Individual Report: Face Recognition Model Training

By Mikael Folkesson, with Asajad Hussein and Isak Andersson

Contents

1. Introduction

This report provides an overview of a short group project (1 week) @ITHS DNN-course, focusing on developing a deep neural network model to recognize facial landmarks using the AFLW dataset (https://www.tugraz.at/index.php?id=17757), consisting of 26k faces with up to 21 annotated landmarks per image. The objective was to evaluate different models and approaches, culminating in identifying the best-performing model for the task, with the intention of first building the network from scratch. The report includes details on the dataset preprocessing, model architectures, training process, results, and insights gained from the project.

2. Dataset and Preprocessing

The AFLW (Annotated Facial Landmarks in the Wild) dataset was used for this project, thanks to Horst Possegger and the Graz University of Technology for providing the AFLW dataset.

We implemented a pipeline to filter images. Images containing more than one face were excluded, and those lacking descriptions of all required landmarks were removed. The resulting data set had some 12,000 images, that we divided into a training set, a validation set and a test

set (80/10/10). Additionally, all images were scaled to 224 x 224 pixels with three color channels, and made to use the ImageNet normalization and standard deviation (as suggested by an LLM). To simplify matters, we decided to just include four landmarks: eyes, nose, and mouth.

One member of the team, Asajad, experimented with data augmentation using Albumentations, but the results did not improve significantly.

3. Model Selection and Training

Many different models were evaluated during the project, mostly custom architectures. They were all trained for 50 epochs, by all of us. The model that finally demonstrated the best performance was **Nalberts (**https://github.com/nalbert9/Facial-Keypoint-Detection/blob/master/models.py**)**, a model structure only updated by us to decrease the number of output dimensions due to the smaller number of landmarks to detect.

MSE Loss for the landmarks, SmoothL1 Loss for the bounding box, and Adam optimizer were basically used all the time. Hyperparameters were for the most part left untouched, `num_workers`, `batch_size` and `learning_rate` were the only ones experimented with (if the former can be called such).

Below is an overview of some of the key models' architectures:

3.1. Model Architectures

We began with simple custom architectures before moving to the winning established model structure. The first model was a convolutional neural network designed to predict both facial landmarks and bounding boxes. The architecture included five convolutional layers followed by batch normalization, pooling layers, and fully connected layers for feature extraction. While the model demonstrated almost acceptable results for facial landmarks, the bounding box predictions were highly unsatisfactory.

Net

- **Conv1:** 3 → 32, kernel size 5, followed by max pooling
- **Conv2:** 32 → 64, kernel size 3, followed by max pooling
- **Conv3:** 64 → 128, kernel size 3, followed by max pooling
- **Conv4:** 128 → 256, kernel size 3, followed by max pooling
- **Conv5:** 256 → 512, kernel size 1, followed by max pooling
- Fully connected layers for bounding box and keypoint regression

This model struggled due to its limited complexity and inability to handle bounding box predictions accurately. Loss values indicated poor generalization, with validation loss plateauing early.

FacialLandmarkAndBBoxModel

- **Conv1:** 3 → 64, kernel size 3, stride 1, padding 1, followed by ReLU and max pooling

