Report “Pawpularity”

by Cass Maes, Demi Soetens, Bente van Katwijk and Ilse Feenstra.

Chapter 1: Basic model

Introduction:

In western countries, owning pets is tremendously popular and pet owners would do anything to protect them. However, this is not so common everywhere in the world. In most countries, animals are living outside on the streets, without an owner taking care of them. They often end up in shelters, where they might be euthanized because of acquired trauma or diseases.

The Malaysian website PetFinder.my is a website where you can find over 180 000 of these stray animals to adopt. What is remarkable about this website is that it uses a basic Cuteness Meter to rank their pet photos. Their goal is to predict the ‘Pawpularity’ of the pet photos, utilizing several features and the performance of thousands of pet profiles. The ‘Pawpularity’ score is derived from the viewing statistics of a pet (see https://www.kaggle.com/c/petfinder-pawpularity-score/data). Having insight into the relation between pictures of animals and the ‘Pawpularity’ score could help shelters to create better pet profiles, by providing them with better pictures of their pets. Consequently, this could lead to a higher probability of the pet being adopted by a loving owner and eventually even preventing shelters from being overcrowded.

The Project

For our project, we will analyze 8.912 raw images and tabular data from PetFinder.my’s pet profiles to create a model that is able to accurately predict the ‘Pawpularity’ of pet photos. The model will take 12 features into consideration, such as whether or not the face of the pet is displayed in the photo, if it is an action shot, if a blur is added to the photo and if there is a human being in the photo. Using these features, the images and the given ‘Pawpularity’ scores, the model will be able to predict this score for our test images.

First version model

The first version of our model will be a simple convolutional neural network that is able to predict the ‘Pawpularity’. First we will preprocess and combine the data, which is explained in the section below. Subsequently, we will train our model with this data. Our goal is to make a prediction that is slightly better than a random prediction.

Data analysis and preprocessing

Firstly, we checked if there were any samples that were missing tabular data, which was not the case. We then looked at the distribution of the data to see if there were outliers (figure 1). The distribution of the data was slightly right-skewed with the peak at 30. The mean is 38.04 and the standard deviation is 20.59. Noticeably was the outlier at a score of 100. Possibly, it would be better to remove these samples, but further analysis should be done to conclude this.

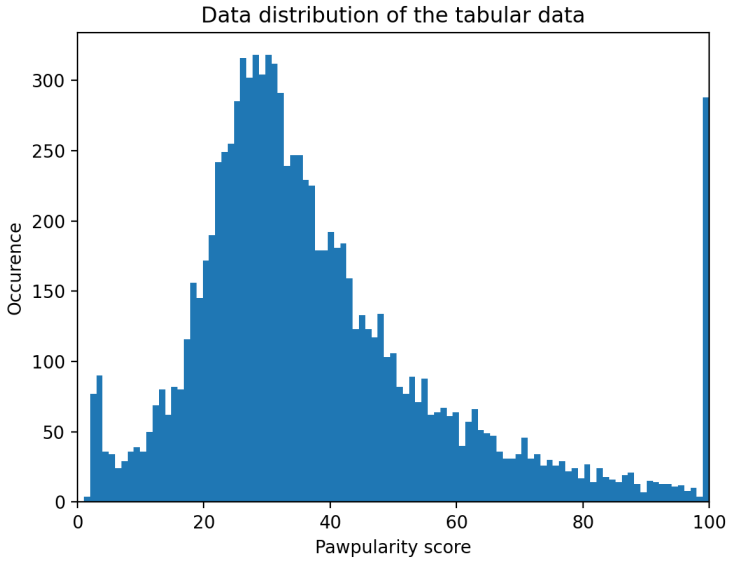


Figure 1: Histogram of the data distribution of the tabular data. On the x-axis is the Pawpularity score presented and on the y-axis the occurrence of each score. The graph is slightly right-skewed with an outlier at the score of 100.

Secondly, we looked at the image data and saw that not all images had the same size and orientation. In order to correct this, we reshaped each of the images to images of the size 64 x 64 pixels. Furthermore, we sorted the dataframe based on the order of the images so they are indexed to the right tabular data. We used this combined data in our Convolutional Neural Network model. Thirdly, we splitted the training data into a training data set (80%) and a validation data set (20%). We started with shuffling the image - and tabular data for the reason that the validation - and training data can be chosen randomly. Then, we sorted the image data, in order that the image - and the corresponding tabular data had the same index. This way, we knew for certain that the same image - and tabular data was selected for the validation data. We choose the first 20% of the sorted data as validation data and the other 80% of the sorted data was assigned to be the training data.

Model Pipeline and Training

At first we started with 2 separate models, one tabular data model and another for the actual images. The input of the image model was the reshaped images (64 x 64) and is a Convolutional Neural Network (CNN) with a linear (regression) output and 1 hidden layer of 20 nodes. We chose to implement a CNN, because this type of model is suited for image data, since spatial information in the images remains preserved. We added 2 convolutional layers, where the first one has 64 filters and the second one 128. The kernel size for both is 3,3 with a relu activation function. After each of these layers we added a max pooling layer with a pool size of 2,2 and a stride of 2. After a flattening layer we added a dense layer where the number of units is equal to the amount of hidden nodes, so 20 hidden nodes. This layer has a relu activation as well.

The input of the tabular data was a csv file with the rows representing the samples and the 12 columns representing the features in an image. If a feature was present in a photo, this feature got score 1 and if the feature was absent, a score of 0 was given. This data was used to build a Dense Neural Network (DNN) with 1 dense layer of 20 nodes and a relu activation function. We chose this

type of model for the tabular data, since a DNN is the easiest to combine with the CNN for the image data.

After creating separate networks for image and tabular data, we merged these models using the concatenate function of the tensorflow.keras library. The model has the tabular neural network output and the convolutional neural network output as its inputs and one hidden layer with 20 hidden nodes and a relu activation function. The output of the model used a linear activation function. We used 20 epochs to train the model with a split of 90% training data and 10% validation data, using the mean squared error function and the root mean squared error as metric, to evaluate the model. An overview of the model with all the layers is shown in figure 2.

