ECE 442 Introduction to Multimedia Signal Processing Winter 2021
Laboratory #5, March 26, 2021

Introduction to Deep Learning based Computer Vision and Edge Computing

===

Objective

In this Lab, you will

1. Implement deep learning based computer vision applications on the Atlas 200DK.
2. Learn to identify the category of an input image using an image classification model.
3. Estimate the viewing direction of a person appearing in a given image, using a head pose estimation model.
4. Learn how to use Python APIs and write code for model loading, inferencing and post-processing.

Requirement

Login to a remotely served Atlas 200 DK using your assigned IP. Please follow the steps outlined in the guide document '**Remote_Login_to_Atlas_200_DK**'.

Special Instructions

1. Please submit in softcopy at eclass by 5pm of April 14th as a single zip file in **FirstName_LastName_lab5.zip**.
2. Questions that should be answered in your code submission are numbered and marked as bold within the following text.
3. Certain questions ask for image results to be uploaded.
4. The submission is expected to contain the following files: your answer sheet, Python code, and input/output files to be reproducible on our side.
5. Make sure your Python code is bug-free and works out of the box. Please be sure to submit all main and helper functions.

1. Procedure: Image Classification

In this workshop, you will implement an image classification app on the Atlas200 DK, which can classify objects in images using the GoogLeNet network and output the top five classes with the highest confidence scores. Detailed steps, and explanations are provided in this guide, so you can understand how to build the app step by step. Figure 1 below shows the building blocks of the application pipeline.

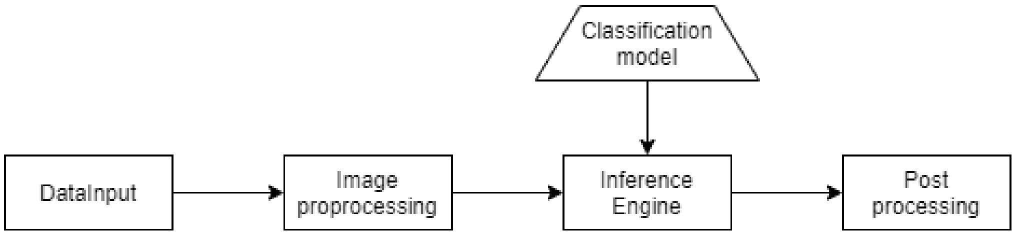


Figure 1. Image Object Classification Network pipeline

The code for this project is available as a GitHub repository. You will first log in to the board, then download the repository to the board and finally run the experiments for the image and video branches of the project step-by-step.

2. Hands-on

1. Download the GitHub repository to the board

Login to the board, run following command. **NOTE:** Please replace 'HwHiAiUser' with your username.

- ➢ *cd /home/HwHiAiUser/HIAI_PROJECTS*
- ➢ *git clone https://github.com/Atlas200dk/sample_image_classification_c73_python.git*

2. Prepare GoogleNet model.

1) Download the 'googlenet.prototxt' and 'googlenet.caffemodel' to the board.

Login to the board, go to any directory, run following command:

- ➢ *wget https://github.com/Ascend-Huawei/models/raw/master/computer_vision/classification/googlenet/googlenet.prototxt*
- ➢ *wget https://obs-model-ascend.obs.cn-east-2.myhuaweicloud.com/googlenet/googlenet.caffemodel*

2) Setup temporary env variable for model conversion by running below command:

```
export LD_LIBRARY_PATH=${install_path}/atc/lib64:/home/HwHiAiUser/Ascend/ascend-toolkit/20.0.RC1/arm64-linux_gcc7.3.0/acllib/lib64
```

3) Run the following command in the same directory as the downloaded model files to convert the model.

```
atc --framework=0 --model="googlenet.prototxt" --weight="googlenet.caffemodel" --input_shape="data:1,3,224,224"  --input_fp16_nodes="data"  --input_format=NCHW --output="googlenet"  --output_type=FP32 --soc_version=Ascend310
```

4) Copy the converted 'googlenet.om' to project directory: sample_image_classification_c73_python/model/

For example, in the remote login scenario, you can run following commands:

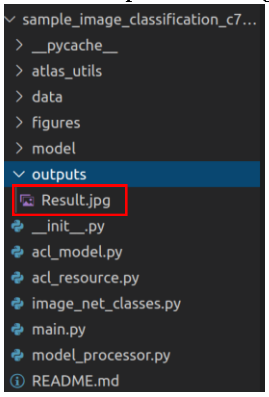
- *mkdir /home/HwHiAiUser/HIAI_PROJECTS/sample_image_classification_c73_python/model/*
- *cp googlenet.om /home/HwHiAiUser/HIAI_PROJECTS/sample_image_classification_c73_python/model/*

5) **Restore the default env variable by running following:**
   source ~/.bashrc

3. **Run the application and view results.**

- *cd /home/HwHiAiUser/HIAI_PROJECTS/sample_image_classification_c73_python/*
- *python3 main.py*

After running the application, you should be able to see the result prediction on the test image.
The result image 'Result.jpg' is in
/home/HwHiAiUser/HIAI_PROJECTS/sample_image_classification_c73_python/outputs/
You can open the image by click on it from VScode file explorer.



```
sample_image_classification_c7...
  __pycache__
  atlas_utils
  data
  figures
  model
  outputs
    Result.jpg
  __init__.py
  acl_model.py
  acl_resource.py
  image_net_classes.py
  main.py
  model_processor.py
  README.md
```

Question 1: Submit this result image, along with the category prediction (class and confidence) (20 points).

