User Guide
Event driven Classification of images: sports or non sport events
Group MCS2

By:
Ng Yu Kang (30897343)
Lin Chen Xiang (30897300)
Edmund Cheong Zheng Hong (32136323)

Table of contents

End User Guide

1. Uploading image to be classified

- First, select an image by clicking on Choose File and navigate to the folder with the image, and select the image
- Then, click on Submit, and wait for the website to return the results. This may take a while.

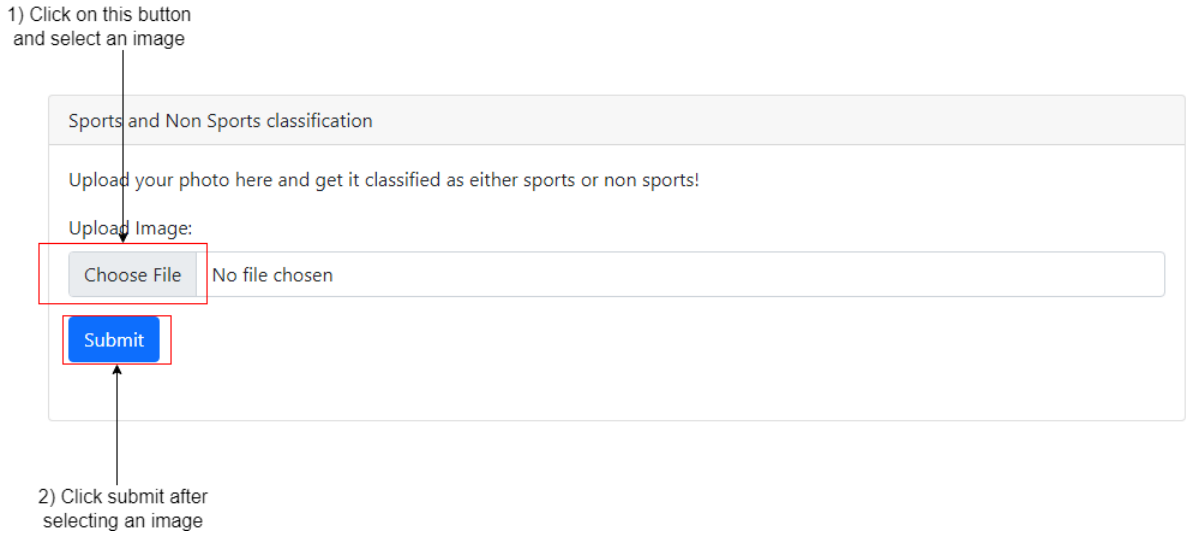

Figure 1: Image outlining steps to use the website

2. Interpreting the results

- The website will render the image uploaded as well as the results
- The first line as shown below as the first row of red boxes (illustration only) shows the main label category (Sports or non sports) and the probability of it.
- The second line (second row of red boxes, illustration only) will only appear if the main category is classified as sports. It shows what type of sports and the probability of it.
- The second line may output “Failed to detect sub label as sports” if the model predicted it as sports, but predicted the sub-label as a non-sport category (Such as walking, driving, etc).

