**Age Prediction from facial imagery:**

First hour was spent understanding the assignment (inputs/outputs, deliverables etc.), setting up an anaconda environment and doing research into the topic and any past approaches/papers for lessons learned and potential approaches to explore and challenges faced.

I then created a GitHub repository and defined the folder structure within the repo.

**Folder structure:**

**1. Project Root:** This is the main folder that contains your entire machine learning project.

**2. Data:** This folder is dedicated to storing your datasets and any relevant data files. It can be further divided into subfolders such as:

* *Raw:* Contains the original, unprocessed data files.
* *Processed:* Contains preprocessed data that has undergone cleaning, transformation, and feature engineering.

**3. Notebooks:** This folder is for Jupyter notebooks or any other interactive notebooks you use for experimentation, analysis, and model development.

**4. Scripts:** This folder contains reusable code scripts or modules that you use in your project. It may include:

* *Preprocessing:* Scripts for data cleaning, transformation, and feature engineering.
* *Utilities:* General-purpose utility scripts or helper functions.
* Not applied during project, but if time allowed, functions from notebooks would be transferred to the following subfolders as .py scripts:
  + - *Model:* Scripts for defining and training machine learning models.
    - *Evaluation:* Scripts for model evaluation, metrics calculation, and validation.

**Models:** This folder is dedicated to storing trained models or model checkpoints. It can be further organized into subfolders based on different experiments, versions, or architectures.

**Documentation:** Include any project-related documentation, such as README files, data dictionaries, or project specifications.

My inspiration was partly inspired by this article. The above folder structure text is also extracted from this article. My intention if continued developing would be to follow this structure (Source: https://dev.to/luxacademy/generic-folder-structure-for-your-machine-learning-projects-4coe)

**1.0: Data Analysis:**

* + There are plenty of ways to check for image quality. Use of a preexisting deep learning or image based model was considered. Particularly for face detection to filter out imagery so that there are only single instances of faces in imagery.
    - Classic Image Quality Metrics:
      * Bluriness detection: Use the Laplacian variance method to detect blurry images (threshold based)
      * Noise Detection: Use metrics like Peak Signal-to-Noise Ratio (PSNR) or Structural Similarity index (SSIM) to detect noisy images or similar.
      * Brightness and Contrast: Analyze the histogram of pixel values to detect images that are too dark or bright
    - Image Content Analysis:
      * Entropy: Calculate the entropy of an image to detect low-information content.
      * Edge detection: Use of algorithms such as Canny Edge detection to identify images with few or no edges
    - Metadata Analysis:
      * Resolution check: Filtering images by resolution. However, since this is consistent in the dataset, this is not valid.
      * File size: Very small file sizes might indicate low-quality images
  + We won’t be exploring image content (analysis-wise) as a result of time constraints and the concern of being overly aggressive in the initial data cleaning. For future, this is definitely a consideration to explore. In addition, Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity index measure (SSIM) won’t be explored due to limited time. These also require a reference image (a “clean” image) in order to work. Deciding on a good reference image is also a purely subjective decision which could take some time to decide on a good “clean” example to use.
  + It was found through a quick manual check of the data that various examples such as cartoons and even a picture of just a guitar was present in the dataset. This led me to think I should be more aggressive in data cleaning compared to my previous attempt. Due to this, I looked into face detection solutions to filter the imagery. Using solutions such as HOG + Linear SVM was considered and so was Haar Cascade for face recognition. HOG + Linear SVM is great for front facing faces, however, lacks performance when the orientation of faces present differed (<https://debuggercafe.com/face-detection-with-dlib-using-cnn/> ). The Haar Cascade is known to be pretty good but the “pretrained” files are only useful for specific cases, i.e. full face frontal, face profile (left side of face – scope can be widened by flipping the image), eye detection etc. While a combination of these were considered, after a quick search the “dlib” library apparently has a great cnn based detector capable of “detecting faces from varying viewing angles, lighting conditions, and occlusion”. (<https://pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/> ). Overall, there are a lot of face detection algorithms, it was not until I came across this article (<https://learnopencv.com/what-is-face-detection-the-ultimate-guide/> ) that I came to a decision on a face detector to use. I ended up using YuNet due to its great performance in the reported tests. Originally, I went with RetinaFace-Resnet however, the original library containing this model would respond with an HTTP 410 Gone error. After exploring and failing other ways to use this model, I opted for the next best approach, YuNet. That said, if I had more time I would’ve explored this further and also tested multiple then chose the better performing one.
    - The threshold was lowered with YuNet, as the original threshold of 0.9 filtered out a lot of faces that were pointed away from the camera. Lowering the value worked great, as the rejected examples included a lot of examples with missing faces or being drawings/cartoons themselves.
  + It was noted that one of the examples, there was a failure to extract the std. deviation from example “…\\dataset\\48\\43.jpg”. This was because the image shape was [1, 128, 128] instead of [3, 128, 128]. This indicated that the image was being read as grayscale by default instead of RGB.
    - Solution: Excluded from dataset as there was only a single example