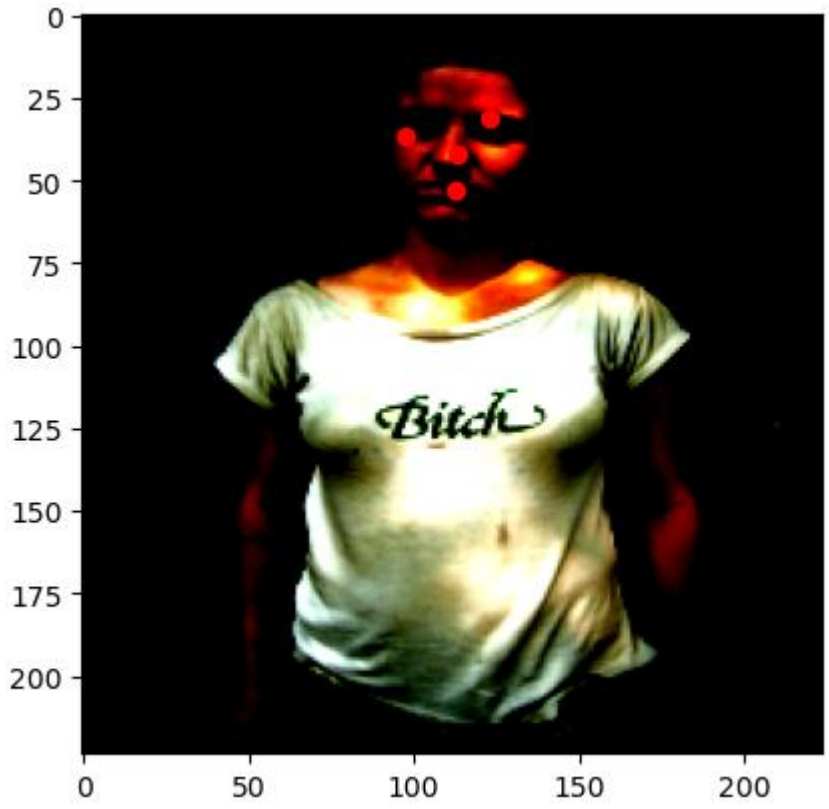
- **Conv2:** 64 → 128, kernel size 3, stride 1, padding 1, followed by ReLU and max pooling
- **Conv3:** 128 → 256, kernel size 3, stride 1, padding 1, followed by ReLU and max pooling
- **Conv4:** 256 → 512, kernel size 3, stride 1, padding 1, followed by ReLU and max pooling
- **Feature Flattening:** Flatten the feature map from the last convolutional layer
- **Fully Connected Layers:**
    - FC1: 512 × (flattened size) → 1024, with ReLU
    - FC for keypoint regression: 1024 → 8 (4 keypoints with x, y coordinates)
    - FC for bounding box regression: 1024 → 4 (x_min, y_min, x_max, y_max)

Training and Loss Trends

This was the best model we came up with that was not based on other people's work. I came up with this one, with the help of Microsoft Phi4. Training metrics indicated steady improvement in facial landmark prediction. The model's training and validation loss trends were as follows:

- **Epoch 1:** Train Loss = 21.04, Val Loss = 0.035
- **Epoch 5:** Train Loss = 0.023, Val Loss = 0.022
- **Epoch 10:** Train Loss = 0.015, Val Loss = 0.020
- **Epoch 20:** Train Loss = 0.0096, Val Loss = 0.0226

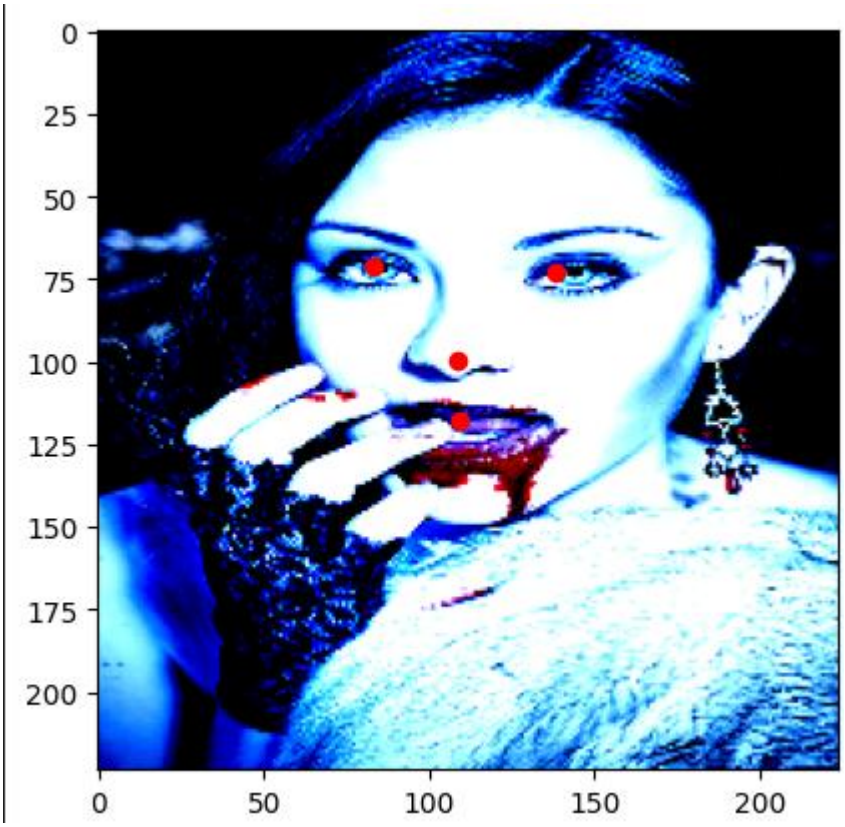
The introduction of a tried and tested model structure, like Nalbert's, significantly outperformed these results though.