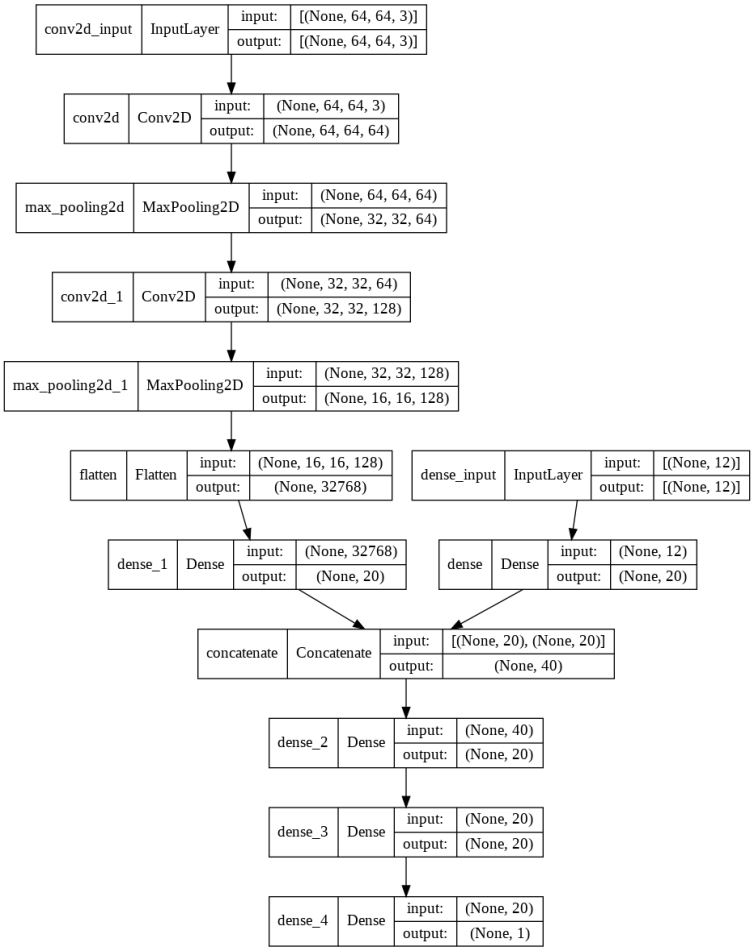


Figure 2: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below).

Evaluation and conclusions

In this first version of our model we scaled the images and used a linear activation function for the output layer. To analyze the performance of the model we used the Mean Squared Error (MSE) as the loss function because this is a relatively simple metric to calculate the gradient. We used the Root Mean Squared Error (RMSE) as the metric because this is easier to interpret than the MSE since the RMSE is the mean absolute deviation. Figure 3 shows the training MSE in blue and the validation

MSE in orange, while the model is being trained. After 20 epochs, the final training MSE is 345.20 and the final validation MSE is 482.29. Figure 4 shows the training metric in blue and the validation metric in orange. The final training metric is 18.57 and the final validation metric is 21.96.

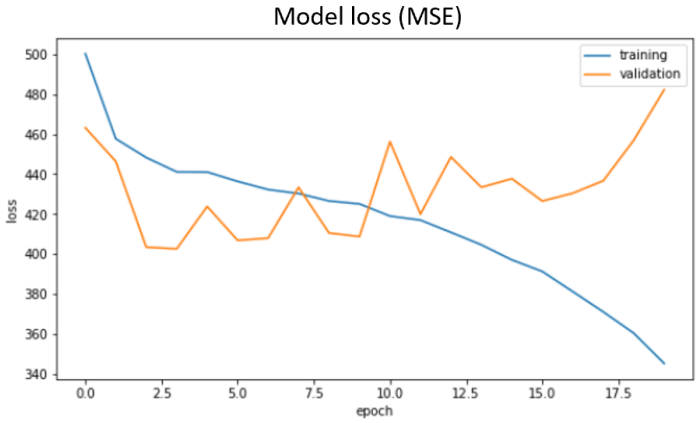


Figure 3: Graph showing the training (blue) and validation (orange) MSE (mean squared error) of the concatenated model for 20 epochs.

Figure 4: Graph showing the training (blue) and validation (orange) root mean squared error of the concatenated model for 20 epochs

The model does seem to overfit on the training data, since the training MSE is decreasing, while the validation MSE is increasing. This means that the model is fitting the training data too specifically, so it does not generalize well to new samples.

In later versions, we can try to add preprocessing methods to prevent the model from overfitting. There are multiple methods that can prevent overfitting, such as input normalization, batch normalization and adding dropout layers. Moreover, we now used a linear activation function for the final output. This means that the output could have any number. The possible output, however, ranges from 0 to 100. In later versions we could set a minimal and maximal output threshold of 0 and 100, respectively.

Chapter 2: Mean centering and standard deviation normalization

Introduction

For this version of our model, we want to focus on reducing the overfitting of our previous model. The model was learning with our training data and not with our validation data, which clearly indicates overfitting. Therefore, we want to add preprocessing of our data in order to address the overfitting. We will normalize the input by mean centering and standard deviation normalization. This way, there is less variation between the samples. Since there is less variation, the model can train less on the variation in the samples and more on the underlying features. This makes the model more generalizable to new data and thus less likely to overfit.

Data analysis and preprocessing

We added the data preprocessing methods mean centering and standard deviation normalization. We applied this function to the train and validation data, by which the training - and validation data are still comparable. We only applied this preprocessing to the image data, since the tabular data is binary.

In order to add preprocessing, we needed to split the data in a training and validation set. To ensure that both the tabular and image data of the samples were split the same, we first shuffled the image data and later sorted the tabular data based on the shuffled images. After this we took the first 20% samples for the validation set and the last 80% for the training set. Because we shuffled at first, this split was randomly and not dependent on the sequence of the image data.

Model pipeline and training

We wanted to confirm that the preprocessing was working properly and handled all data correctly. Therefore, we also trained the model only on the convolutional network for the image data. We compared the MSE values of the concatenated network with the convolutional network and examined if the concatenated model indeed performed better than the convolutional network only.