**Data Cleaning**

* + The question to use contrast as a measure for image quality was considered highly as the lowest contrast bin (after plotting on a histogram) contained multiple faces within 2 of the 5 images. Surely the presence of multiple faces could influence the models outputs?
  + It was considered that other external datasets could be used to expand the dataset. There are multiple datasets of faces around but it would be difficult to ascertain whether there are duplicates if I introduced the external data in (unless I used SSIM, but time is of the essence) This could be explored in the future.
  + Resizing (interpolation vs. zero-padding): Since the images are smaller than some of the input sizes for pretrained models (if considered later on), there are two strategies that stand out to match the imagery with the input size. Since interpolation can distort some of the patterns in the imagery and the imagery itself, we will opt for zero-padding. In addition, zero-padding reduces training time and doesn’t affect classification (according to this paper: <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0263-7#Abs1> )
  + It was identified in the imagery that there are drawings and animations present in the dataset. Use of a face detector helped in filtering these out as well as imagery with no faces at all. While confidence was raised in filtering this out, it was not entirely sure if all instances were removed. Fine-tuning the detectors thresholds may help in the future.

**Model Development**

* + Age prediction appears to be a regression problem at first-hand view of the problem. However, after researching online, there have been challenges with developing a good model and the typical approach appears to group ages and perform classification instead. This is a potential approach. My thought with this is that if I proceed with an age-group classification, I could either bootstrap some custom “regression head” to further refine the result (if possible and feasible). Another idea was that, I could use another ml model (tree-based, cluster etc.) from the output of this model, to do the refinement. While challenging, I opted to attempt regression-based model as a first pass. Future improvements may explore the forementioned classification-based approaches.
  + Pruning and quantization are great techniques for reducing the model size to optimize for mobile. Originally, I considered using a larger pretrained model like VGG16 or ResNet50, and would train the model using L1 regularization (with hope that some parameters are reduced to 0), and then try to prune the model to further reduce the model size.
  + MobileNet, EfficientNet and SqueezeNet are classic lightweight pretrained architectures optimized for mobile and embedded devices. These will be my starting points.
  + Since I am using Windows 11 (windows native), I will be using the PyTorch framework as it is much easier to use the GPU than TensorFlow (2.10 is the latest version that supports windows native). I considered using wsl or a vm (oracle virtualbox) but at this stage I didn’t want to risk the time spent for setup. My preferred framework (due to experience and ease of use in my opinion) is Tensorflow and tensorflow lite for mobile, however, I have had some issues installing the tensorflow-gpu library in a previous personal windows native machine before and chose not to risk it. In addition, version 2.10 has limited functionality compared to the latest versions.
  + I will use MLFLOW for tracking experiments
  + Model selection process: K-fold cross validation to determine which model generalizes better with the current dataset
  + Hyperparameter tuning? Time consuming?