Nalbert_Net

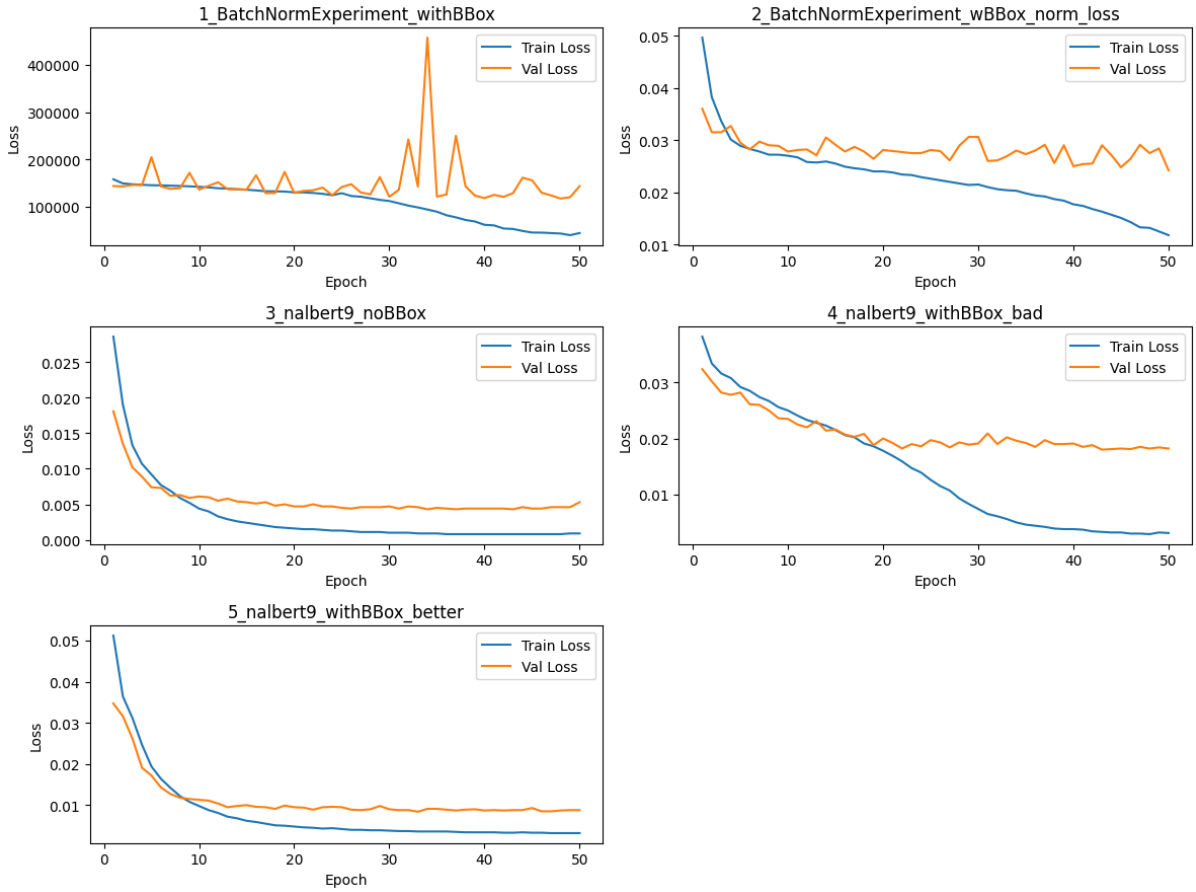
- **Conv1:** 3 → 32, kernel size 5, followed by ReLU and max pooling
- **Conv2:** 32 → 64, kernel size 3, followed by ReLU and max pooling
- **Conv3:** 64 → 128, kernel size 3, followed by ReLU and max pooling
- **Conv4:** 128 → 256, kernel size 3, followed by ReLU and max pooling
- **Conv5:** 256 → 512, kernel size 1, followed by ReLU and max pooling
- **Flatten Layer:** Converts the output feature map to a 1D vector with dimensions 512 × 6 × 6
- **Fully Connected Layers:**
    - FC1: 512 × 6 × 6 → 1024, with ReLU and dropout (p=0.25)
    - FC2: 1024 → 512, with ReLU and dropout (p=0.25)
    - FC3: 512 → 8, output layer for 4 keypoints (x, y coordinates)
- **Output:** Regression output representing 4 keypoints in 2D space (8 values in total)