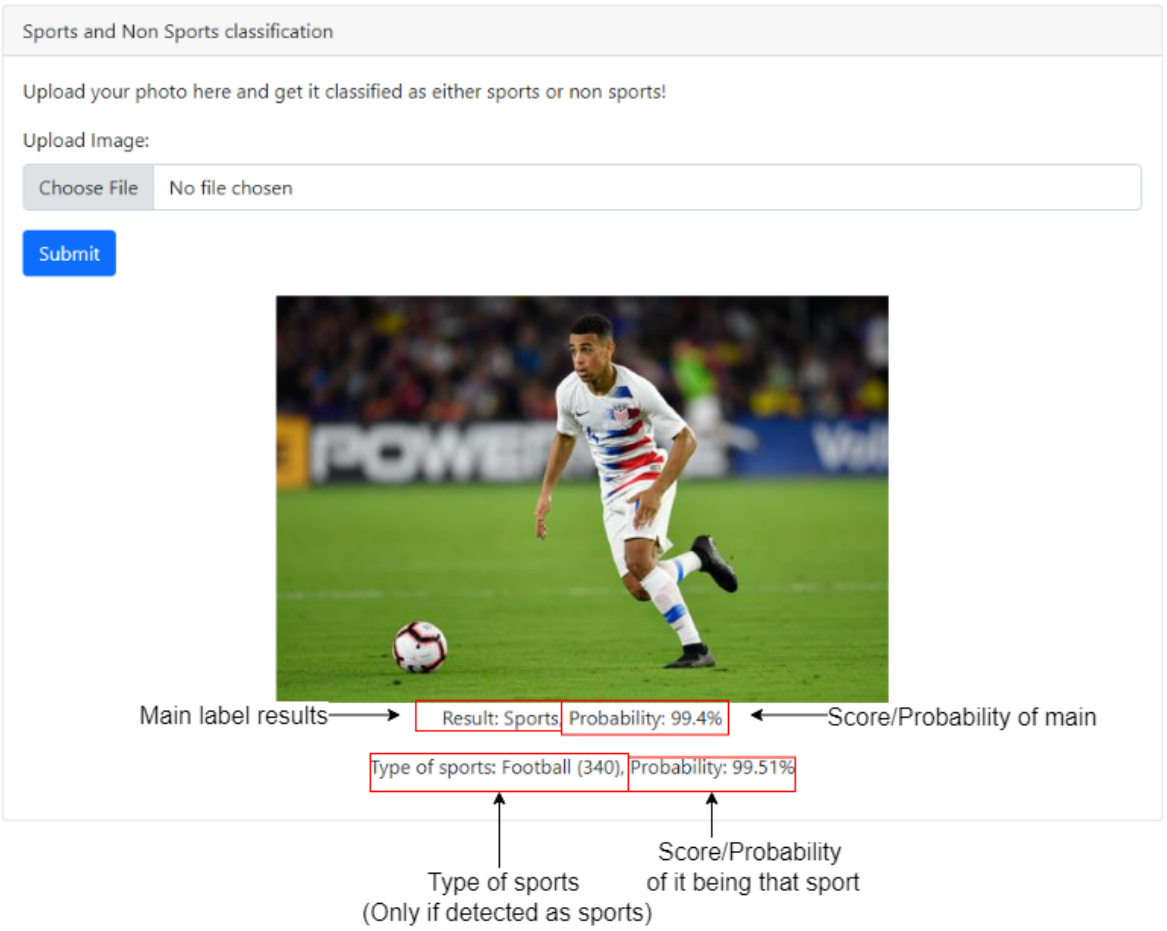

Figure 2: Image outlining how to interpret the results

Technical Guide

1. Setting up Anaconda with required libraries

Before continuing to either subsection, please install Anaconda from https://www.anaconda.com/. Afterwards, download the project files by pulling it from Git or whichever method suits you. Choose either one of the following methods below

Option 1: Download only required libraries via .yaml file (add to base env)

This method is recommended only for Windows based platforms, because the environment file is created from a Windows based platform. Do the following:

- Launch Anaconda prompt (either directly from search, or from anaconda navigator)
- Enter the following command: conda env update -n base --file env.yaml
  The env.yaml should be included with the project files when you pull it.
- Wait for the process to finish.

Option 2: Create new virtual environment directly from .yaml file

This method is also only recommended for Windows based applications. Do the following:

- Launch Anaconda prompt (either directly from search, or from anaconda navigator)
- Enter the following command: conda env create -f env.yaml
  The env.yaml should be included with the project files when you pull it.
- Wait for the process to finish.

Note that by creating a new virtual environment, you will be also using up more space to download all the anaconda libraries again.

Option 3: Manually install required libraries

For this, you can either choose to create a new virtual environment, or just add the required libraries to your base environment. This option is recommended for all types of operating system.

If you plan to create a new environment, else skip these steps:

- Open anaconda command prompt from either search menu or anaconda navigator
- Enter this command: conda create --name myenv
  (Where myenv is whatever name you want to name it)
- Switch over to the env by entering: conda activate myenv
  (Where myenv is whatever you named it in the step before)

Now, for both new virtual environment and base environment:

- Navigate to PyTorch website (https://pytorch.org/)
- Select the stable build, which operating system you are using, Conda, Python and CUDA 11.3
  (MAC Users only) Select CPU for Compute platform. Do note that you will be either not able to train the model afterwards, or you can but training on CPU will take forever. You can however still use the evaluation part of the model.

- Copy the command generated from the run this command and paste it into the anaconda command prompt.
- Wait for the process to finish
- Enter the command: conda install captum -c pytorch
  This will install Captum which we will use to interpret the model.