Evaluation and conclusions

After 20 epochs, the concatenated model has a final training MSE of 105.89 and a final validation MSE of 593.03. The convolutional model has a final training MSE of 179.02 and a final validation MSE of 603.06 after 20 epochs. Therefore we can conclude that the concatenated model is performing better than the convolutional model because the concatenated model has a lower validation MSE. Figure 5 shows the comparison of the concatenated model MSE and the convolutional model MSE with the training metric in blue and the validation metric in orange.

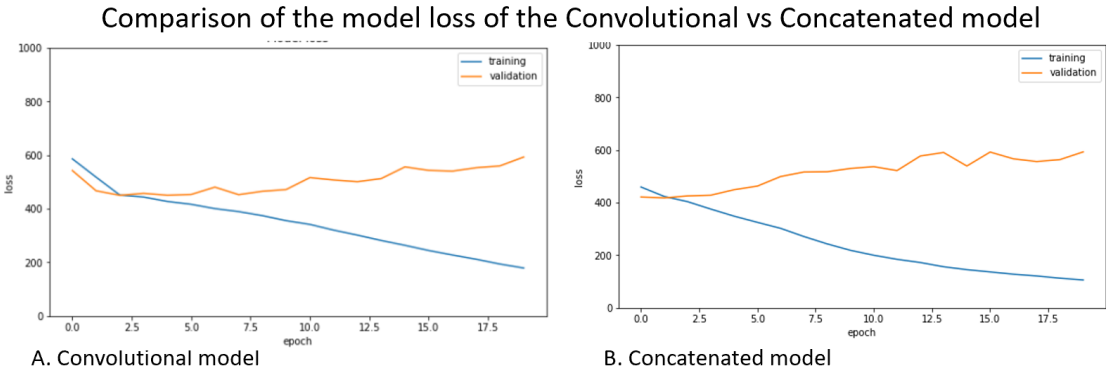


Figure 5: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the convolutional (A) - and concatenated model (B) for 20 epochs of training.

Looking at figure 5b, the concatenated model seems to diverge less than the convolutional model and has a lower validation MSE. This means that the addition of the tabular data does help the model to learn. Therefore, we applied the preprocessing to the concatenated model.

The training MSE of the concatenated model after preprocessing is 12.83 after 20 epochs and 569.05 for the validation MSE. So, the training MSE after preprocessing decreased a lot whereas the validation MSE increased. The results are visualized in figure 6. From this, we can conclude that the preprocessing methods did not solve the problem of overfitting. However, the validation MSE of the model after preprocessing is slightly lower.

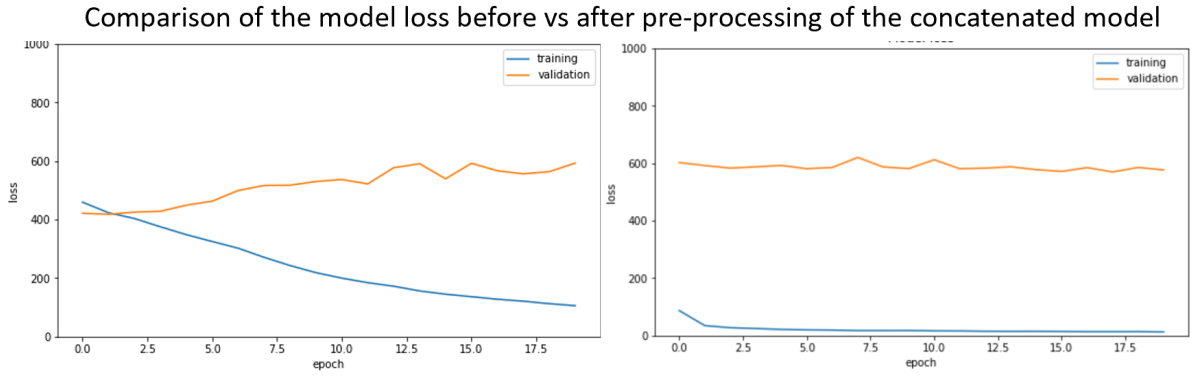


Figure 6: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the concatenated model before preprocessing (A) - and concatenated model after preprocessing (B) for 20 epochs of training.

After we reevaluated, it makes sense that the model still overfits. Standardization and centering of the model does not help against overfitting because it does not learn anything between the relation of the features. Standardization and centering could be used to let the model learn faster because it has more impact on gradient descent, especially for the training data. As preprocessing helps the model to learn better for the training data, we decided to keep the preprocessing. We will try to reduce the overfitting with other means. For instance, dropout and batch normalization could be used.

Chapter 3: Dropout

Introduction

After applying the preprocessing in the previous chapter, the model was still overfitting. In order to reduce the overfitting we will add dropout layers to the model. When adding a dropout layer, you add a probability for the nodes to be silenced. Since a certain number of nodes are randomly silenced for each layer, the next layer cannot rely too much on specific nodes of the previous layer. Therefore, it needs to learn general features, instead of the details. In this way the model becomes more generalizable and therefore it is less likely to overfit.

Data analysis and preprocessing

Data analysis and preprocessing methods remain the same in this version.

Model pipeline and training

We tried various probabilities for dropout for the different layers. In order to add a dropout layer to the tabular neural network, we first needed to add a layer to the tabular neural network. So, the tabular neural network now consists of 2 hidden layers, with each 20 hidden nodes and relu activation (figure 6). The different probabilities tried and their outcome can be found in table 1. We started with varying the probabilities of dropout between 0.2 and 0.5. However, this did not result in a reduction of overfitting. Hence, we tried larger dropout values, such as 0.6 and 0.8 to check if dropout works.