**Training and Evaluation:**

* + Don’t forget logging and experimentation
  + Due to time running out, my initial K-Fold cross validation was going to be set for 20 epochs, 5 folds and 3 different models. Since time was short, so I reduced this to 10 epochs, 4 folds and 2 models.
  + From the cross-validation results, it was seen that the two models (efficientnetB0 and mobilenet\_v3\_small) had very similar performances. Having similar MAE, MSE, RMSE and R^2 values, while efficientnetb0 appeared to be the better model, the decision actually came to using mobilenet\_v3\_small. This was because not only did it have a similar performance, but it also has just under half the number of parameters compared to efficientb0 (1.5 million instead of 4 million). As I need to optimize for mobile deployment, I opted for mobilenet.
  + Since pruning can be detrimental to the models performance my thought was that maybe I should follow this task sequence:
    - Apply L1 reg. for training model (encourage sparsity and feature selection via weight reduction to zero)
    - Prune model to reduce size
    - Fine-tune without L1 reg. to recover some of the performance loss from pruning
  + Since I ran out of time, I couldn’t explore hyperparameter tuning. This is how I would go about hyperparameter tuning
    - Applying a mixture of manual experimentation and using automated methods such as Grid search to search for the optimal parameters.
    - Hyperparameters I would start with and then explore others: Batch size, learning rate, number of trainable layers to start.
  + I had desired to setup mlflow and installed the packages in the anaconda environment for tracking the ml experiments. However, I ran out of time to set this up and decided to proceed with training and manually tracked the experiments. Since there was very few, this was very manageable.

**Model Packaging for C++ Deployment**

* + Since the model is already exported as a TorchScript module, this would need to be passed to the C++ developer. (Filename: “mobilenet\_v3\_small\_regression\_l1\_mobile.pt”)
  + In terms of the steps involved with handing over the model, I would follow the following steps:

1. Save the trained model (as TorchScript)
   1. Fortunately, this has been done at the end of the 2.1-Model-Training.ipynb notebook. Although, for simplicity, I would write a specific python script to take the trained model and then serialize it as a TorchScript module
2. Provide a C++ integration script (i.e., run\_model.cpp)
   1. Loads and runs the model
3. Create a directory to package the model and scripts
   1. Since the developer will need to know how to preprocess the input, I would include the file “…/src/preprocessing/preprocessing.py” which handles preprocessing the input
   2. Run\_model.cpp
   3. CMakeLists.txt
   4. README.md
   5. mobilenet\_v3\_small\_regression\_l1\_mobile.pt (Traced PyTorch model for age prediction)
4. Create a CMake Configuration file for building the project (CMakeLists.txt)
5. Create a README for instructions (README.md)
   1. Contain a list of Prerequisites:
      1. “LibTorch”: C++ distribution of PyTorch
      2. “C++ Compiler”: A compatible C++ compiler (such as GCC, MSVC, etc.)
   2. Instructions detailed (Rough outline below)
      1. Navigate to project directory
      2. Create a build directory
      3. Run CMake to configure the project
      4. Build the project
      5. Execute the binary
6. Package the directory in a .zip and hand over to the developer

Note for mobile apps: Use PyTorch Mobile

* Android: Use the torchvision library with the Android NDK.
* iOS: Use the PyTorch C++ API and integrate it into your iOS app.

**Future Work:**

* + There is no recorded data on the genders, race, facial hair or the presence of accessories on the examples face within the dataset. Without this information, we don’t know what bias could be present in the dataset.
  + Determination of Laplacian threshold could’ve used more time. My impression is that my selected threshold may not be aggressive enough to filter the blurry imagery. Since this is a subjective evaluation, it was difficult to determine a quick, robust threshold. In the future, fine-tuning to find the critical threshold would help.
  + If I had opted for a classification model for age range instead of individual ages through regression, I would’ve opted for the F1-Score metric for evaluation, as not only is it the harmonic mean between precision and recall but (something about class imbalance)