Visual inspection:



3.2. Comparing loss

Here are plots based on the logs of the training and validation of some of the models, as made by Isak:



The results indicate varying performance across the experiments. In the first experiment, **1_BatchNormExperiment_withBBox**, the training loss decreases steadily, but the validation loss shows significant fluctuations and a spike around epoch 40. This suggests that the model is overfitting or experiencing instability, potentially due to an improper balance between the bounding box and other losses or insufficient regularization during training.

In contrast, the second experiment, **2_BatchNormExperiment_wBBox_norm_loss**, where the loss has been normalized, demonstrates a much smoother decrease in both training and validation losses, with the validation loss remaining relatively stable. This indicates a well-balanced training process and good generalization, likely due to the addition of normalization, which appears to stabilize the model's performance. It did however still perform quite poorly.

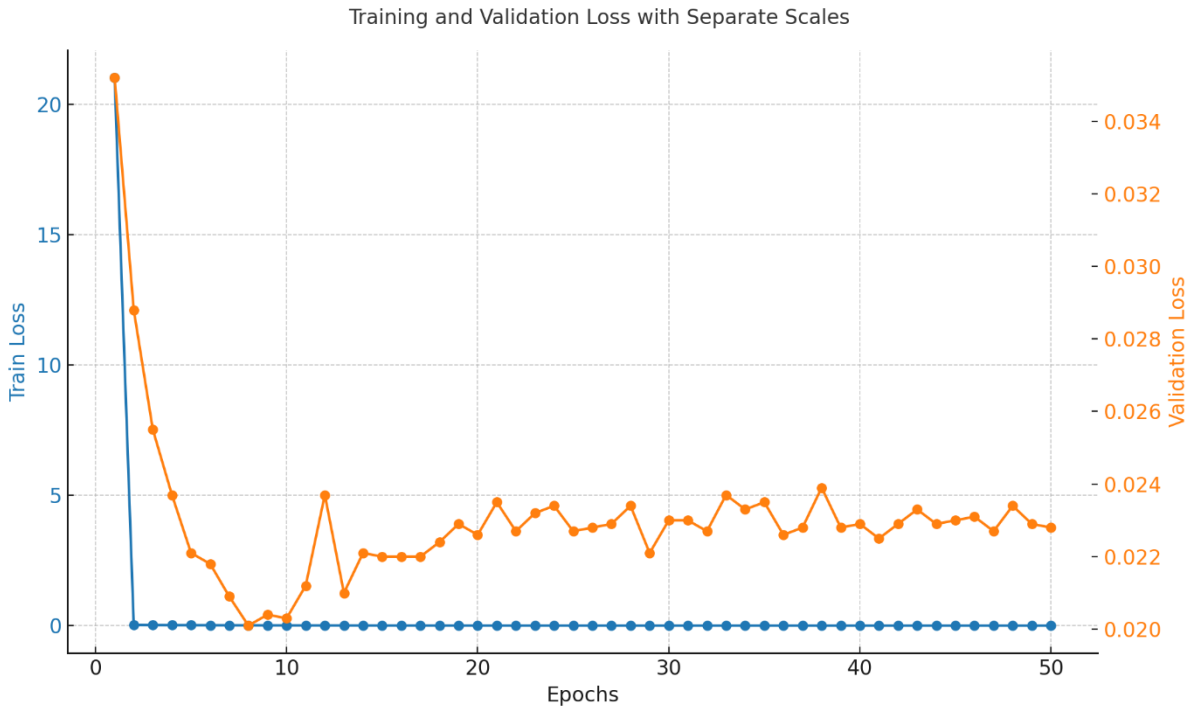
The third experiment, **3_nalbert9_noBBox**, shows very smooth convergence of both training and validation losses, achieving extremely low final losses. This outcome suggests that not introducing the bounding box prediction keeps the problem simple, allowing the model to converge more effectively and perform better overall. Exceptionally low validation loss at 0.0042.