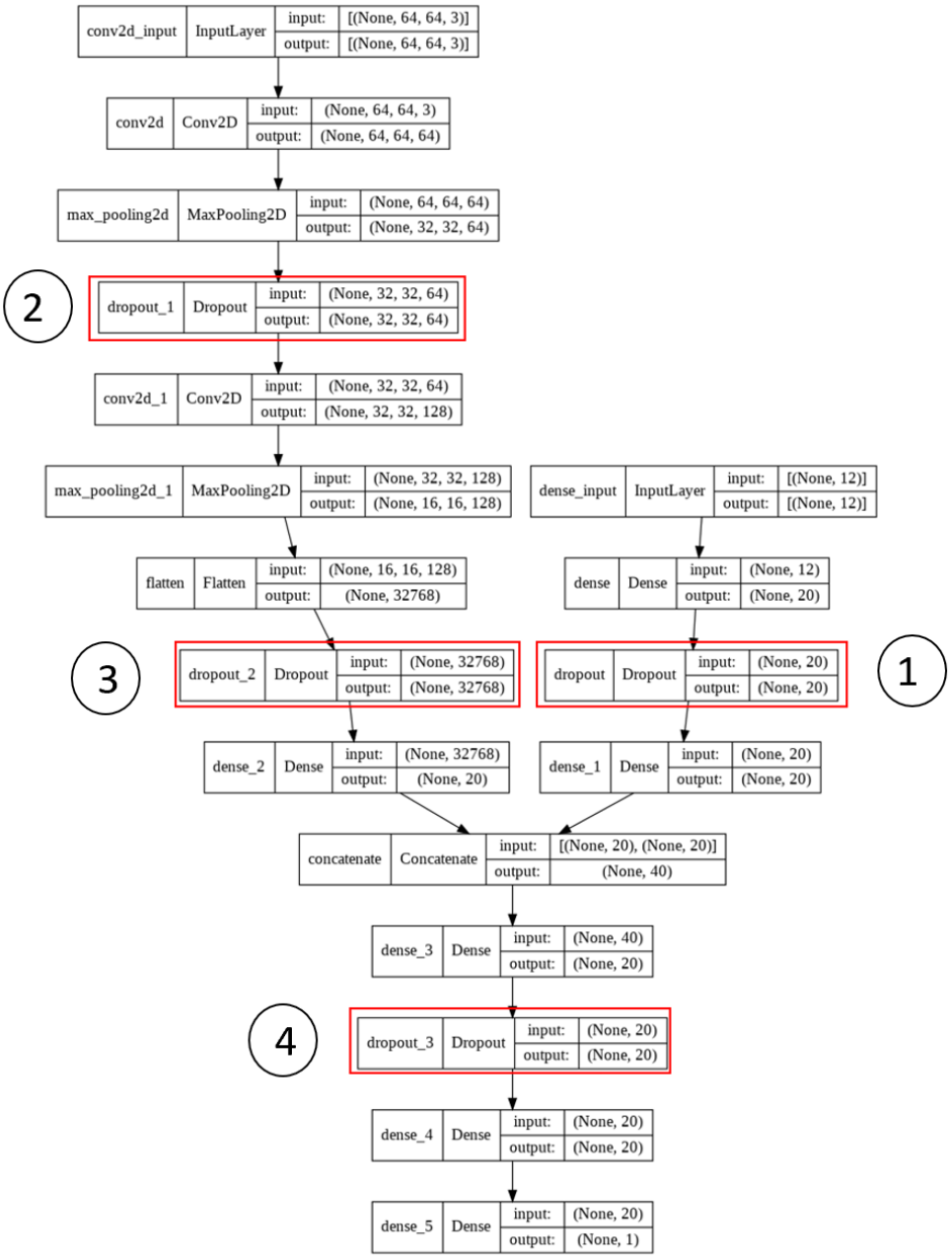
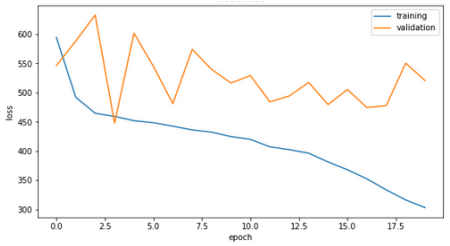
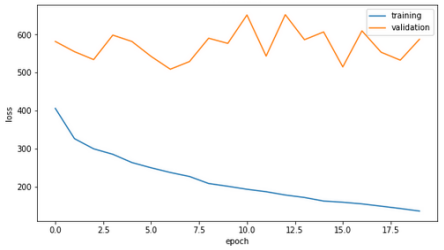
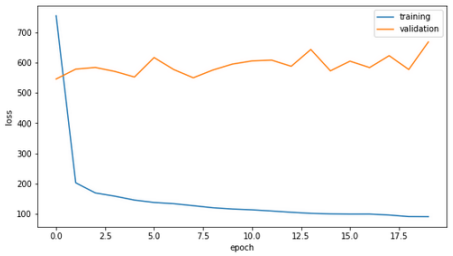
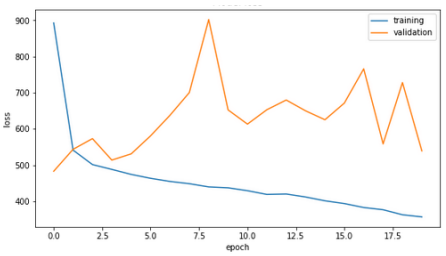


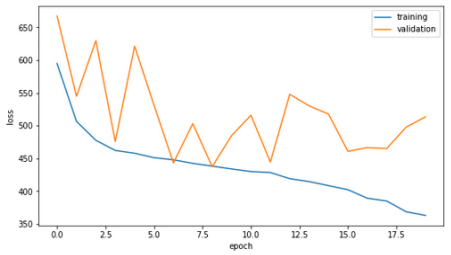
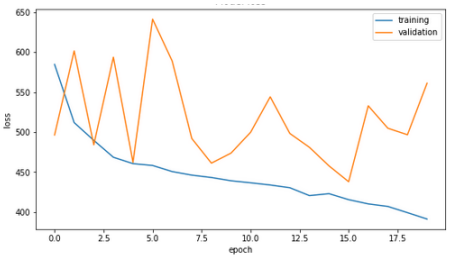
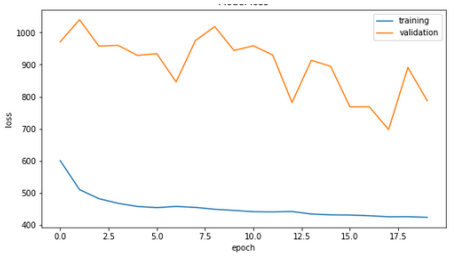
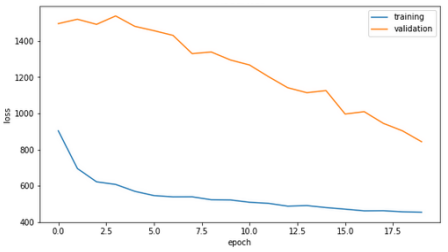
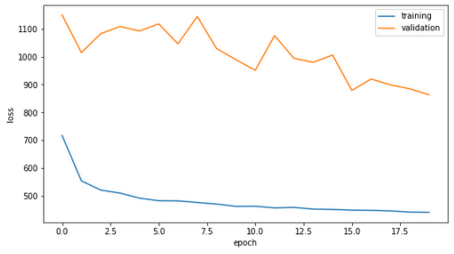
Figure 7: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below) with dropout (red).