3. Procedure: Head Pose Estimation

In this experiment, you will implement head pose estimation application on the Atlas200 DK. This application can be useful in several scenarios; one such scenario is determining the viewing direction of a driver in an automated driver assistance system installed in a vehicle. If the driver is found to be looking away (up/down/left/right) from the optimal viewing direction (straight ahead) for an extended period of time, the system can generate an alert to bring the driver's attention back, and mitigate risky behaviors. This method can also be used in other applications such as online exam monitoring.

This method first detects a face region located in the given input image and then uses the cropped region of the detected face as the input to the head pose estimation model. The head pose of a person is calculated in terms of 3 angles: yaw, pitch and roll in an image (https://simple.wikipedia.org/wiki/Pitch,_yaw,_and_roll). Detailed steps, and explanations are provided in this guide, so you can understand how to build the app step by step. Figure 1 below shows the building blocks of the application pipeline.

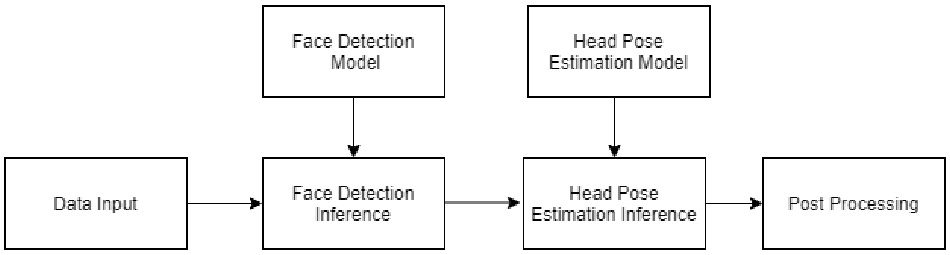


Figure 2. Head Pose Estimation Pipeline

The code for this project is available as a GitHub repository. You will first log in to the board, then download the repository to the board, complete the script as per instructions and finally run the experiments for the input image step-by-step.

4. Hands-on

1) Download the repository by running the following commands:
   - ➢ *cd /home/HwHiAiUser/HIAI_PROJECTS*
   - ➢ *git clone https://github.com/Ascend-Huawei/experiment.git*

2) Models:

Pre-trained models are provided for 1. face detection 2. head pose estimation. Please follow the instructions below to convert the models to 'Offline Model' for inference on the board.

a) Download the pre-trained Face Detection (Caffe) model network and weights to the project directory 'head_pose'.

   - ➢ *cd /home/HwHiAiUser/HIAI_PROJECTS/experiment/head_pose/*

   - ➢ *wget https://c7xcode.obs.cn-north-4.myhuaweicloud.com/models/face_detection/face_detection.caffemodel*

   - ➢ *wget https://c7xcode.obs.cn-north-4.myhuaweicloud.com/models/face_detection/face_detection.prototxt*

   Then, execute the following command from the same project directory 'head_pose' to convert the model:

   - ➢ *export LD_LIBRARY_PATH=${install_path}/atc/lib64:/home/HwHiAiUser/Ascend/ascend-toolkit/20.0.RC1/arm64-linux_gcc7.3.0/acllib/lib64*

   - ➢ *atc --framework=0 --model="face_detection.prototxt" --input_shape="data:1,3,300,300" --weight="face_detection.caffemodel" --input_format=NCHW --output="face_detection" --output_type=FP32 --soc_version=Ascend310*

b) Download the pre-trained Head Pose Estimation (Caffe) model network and weights to your project directory 'head_pose'.
   - ➢ *cd /home/HwHiAiUser/HIAI_PROJECTS/experiment/head_pose/*

   - ➢ *wget https://obs-model-ascend.obs.cn-east-2.myhuaweicloud.com/head_pose_estimation/head_pose_estimation.caffemodel*

- ➢ *wget https://raw.githubusercontent.com/Ascend-Huawei/models/master/computer_vision/object_detect/head_pose_estimation/head_pose_estimation.prototxt*

Then, execute the following command from your project directory 'head_pose' to convert the pre-trained model for head pose estimation to offline model (.om) format:

- ➢ *export LD_LIBRARY_PATH=${install_path}/atc/lib64:/home/HwHiAiUser/Ascend/ascend-toolkit/20.0.RC1/arm64-linux_gcc7.3.0/acllib/lib64*

- ➢ *atc --framework=0 --model="head_pose_estimation.prototxt" --weight="head_pose_estimation.caffemodel" --input_shape="data:1,3,224,224" --input_format=NCHW --output="head_pose_estimation" --output_type=FP32 --soc_version=Ascend310*

c) Restore the default env variable by running following:
source ~/.bashrc

NOTE: If the model conversion process fails, please download the offline model directly from the following link: Google Drive.

