1 Experiment

Throughout the following section the results and configuration details of our trained model are described and explained.

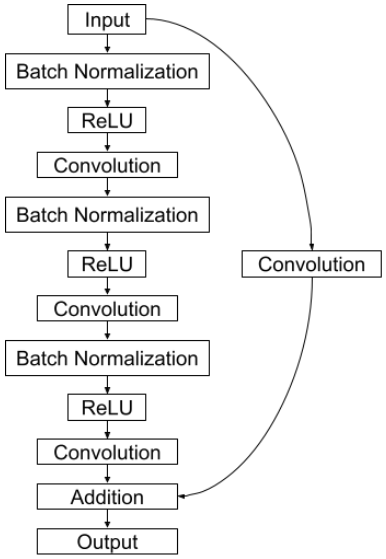
1.1 Configuration Details

- PCK og plateau
- Skriv hvorfor jeg ikke laver flere eksperimenter men kun ét

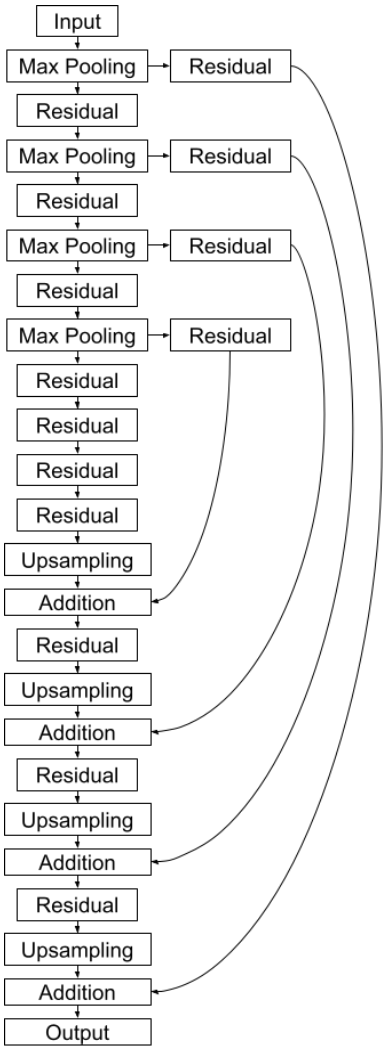
To perform the pose estimation, we make use of the stacked hourglass architecture, however, our model only consists of a single hourglass of 4 downsamples and upsamples, 1 residual module between each down- and upsample, 1 residual module in each skip-connection, and 3 residual modules in the bottleneck. Newell *et al.* [3] and Olsen [4] experiment with different amount of hourglasses and with different amount of residual modules in each hourglass. They both come to the conclusion, that stacking multiple hourglasses or using hourglasses with multiple residual modules between each down- and upsample increases the performance of the model. However, as the main purpose of this thesis is not to create a model with state-of-the-art results, but instead to create a model that can be interpreted and explored, we have chosen to reduce the size of the model.
To prevent the model from overfitting we follow Newell *et al.* [3] and use batch normalization. Newell *et al.* does not describe where to perform the batch normalization, so we follow Olsen [4] and perform the batch normalization before each convolutional layer in each residual module, after the first convolutional layer of the entire network and before the last convolutional layer of the entire network. For the choice of acitvation function we follow Olsen [4] and use the $ReLU$-function after each batch normalization. Each max pooling and nearest neighbor upsampling uses a kernel size of 2, which halves and doubles the size of the input, respectively. The full network has been visualized in Figure 1.
For the initial values of the weights we follow Olsen [4] and initialize each weight by sampling from a *Glorot normal distribution* (also known as a *Xavier normal distribution*), described as

$$\mathcal{N}\left(0, \frac{2}{fan_{in} + fan_{out}}\right)$$

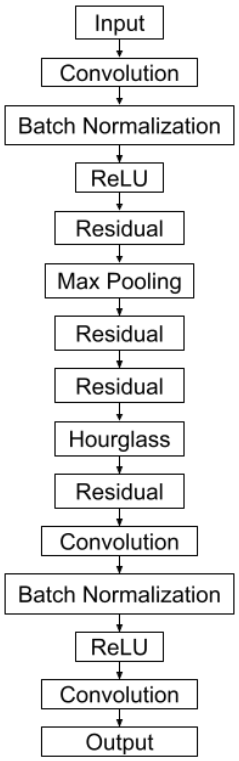
where fan_{in} is the amount of input connections and fan_{out} is the amount of output connections to the layer of the weight [1]. By doing so we make all layers have the same activation variance and gradient variance, essentially helping the model to converge [2].
We make use of a mini-batch size of 16 and no data augmentation, since the dataset is already rather large and captures a lot of the variances.
To optimize the network we follow Newell *et al.* and make use of MSE as our loss function, *RMSPROP* as our optimizer, as well as use a initial learning rate of $2.5e-4$ [3]. After each epoch we find the validation accuracy of the model by computing the *PCK* of the model.



(a) The Residual module



(b) The Hourglass



(c) The Full network

Figure 1: Overview of the used architecture

1.2 Results

- vis grafer
- Overfitting

1.3 Training Details

The stacked hourglass was implemented in Python 3.8.2 using PyTorch version 1.7.1 and Cuda version 10.2 on a machine using Windows 10 version 20H2, build 19042. The network was trained on an 8 GB NVIDIA GeForce GTX 1070 GPU using a Samsung 840 EVO SSD for data storage. Training the network takes about 70 minutes per epoch, totalling to about 58 hours for 50 epochs.