Evaluation and conclusions

The results of the different dropout probabilities are given in table 1.

Table 1. MSE for different probabilities of dropout Layers. The applied dropout rates are displayed in the left column, the corresponding MSE is displayed in the middle and the corresponding graphs are displayed in the right column. In the model MSE graphs, the y-axis scales are not equal.
The layers that are implemented are displayed in figure 7: 1: Tabular Dense layer, 2: Convolutional Conv2D layer, 3: Convolutional Dense layer, 4: Concatenated Hidden layer.

Dropout Layers	**MSE MSE**	**Model MSE Graph**
Tabular Dense Layer: 0.4 Convolutional Conv2D layer: 0.4 Dense layer: 0.2 Concatenated Hidden layer: 0.2	Training MSE: 303.0349 Validation MSE : 520.6554 Difference: 217.6205	
Tabular Dense layer: 0.2 Convolutional Conv2Dl layer: 0.4 Dense layer: 0.2 Concatenated Hidden layer: 0.2	Training MSE: 136.8040 Validation MSE: 587.4723 Difference: 450.6683	
Tabular Dense layer: 0.5 Convolutional Conv2d layer: 0.4 Dense layer: 0.2 Concatenated Hidden layer: 0.2	Training MSE: 90.8209 Validation MSE: 668.6118 Difference: 577.7909	
Tabular Dense layer: 0.2 Convolutional Conv2D layer: 0.2 Dense layer: 0.2 Concatenated Hidden layer: 0.4	Training MSE: 356.9272 Validation MSE: 539.2504 Difference: 182.3232	

Tabular Dense layer: 0.4 Convolutional Conv2D layer: 0.4 Dense layer: 0.4 Concatenate Hidden layer: 0.2	Training MSE: 363.0929 Validation MSE: 513.5646 Difference: 150.4717	
Tabular Dense layer: 0.5 Convolutional Conv2D layer: 0.5 Dense layer: 0.5 Concatenated Hidden layer: 0.2	Training MSE: 391.1426 Validation MSE: 561.2825 Difference: 170.1399	
Tabular Dense layer: 0.8 Convolutional Conv2D layer: 0.8 Dense layer: 0.4 Concatenated Hidden layer: 0.4	Training MSE: 422.5980 Validation MSE: 787.2150 Difference: 364.6170	
Tabular Dense layer: 0.8 Convolutional Conv2d layer: 0.8 Dense layer: 0.8 Concatenated Hidden layer: 0.8	Training MSE: 454.0530 Validation MSE: 843.5389 Difference: 389.4859	
Tabular Dense layer: 0.6 Convolutional Conv2D layer: 0.6 Dense layer: 0.6 Concatenated Hidden layer: 0.6	Training MSE: 440.9618 Validation MSE: 863.1171 Difference: 422.1553	

The graph of the dropout rate of 0.8 for the tabular, convolutional and concatenated network showed the best trend downwards for the validation MSE. Therefore, we chose this dropout rate and increased the number of epochs to 40 to investigate whether this trend will continue. After the increase in the number of epochs to 40, the final training MSE is 527.9708 and the final validation MSE is 530.1221 of this model. The results are shown in figure 8.

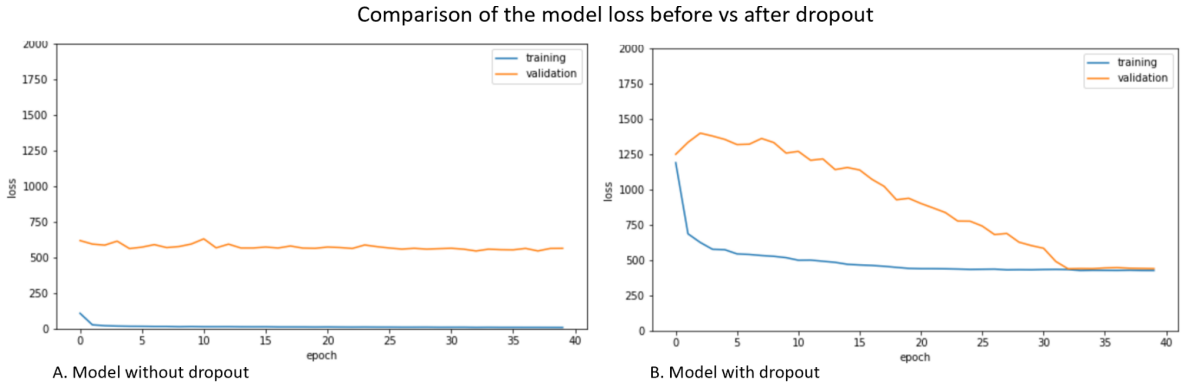


Figure 8: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model before dropout (A) - and model after dropout (B) for 40 epochs of training. The training MSE is 527.97 and the validation MSE is 530.12 after the dropout rate of 0.8 for the the tabular, convolutional- and concatenated network.