Screenshot below outlines the step above to install PyTorch

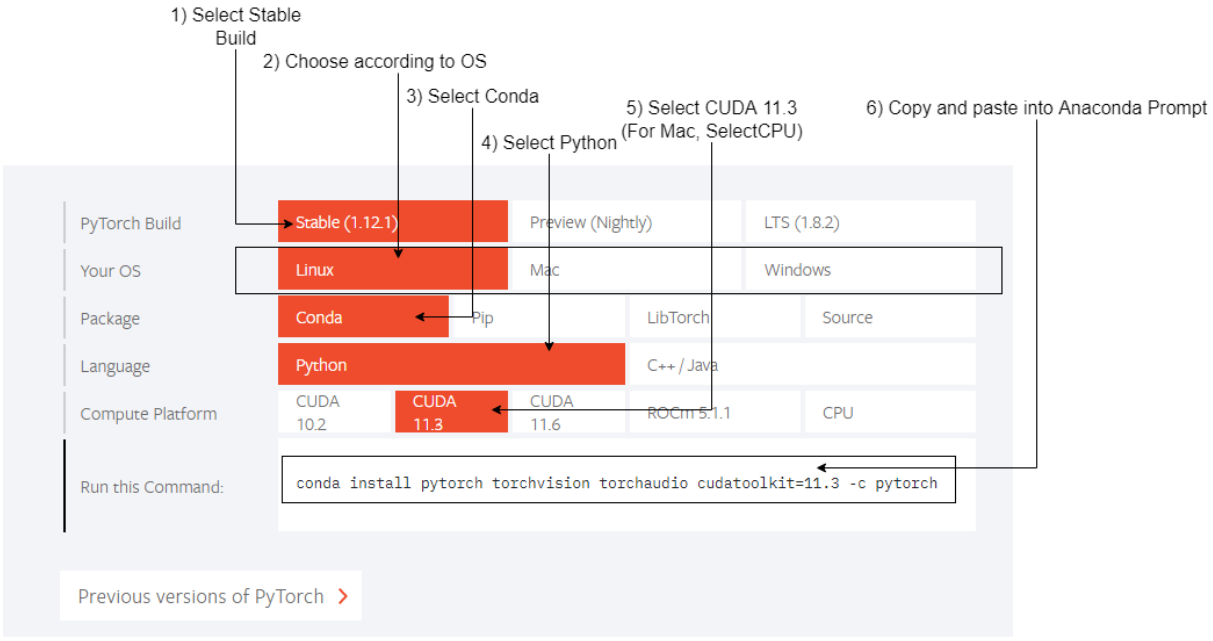

Figure 3: Image outlining what to select and copy

2. Training the models

If you wish to replicate the models on your computer, or you want to add extra data and train the model again, follow the subsections below, else, skip to the next section to learn how to set up the Flask website, with the models we have already trained and given inside the project file.

2.1 Opening Jupyter Notebook

As the project may be placed in a custom directory (on another drive), we are standardising the guide to use the following step to open Jupyter Notebook. Follow the instructions:

- Open Anaconda Prompt from start menu or Anaconda Navigator
- Enter the following but replace H with the letter of the drive the project is located in:
  jupyter notebook --notebook-dir=H:/
  For example, if my project is located in the E drive, it will be:
  jupyter notebook --notebook-dir=E:/
- Jupyter Notebook should have been launched in your browser. Navigate accordingly to your directory where you stored the project.

2.2 Setting up the dataset for training

2.2.1 Placing the dataset folder

- Make sure that the dataset is downloaded and the folder of the dataset is named as “dataset **(without the quotes)**.
- Please also make sure that inside the dataset folder, there are only two folders named “sport” and “non sport” **(without the quotes)**. Screenshot below illustrates how it should look.
- Place the dataset folder at the project root, where the ipynb files are located at.

The dataset folder:

Figure 4: How the dataset folder should be

Inside the dataset folder:

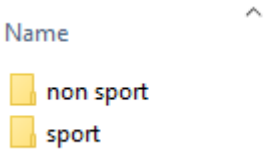

Figure 5: How the folders inside dataset should look

2.2.2 (Optional) Adding extra data

To add to the dataset, if the category does not exist:

- Create a folder in either the sport or non sport folder according to what it is
- Put all the related images in the newly created folder.

To add to the dataset, if category already exists:

- Put the images in the pre-existing folder.

