with a 3 or 4:1 ratio, respectively, but adding too many will be counterproductive so be careful. If you see in one instance that you're getting a false positive for "apple", for example, you may want to teach the system that it's wrong.

11. Model Deployment and usage

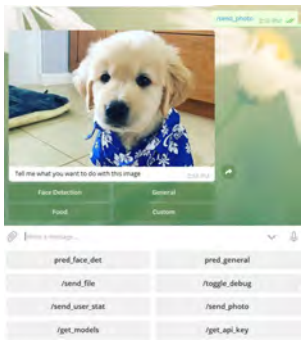
a) Once you trained a model, an API key is automatically created for you. To use this model in an application, you will need this API key and the model name. Click on Application Details to get this information from the screen.



12. Use the model via a chatbot (optional)

a) Launch you telegram and add the bot @tooka_skw_bot (use search)

b) To see an example of how to perform some image classification tasks, send a /send_photo.



c) For this chatbot app, to use your custom model, you need to

   a. Use the command **`/set_api_key your_api_key`** to specify your application. Replace **`your_api_key`** with the key you obtained in step 11a.

   b. Use the command **`/set_model_name your_model_name`** to specify the model you want to use. Replace **`your_model_name`** with the model name in step 11a.

d) Send a photo to the chatbot, in the inline keyboard, select Custom. After a while, the bot will return the result of the classification.

e) Test out the model you trained using other images. This chatbot also provide demonstration of some other AI services provided by Clarifai AI. You can try out their General model, Face Detection model and Food model.

13. Use the model via a web app

a) Launch you web browser and go to http://kwseow.pythonanywhere.com/forms. This is a demo site created to utilise your model to perform image classification.

b) Key in your Clarifiai API key and the name of the model you trained in the previous step.

c) Select a photo you want to test with and click on submit. After a short while you should see the result of the classification of your model. See the following screen capture:

d) The following code snippet provide gives you an ideal of how this is done.

```
@app.route("/forms", methods=['GET', 'POST'])
def myforms():
    form = ReusableForm(request.form)

    #print(form.errors)
    if request.method == 'POST':
        form = ReusableForm()

        if form.validate_on_submit():
            #write_to_disk(name, surname, email)
            f = form.photo.data
            filename = secure_filename(f.filename)
            f.save(os.path.join('./mysite/static/photos', filename ))
            try:
                #api_key = ""
                api_key = form.api_key.data
                os.environ["CLARIFAI_API_KEY"] = api_key
                clarifai_app = ClarifaiApp()
                #custom model
                #model_name="Fruits"
                model_name=form.model_name.data
                model = clarifai_app.models.get(model_name=model_name)
                response = model.predict_by_filename(os.path.join('./mysite/static/photos',
filename ))

                msg =""
                for concept in response['outputs'][0]['data']['concepts']:
                    msg += "%s|%s\n"%(concept['name'],concept['value'])
                tmp_model = response['outputs'][0]['model']
                msg += "\nUsing model:\n%s|%s\n"%(tmp_model['id'],tmp_model['name'])
                msg = msg.replace('\n', '<br>')
            except ApiError as e:
                msg = 'Error status code: %d\n' % e.error_code
                msg += 'Error description: %s\n' % e.error_desc
                if e.error_details:
                    msg += 'Error details: %s' % e.error_details

                if e.error_code == 21200:
                    api_key=form.api_key.data
                    os.environ["CLARIFAI_API_KEY"] = api_key
                    clarifai_app = ClarifaiApp()
                    msg += "\nAvailable models:"
```

```
                    for model in clarifai_app.models.get_all():
                        msg += "\n%s"%model.model_name

                msg = msg.replace('\n', '<br>')

            #flash('Hello: {}'.format(filename))
            flash('{}'.format(msg))
            #filename = secure_filename(form.file.data.filename)
            #form.file.data.save('uploads/' + filename)
            #flash('Hello: {} {} {}'.format(name, surname,filename))

        else:
            flash('Error: All Fields are Required')

    return render_template('form.html', form=form)
```

Activity wrap-up:

We learn how to:

- ❑ Train and evaluate an image classifier
- ❑ Deploy the model for use in a chatbot and a web browser