The model with the dropout layers has a final training MSE of 527.97 and a final validation MSE of 530.12 (figure 8). The validation MSE is almost equal to the training MSE, which means that the model does not overfit anymore. The current training MSE is 427.97 and the previous training MSE was 12.83 after 40 epochs. This means that the training MSE increased using dropouts. Consequently, we lost a lot of information due to dropout, so the learning performance of the model worsens.

The model now does not overfit, but has a high MSE, meaning that our model does not learn the data perfectly yet. For our experiment, we will first try to make the model learn better, by adding extra layers. If that succeeds, we want to try to finetune the dropout rate in order to retain more information. We will try to lower the dropout rate, while still preventing the model from overfitting. Since the overfitting was reduced the most when the dropout before the concatenated layer was set to 0.8, we will first try to reduce the dropout of the other layers.

Chapter 4: Adding layers

Introduction

After adding dropout layers, it takes the model 40 epochs to correctly learn. Furthermore, the model does not overfit anymore, when the dropout rate for each layer is set to 0.8. However, the validation MSE is still fluctuating a lot. Therefore, we will try to add several layers to create a deeper network that is able to learn better, so the training and validation MSE will decrease. We will still use a dropout rate of 0.8, to isolate only the effect of adding extra layers.

Data analysis and preprocessing

Data analysis and preprocessing methods remain the same in this version.

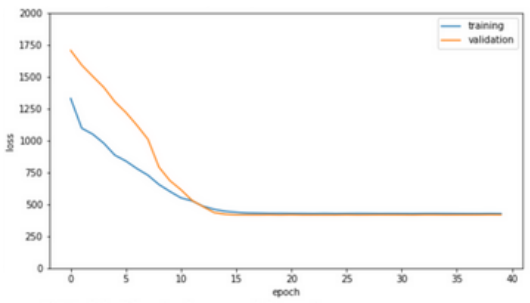
Model pipeline and training

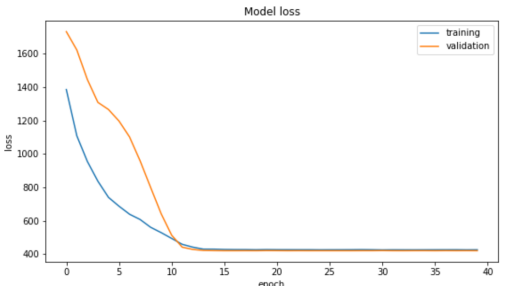
We tried two new versions of the model. In the first version, we added one extra layer in the tabular neural network, with a dropout rate of 0.8 and 20 hidden nodes and relu activation. In the convolutional neural network we added one more convolutional layer and one extra dense layer, with 20 hidden nodes. In the concatenated model we added one dense layer as well. The number of epochs is again set to 40. In the second version, we added another convolutional layer.

Evaluation and conclusions

Table 2 shows the different versions of the network, showing the learning curves and the final model MSE (root mean squared error).

Table 2. Model training and validation MSE when adding more layers. The new layers are displayed in the left column, the corresponding MSE is displayed in the middle and the corresponding graphs are displayed in the right column. In the model MSE graphs, the y-axis scales are not equal. The layers that are eventually added are visualized in figure 10.

Layers	Model MSE	Model MSE graph
Tabular 1 extra Dense layer Convolutional 1 extra Convolutional layer 1 extra Dense layer Concatenated 1 extra Dense layer	Training MSE 525.85 Validation MSE 521.31	

<u>Tabular</u> 1 extra Dense layer <u>Convolutional</u> 2 extra Convolutional layers 1 extra Dense layer <u>Concatenated</u> 1 extra Dense layer	<u>Training MSE</u> 525.84 <u>Validation MSE</u> 520.21	

For the model with one extra layer in each submodel, the training and validation MSE decrease as the model is being trained. The validation MSE is lower than the training MSE, meaning that the model does not overfit. The same is true for the model with 2 extra convolutional layers. Since both networks perform equally well, we choose the model with one extra convolutional layer because this model is simpler.

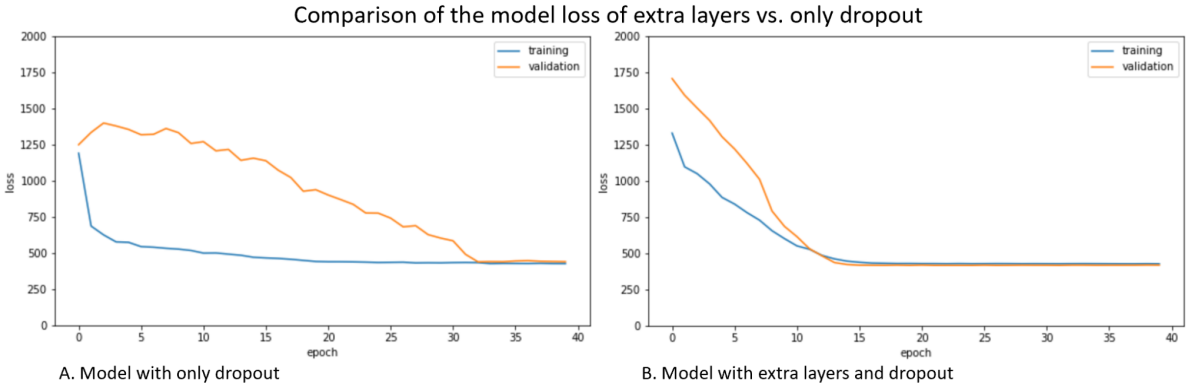


Figure 9: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model with only dropout (A) - and model after adding extra layers (B) for 40 epochs of training.

The final training MSE of the model with 40 epochs with only 1 extra layer was not lower than in the previous model from chapter 3 (525.85 vs 527.97, respectively). However, the shape of the validation MSE now shows less variability, which is therefore easier to reproduce. The comparison of the model is shown in figure 9. An overview of the new model with the extra layers is displayed in figure 10. Hence, we will continue with this model. We will try to decrease the dropout per layer to retain more information about the data. We will do this by slowly decreasing the dropout probability per layer. We expect that the neural network can learn specific features better and that the MSE will decrease. We have to be cautious that the model does not start to overfit again with a lower dropout rate.