Afterwards, please open check_corrupt_image.ipynb

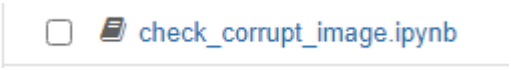

Figure 6: The file to open to run code to check for bad image and generate new mean and standard deviation

and run all the code. Please take note and record the output of these two code, for example:

```
In [5]: imgs.view(3, -1).mean(dim=1)
Out[5]: tensor([0.4757, 0.4559, 0.4415])

In [6]: imgs.view(3, -1).std(dim=1)
Out[6]: tensor([0.3124, 0.2979, 0.3070])
```

Figure 7: Output of mean and standard deviation of the new dataset

We will need to update the mean and standard deviation as we have added new images. This is required for data normalisation. Additionally, if the code prints out error on images

that can't be opened, either delete them, or try to fix it by opening it in Paint 3D and resaving it.

Extra information on how the labels are automatically collected
Within each folder in either non sport or sport, there will be many categories, for example the sport folder would have all the categories of sports we collected, and the non sport folder will have all the categories of non sports images we collected

Note that our training code will only automatically get the label (main label of sports and non-sports) via the folder name for the folders inside the dataset folder, and the folders inside non sport and sports.

For example, our code will automatically retrieve the main label of non sport and sport because of the folders inside the dataset folder. The code will also retrieve the sub-labels inside sport and non-sport, for example from the sport folder: Football, Volleyball, Tennis, according to what folder names are there inside the sport folder. It will not recursively find labels inside each individual sub-label.

2.3 Training and saving the model

Throughout our project, we had experimented with different neural network architectures. If you wish to try the other ones, you can choose to use those other files instead. Otherwise, please open train_model_resnet_multilabel.ipynb, as the flask website will be using ResNet50.

Follow the steps below to set up the training:

- Set BATCH_SIZE according to your graphic card. For example, I have a Geforce 1060 6GB, which can only have up to a batch size of 15 for Resnet50, if no other application is using up the VRAM.
- Set PROCESSES according to how many logical processors you have, minus some considering the amount of RAM you have. For example, I have an AMD Ryzen 7 3700x with 8 core 16 threads, and I can choose to put 16 for processes. However, due to RAM limitations, I've set mine to 8. This setting is for how many workers there are going to be to load the dataset.
- Set EPOCHS to how much epoch you want to train. Keep it to 8 or 10 if you want to follow how we conducted our training.

See below on the code to edit:

Init parameters and transforms

```
In [ ]: # set training parameters
        BATCH_SIZE = 15 ## Change batch size as required, if when training there is not enough memory, decrease it,
                        ## Else try to increase it and push it, bigger batch size, less epoch may be required to reach
                        ## desired accuracy/diminishing return point
        PROCESSES = 8 ## Maximum is how much logical processors.
        EPOCHS = 10
```

Figure 8: The codes to edit to change batch size, processes and epoch

- **(IF YOU HAVE ADDED NEW DATA)** Replace the mean and standard deviation values in transform and transform_test's Normalize function to the new mean and standard deviation obtained from 2.2.1 as below:

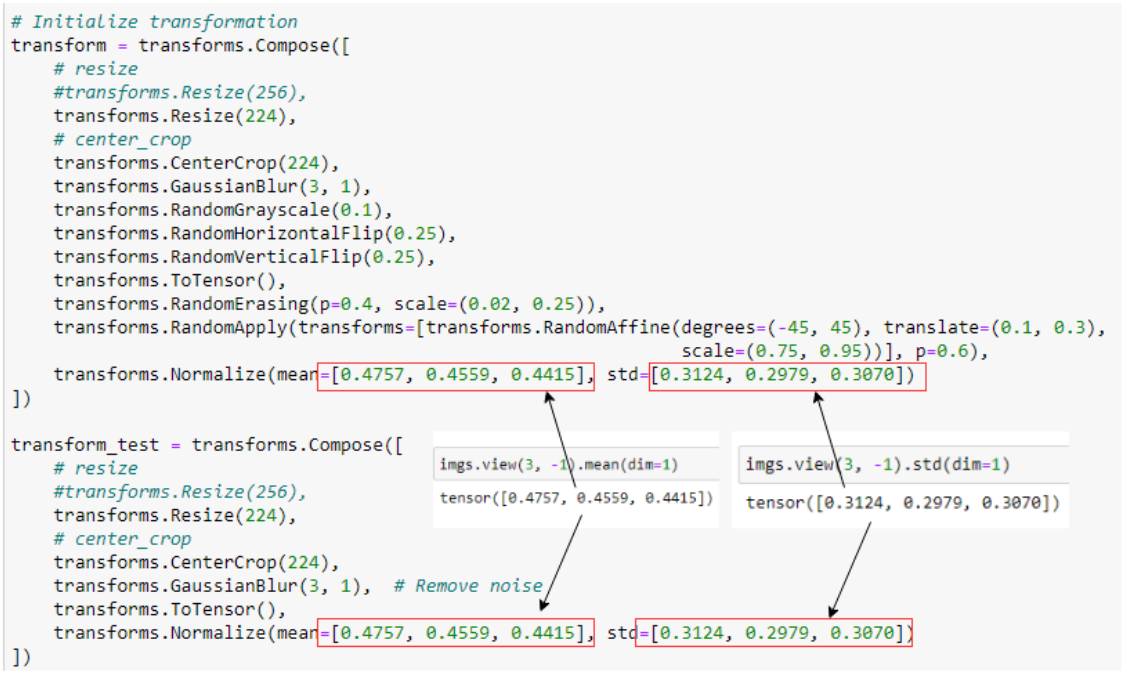

Figure 9: Image outlining how to change the mean and standard deviation

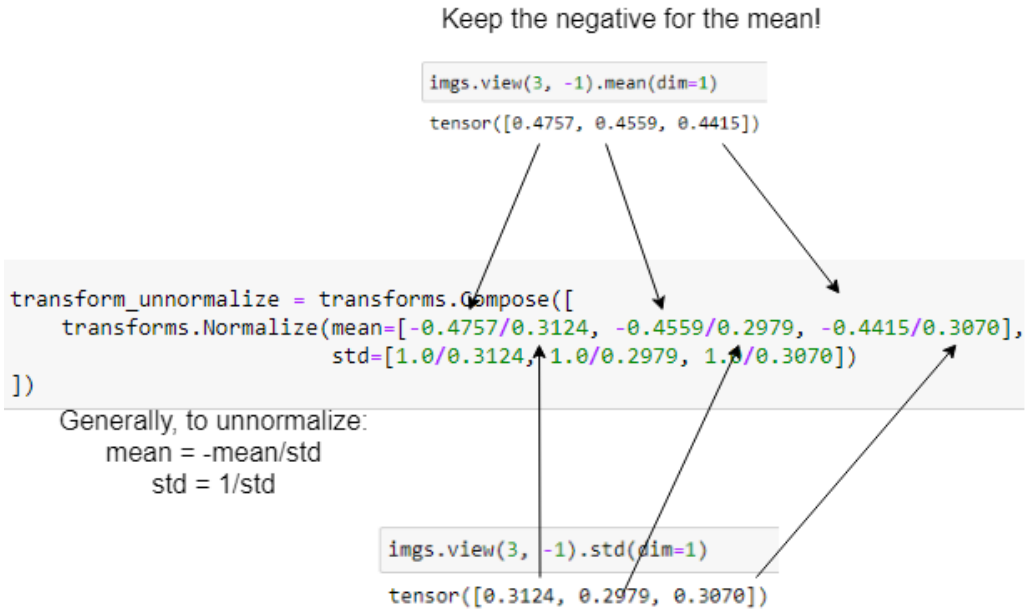

Figure 10: Image outlining how to change the mean and standard deviation

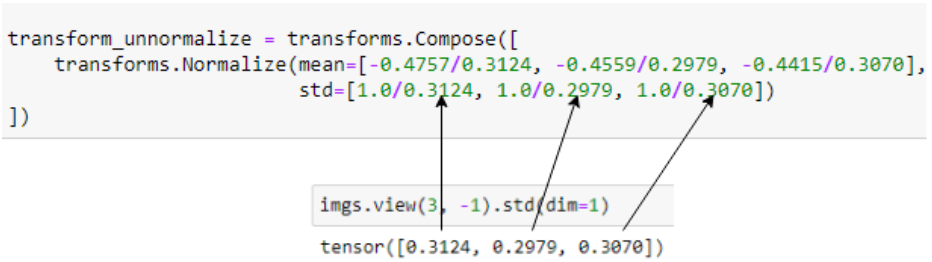

Figure11: Image outlining how to change the mean and standard deviation

- **(If you have added new sub category)** Please also copy and save the following output:

```
In [6]: both_combined

Out[6]: ['Pedestrian (260)',
         'Queue (245)',
         'Reading (202)',
         'cello (267)',
         'dab (89)',
         'dog chasing ball (329) (ambiguous)',
         'driving (247)',
         'guitar (303)',
         'harp (226)',
         'holding ball (266) (ambiguous)',
         'holding something (243)',
         'jumping (273) (ambiguous)',
         'random pic with sky view (62)',
         'sleeping (401)',
         'standing (302)',
         'standing beside bicycle (250) (ambiguous)',
         'using computer (306)',
         'violin (286)',
         'waving (81)',
         'Badminton (679)',
         'Basketball (338)',
         'Cycling (271)',
         'Football (340)',
         'Tennis (510)',
         'Weightlifting (152)',
         'boxing (248)',
         'fencing (124)',
         'race walking (230) (ambiguous)',
         'sky diving (343)',
         'squash (395)',
         'swimming (428)',
         'table tennis (179)',
         'track and field (749)',
         'volleyball (405)']
```

Figure 12: Output of all the sub categories labels.

- Afterwards, Run all the code from the beginning until the Save Model code (inclusive, so we save the model into a file). Run until the code below, including it.

Save Model

```
In [ ]: model_scripted = torch.jit.script(model_ft) # Export to TorchScript
        model_scripted.save('resnet50_sports_non_sports_multilabel.pt') # Save
```

Figure 13: The code to save the model. Run until this, including this.

2.4 Evaluating results of training

- When training the model, the training code will output training loss, training accuracy, validation loss and validation accuracy for each epoch
- The code will also output how long it took to train the model.
- Running these code below after training will show the loss and accuracy for both training and validation on a graph

Visualise training stats

```
In [ ]: plt.style.use("ggplot")

        plt.figure(figsize = (12,5))
        plt.plot(TRAIN_STAT["train_acc"], label="train_acc")
        plt.plot(TRAIN_STAT["val_acc"], label="val_acc")
