[D]: Let’s start our OpenVINO Activity. For this, we will access the OpenVINO DevCloud server by going to the OpenVINO section of the DevCloud

[D]: I’ll give you a minute to get to the screen here.

[D]: After getting here, we will click on File, then put the cursor in new, and select the terminal option.

[D]: In the terminal, we will type the following command in screen and then press enter

[D]: After doing so, you should see a folder called “classification\_mod\_workshop” along with the Reference-samples folder.

[D]: Now, double click on the “classification\_mod\_workshop” file, and then click again in the activity folder

[D]: Then, double click on the “activity\_a” file

[D]: This will open a notebook with our activity on it.

[D]: For people unfamiliar with a Jupyter Notebook, the thing you need be aware of is to double click the top text block in the notebook as I am doing now, and then press the shift key and enter to move to the next block. For the sake of this activity, follow along as I move from one block to the another.

[D]: This activity explains how to use OpenVINO to implement a pre-trained model. The model we will use is “squeeznet1.1”.

[D]: This model is supposed to behave similar than the AlexNet model but using 50x less parameters and retain its performance. We will get this model by using the Model Downloader from OpenVINO.

[D]: We will use it in this case for object detection, and it is capable of classifying and reporting the probability of 1000 different objects including different species of cats, dogs, birds, insects, and non-living objects, etc.

[D]: Let’s look at our first step: Importing the Main Libraries

[D]: The libraries we’ll use are the following:

[D]: The os library, which we will use for file name parsing

[D]: For the ones who don’t know, file name parsing is I don’t know

[D]: The cv2 library, which allows us to use OpenCV and their functions

[D]: For the ones who don’t know, OpenCV is

[D]: The time library, which is used to measure execution Time

[D]: The numpy library, which is used for n-dimensional array manipulation

[D]: The openvino.inference\_engine, which is used to create our Inference Engine Network and Inference Engine Core objects

[D]: These two objects are used to implement our model in an specific hardware

[D]: And finally, the matplotlib library, which is used for displaying output images

[D]: After declaring our libraries, now we can declare some functions that will help us set up our images to input it in our model.

[D]: Little disclaimer, when I say batch, in this case I mean a group of pictures.

[D]: The first function will cover is called loadInputImage. This function has an input parameter which is the path of our input image. \*-highlight both with the cursor-\*

Then we will use OpenCV to convert our path into our desired image, then we will get the image width and height and print the image’s path, width, and height. Finally, the function will return our image.

[D]: The second function is resizeInputImage, and as its name says, it will help us change our picture original dimensions to fit our model’s input. This functions has an image as a parameter, and it returns a resized image.

[D]: Next we have showImage. As you could imagine, this function will display an image in our screen, with the header saying that the analysis results are at the top of the image. The functions parameters are the image name, and the original image. This function doesn’t return anything.

[D]: The fourth function is processAndDisplayResults. This function will show the possible things that our model thought the picture was, as well as displaying our image. The functions parameters are the probability array, the original image, the original image location, and the image name. This functions, like the previous one, doesn’t return nothing

[D]: After that, we’ll have the inferImage function. This will directly get an image and resize it, input it in the model, and process and display the analysis results. The functions parameters are an image, the image location, and the image name. This functions, also like the previous one, doesn’t return nothing.

[D]: The fifth function is batchLoadInputImages. This function will accept a batch of image locations, and it will process them, and then return the batch size,

[D]: The last function we will use is batchProcessAndDisplayResults. This function has the same parameters as the regular processAndDisplayResults function. The difference is this one will be able to handle multiple input processing and displaying.

[D]: Now, we’ll look into setting up our model in OpenVINO.

[D]: First, we’ll use the Model Downloader utility to download our model from the Open Model Zoo.

[D]: Then, we will insert the converter.py command to convert our model into the Intermediate Reference format by using the Model Optimizer. This make take a while to load.

[D]: After that, we’ll have to configure some things so we can use the Inference Engine.

[D]: The first thing we’ll need assign the variables **model\_xml** and **model\_bin** to store the path to the .xml and .bin files of the model. As we mentioned before, the first one describes the Neural Network topology, and the second one contains the weights and biases of the Neural Network.

[D]: The second thing is the **input\_path** which is the location of our test image, then, we have some library extensions to use a CPU from the Intel devcloud, and we set up our device name CPU. After that, we set up the **report\_top\_n** variable to 10.

