**Age Prediction from Facial Imagery Report:**

This document contains a summary of the results and procedure of the Age Prediction project. The task of the project was to develop a deep learning model to predict age from facial images. The following procedure was followed in the development of the deep learning model:

1. Data Analysis: Refer to 1.0-Data-Analysis.ipynb for the full analysis
2. Data cleaning and dataset creation: Refer to 1.1-Data-Cleaning-and-Dataset-Creation.ipynb
3. Model Development: Refer to 2.0-Model-Selection.ipynb
4. Training and Evaluation: Refer to 2.1-Model-Training.ipynb

**Dataset**

The raw dataset provided contains nearly 30,000 facial images, ranging from 20-50 years. In terms of the dataset cleaning and creation, it was found from the data analysis that various low-quality examples were present. Examples either contained blur, low brightness, or content missing faces or resembled drawings/sketches/animations. In short, the following procedure was used to clean the data and produce our deep learning dataset.

1. Removing blurry imagery: Using the variance of the Laplacian of an image, we could determine the presence of blur (low values = blur, high values = no blur)
2. Removing overly dark imagery: By analyzing the histogram of each image, we could determine where the majority each pixel intensity sat on the scale. If a large majority of the pixel intensities sat on the low end of the pixel intensity scale, the result can be very dark imagery
3. Removing images without faces: Using a face detector (YuNet), we could see if there are any faces present in the imagery. For examples where it failed (because it was a drawing or didn’t have a face present), the detector would fail to find anything and we could reject these examples.
4. Removing default grayscale imagery: There was one example that would require additional steps to convert to a RGB image. For simplicity and mitigation of future problems, this example was removed from the final dataset.

**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 an effective model for this task, with the typical approach appearing to group ages and perform classification instead. While this a potential approach, my initial thoughts were to build upon this idea and refine the classification with some bootstrapped regression head or output to another machine learning (i.e. tree based or even cluster based dependent on the output features). My concern with this centred around the model being optimized for mobile, and therefore, I opted to explore a regression-based model as a first pass.

While it seemed feasible to train a custom model from scratch, given the number of images, classes present in the dataset and the data type itself, it was desired to pursue transfer learning where we can take advantage of previously trained information in preexisting models. After a quick search online, it was found that three candidates stand out in terms of lightweight architectures and/or optimized for mobile deployment. These were efficientnetb0, mobilenet\_v3\_small and squeezenet.

In order to narrow done model selection, from the candidates identified, K-Fold Cross Validation was performed to determine the better performing candidates. However, due to the time-consuming nature and limited time left, I narrowed down the models (efficientnetb0 and mobilenet\_v3\_small) and reduced some of the parameters used for validation (i.e. number of folds and epochs) to speed the process. While this may affect the performance, I needed some results to build upon.

At this point, it was decided that pruning would happen after the training, in hope to get the most out of the original architecture. While it was eventually seen that I failed to get a working pruning function to reduce the parameters in my trained model, the approach of pruning before training came into focus. While the thought of pruning before training would ensure that I would have a model that would be less than 1 million parameters, it still wouldn’t be possible without a working model pruning function.

Quantization was also a consideration but after initial failed attempts and running out of time, this was not applied.