3) Complete the lines of code in the script *'head_pose_estimation.py'.*
Fill in the missing lines of code, according to the instruction provided in comments. Insert your code wherever you find the comment:
### Your code here … ###

Question 2: Finish Codes to Initialize ACL and ACL Runtime (10 points).

Question 3: Finish Codes to Load the model for face detection and head pose estimation (20 points).

Question 4: Finish Codes to Performing inference: call the *execute()* method for face detection inference (10 points).

Question 5: Finish Codes to Performing inference: call the *execute()* method for head pose inference (10 points).

[**HINT** You may refer to the sample code in the image classification project, to learn how to use the required APIs in the code: https://github.com/Atlas200dk/sample_image_classification_c73_python]

Question 6: Based on the estimated (pitch, yaw, roll) angles, determine the viewing direction (30 points).

The viewing direction, or head pose, is determined from the values of any or a combination of the three angles: yaw, pitch and roll, predicted by the head pose estimation model. The following are the types of poses, or viewing directions, and the angle values they are based on:

'Straight Ahead': pitch, yaw and roll angles
'Up' or 'Down': pitch angle

‘Left’ or ‘Right’: yaw angle
‘Swing Left’ or ‘Swing Right’: roll angle

Your task in this question, is to complete the code blocks in function ‘*head_status_get()*’ by assigning the viewing direction (head pose) based on the estimated ranges of angle values. As an example, codes for determining the viewing direction of ‘Up’ or ‘Down’ has been provided in the function ‘*head_status_get()*’. Similarly, complete the code for viewing directions of ‘Straight Ahead’, ‘Left or Right’, ‘Swing Left or Right’, following the instructions provided in comments in the function.

4) Run the Application

To run the application, open up a terminal on the board (VScode->Terminal) and navigate to the project folder and run the python script:

- *cd /home/HwHiAiUser/HIAI_PROJECTS/experiment/head_pose*
- *python3 head_pose_estimation.py*

Expected outputs printed to terminal (sample):

```
Head angles: [9.7900390625, 1.219940185546875, -1.7639160015625]
Pose: Viewing direction: Straight ahead
```

You can also see the 64 detected face keypoints plotted on the given image, saved to folder ‘*/home/HwHiAiUser/HIAI_PROJECTS/experiment/head_pose/out*’. Please run the application for the image files ‘**head_bend.jpeg**’, ‘**head_tilt.jpeg**’ and ‘**head_upward.jpeg**’ in your project folder ‘*/home/HwHiAiUser/HIAI_PROJECTS/experiment/head_pose/*’, and **include the result images in your submission** (right-click on the file in VScode to download it). Compare the head pose outputs obtained for these images and **provide the predicted output pose as printed to the terminal** in your submission.

Bonus (20 points)

Can you catch a distracted driver? Assume that you are designing a system that raises an alert if the camera of your driver assistance system detects the driver looking away (sideways or down) for more than a reasonable amount of time, say 3 seconds. Write some additional code to design this functionality. You may print a message ‘ALERT’ to terminal, when your algorithm detects “distracted” behavior. Please use the provided test videos for demo. **Please submit the processed video (10 points) and code (10 points)**.

Clue:

1. Refer to **Appendix** for how to read a video file.
2. If the video is 30 frames/second, then 3 seconds mean 90 frames.

Appendix

Coding Guide

1 Reading and Writing Videos with Python and OpenCV (more info):

In the following code example, a video is read from a file ‘test.mp4’ using OpenCV’s VideoCapture class. The video is read frame-by-frame, in a ‘while’ loop, until the end of the video is reached. Each frame undergoes some processing and the processed frame is written to an output file ‘out.mp4’, using the write() method of OpenCV VideoWriter class.

```
import numpy as np
import cv2

# Define the codec and create VideoWriter object
out = cv2.VideoWriter(‘out.mp4’, 0x7634706d, 25, (1280, 720))
cap = cv2.VideoCapture(‘test.mp4’)
while(cap.isOpened()):
    ret, frame = cap.read()
    processed_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # write the processed frame to file
    out.write(processed_frame)

cap.release()
out.release()
```

You may refer to this code sample: https://github.com/Atlas200dk/sample_bodypose/tree/master/code_video to better understand how a video is read and processed using OpenCV and Python on the Atlas DK.

2 For help on using Python compound statements, please refer to the following webpage: https://docs.python.org/3/reference/compound_stmts.html

3 For help with basic Linux commands, such as listing and moving files, please refer to the following: https://www.linuxtrainingacademy.com/linux-commands-cheat-sheet/

4 Debugging in Python:

The typical usage to break into the debugger from a running program is to insert breakpoints (more info).

```
import pdb
pdb.set_trace()
```