        plt.title("Training Accuracy on Dataset")
        plt.xlabel("Epoch #")
        plt.ylabel("Accuracy")
        plt.legend(loc="lower left")

        plt.savefig('accuracy.png')

        plt.figure(figsize = (12,5))
        plt.plot(TRAIN_STAT["train_loss"], label="train_loss")
        plt.plot(TRAIN_STAT["val_loss"], label="val_loss")
        plt.title("Training Loss on Dataset")
        plt.xlabel("Epoch #")
        plt.ylabel("Loss")
        plt.legend(loc="lower left")

        plt.savefig('loss.png')
```

Figure 14: The code to visualise training stats

- Running these two blocks of code will output the main label accuracy (sports and non sports) and sub label accuracy (football, tennis, driving, etc) respectively.

```
In [ ]: print("Testing model...\n-----------------------------")
with torch.no_grad():

    model_ft.eval()

    predictions = []
    actual = []
    test_correct = 0

    for i, data in enumerate(test_loader):
        # make predictions and add to list of predictions
        inputs, labels = data[0].to(device), data[1]
        main_label = labels["main_label"]
        outputs = model_ft(inputs)
        output_main_label = outputs["main_label"]
        predictions.extend(output_main_label.argmax(axis=1).cpu().numpy())
        actual.extend(main_label.cpu().numpy())

# print the results of the predictions in the form of a confusion matrix
print(classification_report(np.array(predictions), np.array(actual), target_names=labels_map))
```

```
In [ ]: print("Testing model on sub-category...\n-----------------------------")
with torch.no_grad():

    model_ft.eval()

    predictions = []
    actual = []
    test_correct = 0

    for i, data in enumerate(test_loader):
        # make predictions and add to list of predictions
        inputs, labels = data[0].to(device), data[1]
        main_label = labels["sub_label"]
        outputs = model_ft(inputs)
        output_main_label = outputs["sub_label"]
        predictions.extend(output_main_label.argmax(axis=1).cpu().numpy())
        actual.extend(main_label.cpu().numpy())

# print the results of the predictions in the form of a confusion matrix
print(classification_report(np.array(predictions), np.array(actual), target_names=both_combined))
```

Figure 15: The code to evaluate the model using test data

2.5 Using Captum to interpret the model on own image

- First, edit filename inside Image.Open to the image directory as shown below. In the example below, the image is placed at the project root.