**Training and Evaluation**

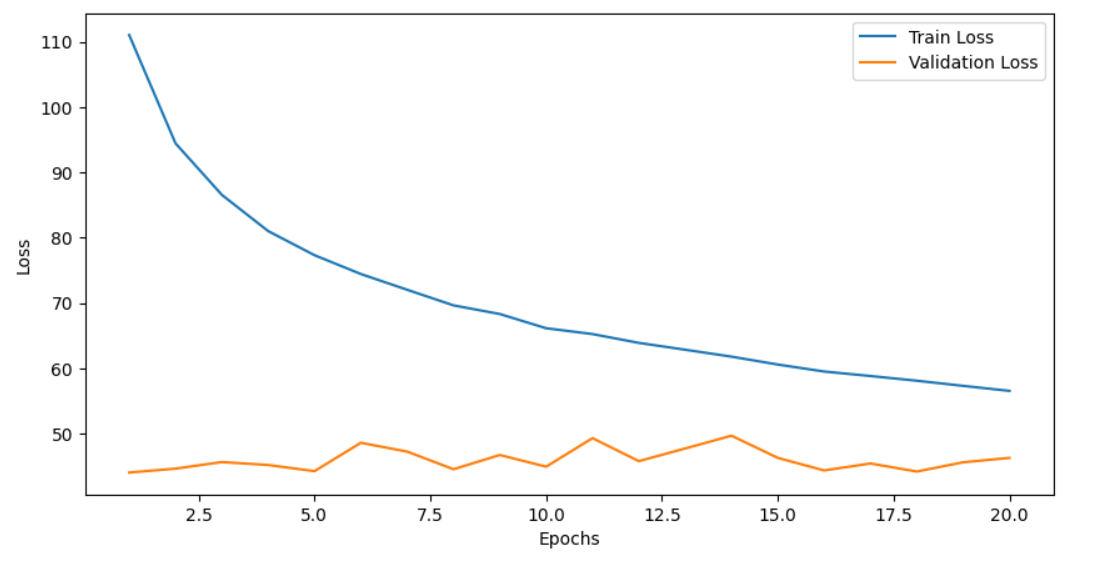
For training, the first set of training was done for K-Fold Cross-Validation. The following parameters and setup was used:

* 2 Models assessed: efficientnetb0 and mobilenet\_v3\_small model (both available from torchvision.models)
* Mean Square Error Loss function (since we are training a regression model)
* 4 Folds (Originally 5 since it is closer to dataset train-val split, but changed due to time)
* Adam (optimizer)
* Learning rate: 0.001
* Batch size: 32

Four metrics were recorded at the same time during training; Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and R-Squared value (R^2). This was done to view the landscape of the metrics over the course of training the folds and models. For a full run down, refer to the 2.0.-Model-Selection.ipynb notebook. The result of this showed that both models performed fairly similarly with efficientnetb0 performing slightly better in all metrics (excluding one fold, where the r-squared value was negative). This could be expected as efficientnetb0 has over twice the number of parameters as mobilenet does (4 million vs 1.5 million). Since the performance is very close, it was chosen to use the smaller model, mobilenet, as less pruning would be required to meet the requirement of less than 1 million parameters.

Since R-squared doesn’t perform well with non-linear data and MSE and RMSE are sensitive to anomalies in the data, the Mean Absolute Error was selected as the main evaluation metric. In addition to this, MAE is intuitive and easy to understand, as it directly represents the average magnitude of errors in the predicted ages. Unfortunately, when it came to training the model,

While I had used the MAE metric in the previous notebook for cross-validation of the two pretrained models, I had failed to remember to add it when training the mobilenet model and ran out of time when I had discovered this (after training and the jupyter kernel had restarted). However, we do have the results of the loss function below:



What we can see from this graph is that while the model is gradually fitting well with the training set, it has not improved on the validation set. This indicates that the current model does not generalize well and tis underfitting on the validation data, as the validation loss doesn’t decrease. In addition, there is a significant gap between the training and validation loss, indicating the model might be overfitting with the training data.

**Model Packaging for C++ Deployment**

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:

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

**Future Improvements:**

* Revision of the data cleaning metrics. For increased confidence, fine-tuning the blur threshold may improve the dataset quality.
* Explore an age group classification solution (briefly mentioned in Model Development) and then explore refinement from the output. May produce better results than the current regression approach.
  + In addition, exploration of other pretrained models and potentially custom models may provide more accurate solutions.
* Get a working prune\_model function going and evaluate the performance of pruning before and after training. Determine whether before or pruning after (alongside fine-tuning) provides a more accurate model with consideration to inference time.
* Hyperparameter tuning to optimize the model’s training. Variables of immediate focus include: Batch size, learning rate, number of trainable layers, etc.
* To improve the model (as seen from the training and validation loss), the following strategies could be explored to prevent overfitting and improve generalization
  + Regularization, early stopping, more data, more data augmentation, hyperparameter tuning, cross-validation, etc.
* Overall, this a far from perfect solution. There are many improvements and methods that could be explored to improve this solution. Now with the foundations set, this solution could be developed much further.

**Learning Lessons:**

Upon reflection, I realize the cost of being too thorough in my data analysis and cleaning caused a lot of time to be taken for this assignment. While the strategies I used were helpful in filtering the dataset, I should’ve been more mindful of the time and scale of the whole project. The lesson learned here was time management and knowing when to move to the next task.