However, in the fourth experiment, **4_nalbert9_withBBox_bad**, the training loss decreases steadily, but the validation loss levels off at a higher value and does not show significant improvement. This indicates poor generalization, which could be attributed to issues in the

architecture, loss weighting, or the challenges introduced by including bounding box predictions.

The fifth experiment, **5_nalbert9_withBBox_better**, shows a smoother decrease in both training and validation losses, with the validation loss converging well. This indicates that the model in this setup balances the bounding box task and other objectives more effectively, resulting in improved generalization compared to the fourth experiment.

An additional plot for the FacialLandmarkAndBBoxModel:



This model demonstrates a rapid decrease in both training and validation losses during the initial epochs, with the validation loss stabilizing around 0.022 after epoch 20. The lowest validation loss is achieved at epoch 8 (0.0201), suggesting strong early generalization. The close alignment between training and validation losses in later epochs indicates the model avoids overfitting, though further improvements plateau. The initial drop of the training loss was a bit odd. Overall, this setup seemingly achieves reliable convergence and consistent performance based on the loss alone. Visual inspection showed that it did not perform perfectly though, and the loss is orders of magnitude larger than that of Nalbert.

Overall, the experiments highlight the importance of normalization, balanced loss weighting, and task simplification. Normalization significantly stabilizes training and improves generalization, as seen in Experiment 2. Additionally, the simplified problem in Experiment 3 led to better performance, while Experiment 5 shows that a well-balanced approach can handle more complex tasks effectively. The first and fourth experiments, however, suffer from overfitting or poor generalization, pointing to areas where the model's setup or training strategy could be refined further.

3.3. Tools and Frameworks

PyTorch was employed for model implementation and training, while OpenCV and PIL were used for image preprocessing, though that was later moved to GPU. CUDA GPU acceleration enabled faster training times and more efficient experimentation.

3.4. Collaborative Work and Exploration

The group adopted a parallel working approach, allowing members to independently explore and experiment with potential solutions. While everyone performed similar tasks, free exploration was encouraged to identify the best models and methods. To sync our progress, we had dailies most workdays during the exercise.

During the project, one task, that I personally got, involved exploring the development of a Faster RCNN model for bounding box detection. However, this was ultimately deemed overly complex for the scope of the project. Instead, the team focused on building a neural network to identify bounding boxes alongside facial landmarks. Despite efforts, the bounding box detection never achieved satisfactory results, while facial landmark detection proved to be more successful. Something we could have explored for object detection but did not were YOLO-models.

3.5. Inference

Inference involved using the trained model to predict keypoints and bounding boxes for a given image. First, a sample image was extracted from the test dataset and prepared by converting it into a tensor, expanding it to include a batch dimension, and moving it to the appropriate device (e.g., CPU or GPU). The model was then switched to evaluation mode, ensuring layers like dropout or batch normalization do not interfere with the results. The prepared image was passed through the model to generate predictions for the keypoints and bounding box. These predictions were scaled back to the original image dimensions to match the input. Finally, the image was visualized, overlaying the predicted keypoints and bounding box.

4. Results

The models were evaluated based on the accuracy of landmark (and bounding box) placement and loss metrics. Nalbert demonstrated the best performance with precise landmark placement and the by far lowest loss value. The loss function used was Mean Squared Error (MSE), which calculated the difference between predicted and actual landmark positions.

The visualization helped verify the placement of keypoints and identify any misalignments or errors. While the bounding box predictions remained unreliable, the facial landmarks were consistently semi-accurate, except for Nalbert, that was accurate, and provided the solution to the problem we set out to solve.

5. Insights and Challenges

Key takeaways include that existing model structures such as Nalbert's outperformed our custom architectures, and starting from a robust base model saved time and yielded better results. Suggestions from chatbots like ChatGPT and Microsoft's Phi4 often lacked reliability when it came to the structure of the neural networks, emphasizing the need to rely on

established models and frameworks. For help in creating the code for training they worked fine. Training on a GPU significantly accelerated the process, naturally, allowing for more experimentation and quicker convergence. Training time could further be cut in half by also doing the preprocessing of the images on the GPU. Combining bounding box and landmark detection in a single model led to poor results, highlighting the importance of separate and specialized pipelines for these tasks. Given the brevity of the project, and the small scope, the method with no specialization, but sharing insights and progress, felt natural.

In the end, having one utils.py and one train.ipynb and many different branches also led to issues when merging the code, as the helper functions broke for many of the different solutions.