[D]: This variable represents the number of predictions our model will make given an input image. Next, we will set out the **labels\_path** variable to the file containing the labels of our model. Finally, we will print out what we just did to the user.

[D]: Next, we will create our inference engine instance. This means we’ll create an inference engine object.

[D]: After that, we’ll create the Inference Engine Network object, and we will load our files .xml and .bin into it.

[D] Then, we’ll check if any of the layers are not supported by the plugin we will use.

[D]: Now, we’ll load our Inference Engine Network into the plugin and run the inference. We will store the load into the variable **exec\_net**, which will be used later to actually run the inference. After loading it, we store the names of the input and outputs in the variables **input\_blob** and **output\_blob** respectively. Finally, we’ll store the model’s input dimensions into the following variables:

[D]: **n**, which will store the batch size; **c**, which will store the number of input channels, in this case that’s a channel per color, which is RGB; **h**, which will store the input height; and **w**, which will store the input width. After that, we’ll print what each one of those are.

[D]: The next step is a “cosmetic step” in quotation marks. For each detected object, the output from the model will include an integer to indicate which type of trained object has been detected (e.g. daisy, bee, etc.). We will use the following command block to translate the integers into text by mapping it with the squeezenet1.1labels file we have in our activity folder. We will do this by using our **labels\_path** variable we created on Step III, and we will insert the map into a variable called **labels\_map**.

[D]: Now, we’ll prepare our inputs. First, we’ll create a variable image to store the output of our loadImage function defined before, and then we will use a variable in\_frame to store the output of our resizeInputImage function. Finally, we will display our image.

[D]: After that, we will run the inference. At the same time, we will measure and display the time it took to complete the inference. The line to run our inference is highlighted here, and as we can see, we are using our previously declared **exec\_net** variable, where we store our load of our model into the Inference Engine Network. We’ll store the results of our inference run in the variable **res**.

[D]: Our final step, is to process and display our results. For that, we’ll call our function processAndDisplayResults. As you can see, the results output is being called, as well as the input image itself, its location, and its name.

[D]: Before we go into our activity for today, we will look out for some things we should be aware of.

[D]: The first thing is how to load another image. We can do this by setting our **input\_path** another image location. We’ll use the ai1\_dogo.jpg image we have in our activity folder. After that, we just need to call our loadInputImage function, and then use our inferImage function to process and display our image into our model.

[D]: The second thing is how to load images from the internet into our model. For this, we will need to go into google or any other searcher and type the image of the object we want to analyze. The trick here is that in the url or link of the image, it must end in .jpg since that’s the format we are using. For example, in this block I’m using an image I found on google of an Anemone, and I’m repeating we saw before.

[D]: Finally, we’ll cover how to load multiple images into our model, and run it. To do this, we’ll need to create two arrays: one that stores our file paths (which can be from our activity folder, or any outside image found in the internet), and other that stores the image names. Then, we will use our batchLoadInputImages function to load our images. The input for this function, as you learned before, it’s the array that contains our image’s locations. And finally, we will print out the number of images we printed.

[D]: After loading the images, we will run our inference again to account for our new inputs. We’ll basically change the net batch size, and then re-load the Inference Engine Network. At the last step here, we’ll run and get the results of our inference, same as in step 10.

[D]: Finally, we’ll finish by calling our batchProcessAndDisplayResults function to process and display our batch of images. Recall the function we created previously (the last function we created before going into implementing our model.)

[D]: Well, now I tell you about our activities. Your first task is to run this model using a batch of images generated by the Generative AI of your choice. You can either drag your generated image into the activity folder, or you can use a link to the image.

[D]: Remember that no matter what image you use, you’ll need to use the .jpg format. I’ll show my example now.

[D]: Your second task is to change the processAndDisplayResults format to something different. This means either change some of the text used to display our results, to using the matplotlib library to show a graphical representation of the data. I’ll also provide an example for my attempt.

[D]: The final task is a little more challenging. You’ll need to change the following things: the number of possible things that our model outputs, the time units from milliseconds to seconds, and finally, and probably the most challenging one, the squeeznet model from 1.1 to 1.0. Again, this last part is not so intuitive, so I won’t expect everybody to finish it.

[D]: If you need any help, please ask me or any of the troubleshooters around you to help you.