**Mission Sheet for KF5012**

Please create one of these sheets **for each mission you have done,** and please include these sheets **in your project submission**.

The point of this form is to give you a chance to direct us to the work you have done, so that marks can be given fairly, and without missing any of your work. We want you to draw attention to which work meets the criteria for the specified elements in the mission brief. Also, this gives you a chance to draw attention to any work you have done that goes beyond expectations

|  |  |
| --- | --- |
| Mission name | Baseline Implementation |
| Team name | KeepingUpAppearences |
| Student responsible for mission (it team write “team”) | Dawid Michniuk |
| For team missions: should this mission be marked? | N/A |
| In the project brief, the details for this mission outlined several **elements** that should be created for the mission. Please identify clearly which work meets which elements. | * **Baseline Pipeline** – Dataset preparation takes place in main.py (normalizing the data takes place after splitting it into training and testing subsets) with the added functions from utils.py. When going through the colorized.py file, dataset is normalised using prepare\_accuracy\_visualisation\_images() function. * **Baseline Solution** – the current implementation of the baseline can colorize the pictures it is given with loss equal to around 0.009 and 0.01 when using SGD optimizer. The pictures are overall still grey, but the colours given by the neural network are correct and not overwhelmingly one-sided. * **Baseline Evaluation** – a loss value is given while training. Furthermore, when the learning loop is over, a small portion of the dataset that was not in the training loop is used to evaluate how good the model can perform. As previously mentioned, the model is able to colorize the pictures with a very low loss value, but the actual pictures are still predominantly grey, with splashes of colors around skin, cloths, walls, etc. * **Environment Setup** – README.md specifies what needs to be installed in order for the program to run as well as giving a small instruction on how to install them. To add to this, requirements.txt file is located inside the repository with the same modules, to allow for quick installation. |
| If you have work that you think means you have **gone beyond expectations** for the mission, please give details here. Note: this includes sensible work you have done to compensate for team members dropping out or team missions not being completed. | * I have successfully applied decaying learning rate, speeding up the learning process to mere minutes from hours. * To improve quality of our baseline, I managed to split it into different files (main.py, model.py, utils.py and colorize.py), giving freedom to the user, whether they wanted to use specific functions from utils.py, train the model using main.py, change the model in model.py or colorize some black & white pictures using a trained model with colorize.py. * I managed to implement ImageDataGenerator, increasing the number of pictures the model can train on. * The model can be saved and loaded as users see fit. * I did extensive checks on different steps\_per\_epochs, epochs, batch\_sizes, training/testing percentage split, optimizers, loss functions and decaying learning rate options to ensure the best possible quality in lowest required time and computational resources. * The earlier parts of the baseline were implemented in a jupyter notebook format, allowing for testing everything around the model and learning parts of the baseline. After that has been done, I moved the program into python files so that they can be ran in command line and allowing for easier importing across different files. Should there be a need to test the neural network cell-by-cell, beta version of the baseline is available. * I tested the baseline extensively across 2 machines, each with dual-boot option (win10 + Manjaro, win10 + Endevour OS), allowing to remove problematic code (example: tensorboards) and locate potentially code-breaking version issues (models saved in one version of Keras/Tensorflow will not load in different version of same modules when using input\_layer on the neural network) * Code is commented well, explaining each step. * If iterative mission was to happen, the code is split in a way that enables rapid development, where changing one part of the program shouldn’t break the other parts, while allowing the code to be easy to understand. * Code is structured so that the data preprocessing part is clearly differentiated from training part, meaning it is easy to know what code will change the learning loop of the model. The model itself is located in a separate file, to ensure readability. In other words, learning parts of the program are encapsulated well in current baseline implementation. Should a cell-like structure be required, beta and alpha version of the baseline are available. |
| Any other **notes** you want to make about this mission. | * After producing the baseline in Keras, my main task was to reproduce it in PyTorch. I struggled with this mission for months, before giving up and sticking with Keras. Remaining code can be seen in branch called “new”. * Every detail needed to understand and run the baseline is included in README.md of the repository, including explanation of (and links to) different versions of the baseline available in the repository. * Some of the interesting features were cut since due to issues with the baseline, they had to be moved to iterative development, which was dropped due to COVID-19. Examples of that would be argument parsing, auto-encoders (GANs), applying dropout, early stopping, proper implementation of callback and tensorboard. * I was unable to set up CUDA correctly, meaning that all the training of the model had to be done on my CPU. * Speaking of models, after saving a bunch of copies of models that were able to run for anywhere from 6 to 10 hours, I learned that due to issues with Tensorflow’s and/or Keras’s versions and how it handles input\_layer of neural networks all of the models were unable to be loaded back in. This means that I had to retrain the model in the last minute, dropping the prediction’s quality but allowing to test different optimizers as well as applying decaying learning rate. |