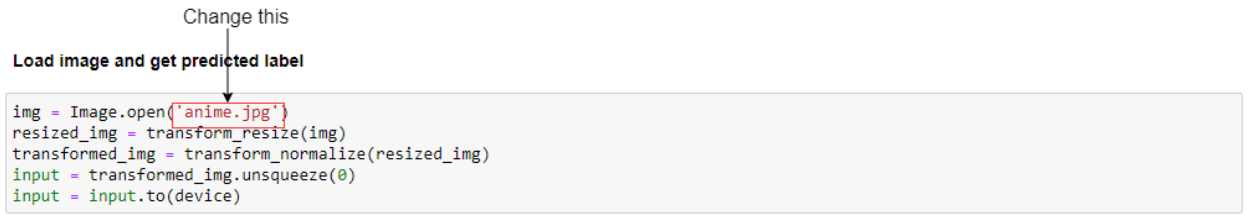

Figure 16: Image showing what to change for Captum

- Secondly, increase or decrease nt_samples as shown below as required based on your GPU specifications. If using CPU only, adjust according to CPU and RAM.

Figure 17: Image showing where to change nt_samples

- Run Starting from “Init values and related stuff” to all the code as shown above. Code from both blocks above will show interpretation based on main and sub label respectively.

3. Launching the Flask Website

- **(If you have trained a new model with new data)** Replace the mean and standard deviation in app.py’s transform variable as follows:

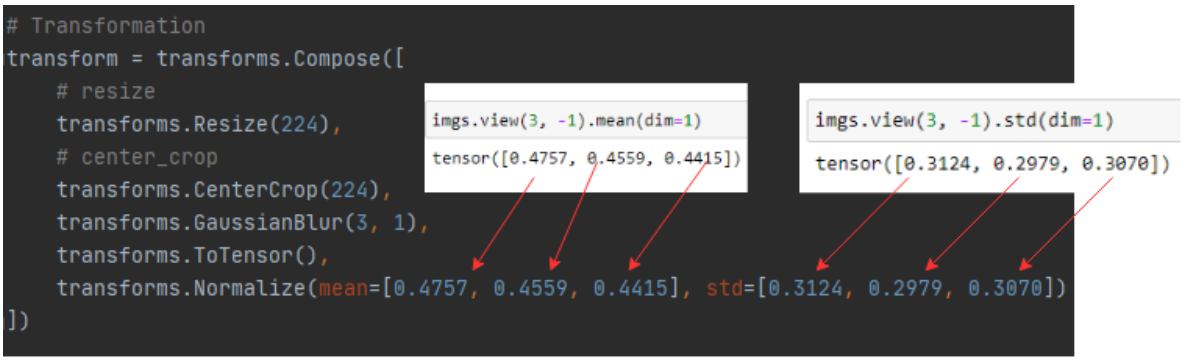

Figure 18: Image showing where to update mean and standard deviation

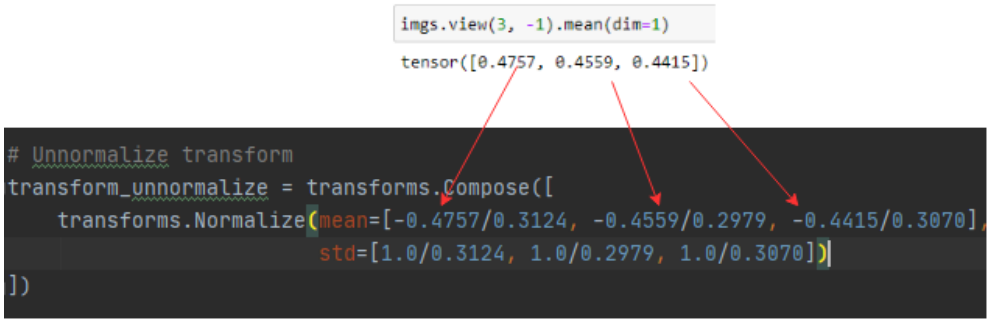

Figure 19: Image showing where to update mean and standard deviation

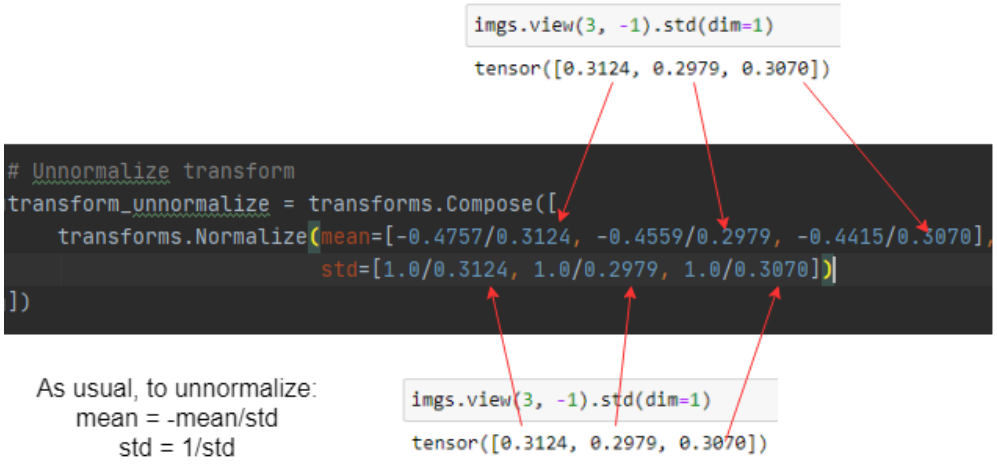

Figure 20: Image showing where to update mean and standard deviation

- **(If you have trained a new model with new data)** Please also copy the list of sub label from 2.3 and paste replace the sub label in app.py as follows:

```
labels_map = {
    0: "Non Sports",
    1: "Sports",
}

sub_label = ['Pedestrian (260)',
             'Queue (245)',
             'Reading (202)',
             'cello (267)',
             'dab (89)',
             'dog chasing ball (329) (ambiguous)',
             'driving (247)',
             'guitar (303)',
             'harp (226)',
             'holding ball (266) (ambiguous)',
             'holding something (243)',
             'jumping (273) (ambiguous)',
             'random pic with sky view (62)',
             'sleeping (401)',
             'standing (302)',
             'standing beside bicycle (250) (ambiguous)',
             'using computer (306)',
             'violin (286)',
             'waving (81)',
             'Badminton (679)',
             'Basketball (338)',
             'Cycling (271)',
             'Football (340)',
             'Tennis (510)',
             'Weightlifting (152)',
             'boxing (248)',
             'fencing (124)',
             'race walking (230) (ambiguous)',
             'sky diving (343)',
             'squash (395)',
             'swimming (428)',
             'table tennis (179)',
             'track and field (749)',
             'volleyball (405)']
```

Figure 21: Image showing where to update the sub labels. Replace the list with the list from Figure 12.

- To launch the Flask website, either run app.py directly, or navigate the anaconda prompt to the project root and enter: python -m flask run
  Note that it can be python3 or py, depending on what anaconda named it as.
- The website should now be launched, and by default the url is http://127.0.0.1:5000/.
  Note that this is not yet packaged for production use, instead it is more towards own

use for research, so if you want to host it on a public server, please research on how to prepare the flask website for production.

4. Commonly faced issues

4.1. Model training

- RuntimeError: CUDA error: out of memory
  This is due to your graphic card being out of VRAM memory. Please make sure no other programs are using VRAM when training. If you encounter this error, please make sure to restart the kernel and clear the output to deallocate VRAM used as shown below

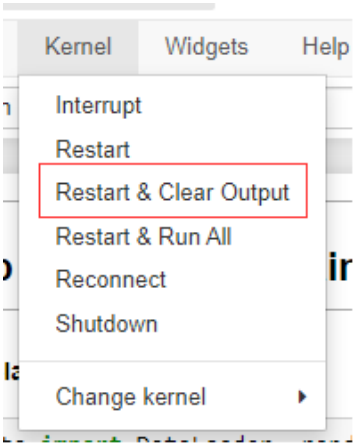

Figure 22: Outlining the step to restart and clear output

- "Caught IndexError in DataLoader worker" or any error relating to worker
  This is because maximum ram usage has been reached (Not VRAM). Lower the amount of workers (NUM_PROCESSES).
- Image error during training
  This could be an image in the dataset being corrupt. Run the check_corrupt_image.ipynb code (only the one that checks for corrupt image) and delete the image accordingly.

4.2. Flask Website

- Could not launch Flask website
  This could be because you were trying to run the app.py on a python environment where Flask isn't installed (which would also imply it has no PyTorch and Cuda libraries for this environment). Select the correct environment and try again.
  This could also happen if you have multiple installations of python, which leads to selecting the wrong python environment to launch the website.
- Unexpected error when predicting an image
  This can be a multitude of things, and it all stems from internal errors, such as uncaught errors during runtime. The error log provided by Python should provide a good enough starting point to debug and fix it.
- Index out of range error
  This can be due to adding new data and forgetting to update the sub label list in app.py. Please follow the steps in 2.3 and 3 where the sub label is updated.