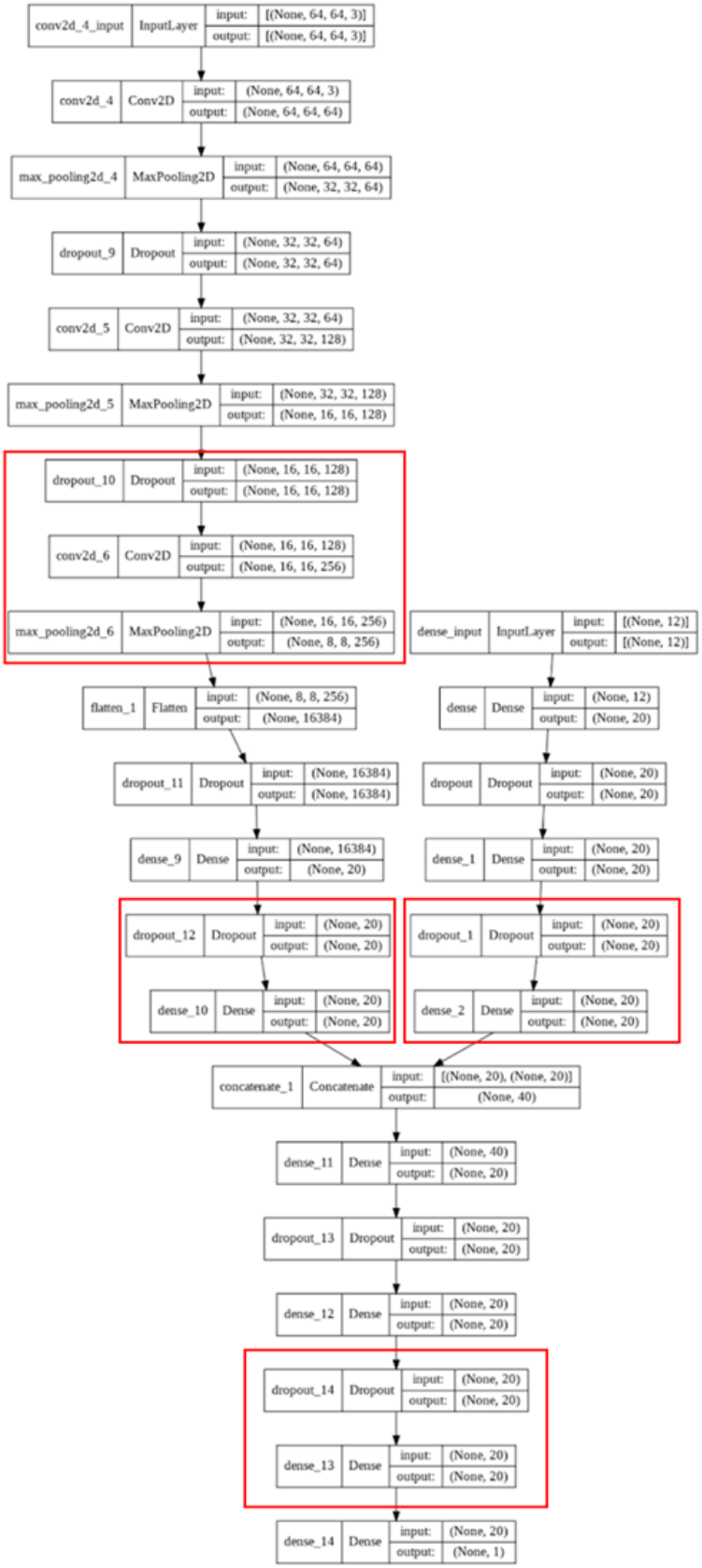


Figure 10: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below). The new layers are highlighted in red. This is the network we will use in later versions.

Chapter 5: Reducing the dropout rate

Introduction
In this version, we will try to reduce the dropout rate per layer. Our previous model had a dropout rate of 0.8 for each layer. Because of this, plentiful information is lost. In order to lose as little information as possible, we will decrease the dropout rate, while still preventing the model from overfitting. We expect that this will result in a lower training and validation MSE.

Data analysis and preprocessing
Data analysis and preprocessing methods remain the same in this version.

Model pipeline and training
For this version, we tried several dropout rates per layer. We started with decreasing one layer at a time from a dropout rate of 0.8 to 0.7. We continued this method of decreasing the dropout rate with 0.1 per layer and checked if the model still did not overfit after the decrease of the dropout rate. Since this method would require trying a lot of combinations, we did not try every combination but instead tried only a couple. We monitored carefully whether the model was overfitting.

Evaluation and conclusions
When using a dropout rate of 0.2 for one or more layers, the model started to overfit again. Eventually, the best option was a model where all layers had a dropout of 0.4, except for the dense layers of the CNN, which had a dropout of 0.3. An overview of the dropout rates per layer is given in figure 11.

The table with all the attempts can be found in appendix A. Looking at the learning curves and the final training and validation MSE, we choose to continue with the model that has each dropout rate set to 0.4, except for the dropout rate in the two dense layers in the CNN, which are set to 0.3. We choose this model, because the final validation MSE (467.07) is lower than the previous final validation MSE final training MSE (525.17) and the learning curves show that the model is not overfitting. The best attempt with decreasing the dropout rate is displayed in figure 12.

Slowly decreasing the dropout rates leads to more fluctuating learning curves, especially for the validation MSE, and the model overfitting again. This makes sense, because the model now can learn features more specifically, so it is more sensitive to noise in the data.

The model does not overfit anymore, but the MSE is still quite high. Moreover, there is more variation in the MSE over the epochs. To improve the network, the network should be more stable and the network should better learn the features. We will try to achieve this by applying Batch normalization and limiting the output of the model from 0 to 100.

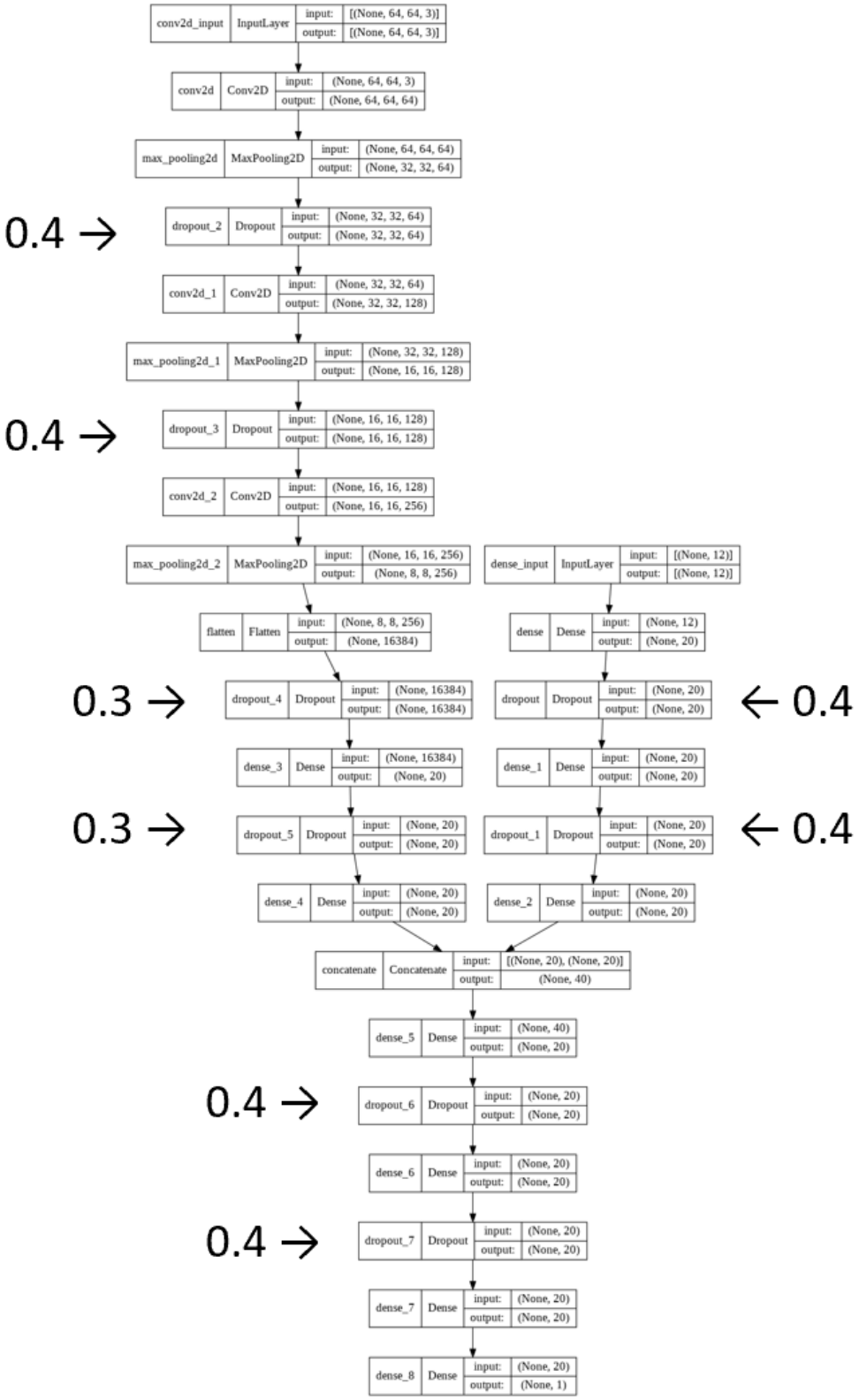
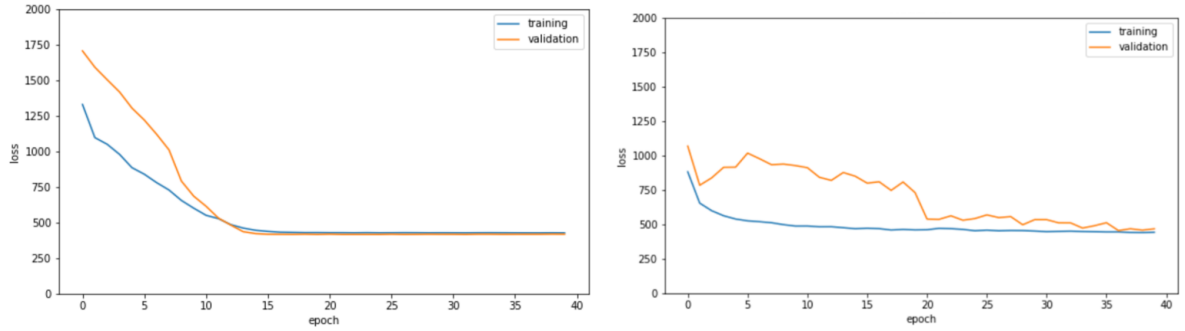


Figure 11: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below). The updated dropout rates per layer are displayed on the side. This is the network we will use in later versions.

Comparison of the model loss of old dropout rate vs. a reduced dropout rate



A. Previous model

B. Model with reduced dropout rate

Figure 12: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model before reducing the dropout (A) - and model after reducing the dropout (B) for 40 epochs of training.

Chapter 6: Batch normalization

Introduction

To increase the learning speed of the network we will apply batch normalization. When performing batch gradient descent, the input range varies greatly between batches. Batch normalization will reduce this variance by applying a scaling factor to them. After this, all weight vectors will be closer to each other which will lead to more stable gradients and thus faster convergence. The network is also able to learn better because the individual neurons will not react strongly to small changes. It might eventually lead to a reduction of the number of training epochs required to train the network.

Data analysis and preprocessing

Data analysis and preprocessing methods remain the same in this version.

Model pipeline and training

For this version we applied a batch normalization layer after each maximum pooling layer in the Convolutional Neural Network. This places the normalization layer before each of the activation layers. We kept the batch size at the default size of 32 and used 60 epochs, as also done in the previous versions.

Evaluation and conclusions

The outcome of adding batch normalization is depicted in figure 13. After applying batch normalization, the training MSE has decreased from 525.17 to 198.43 because of the reduced dropout rate and the 60 epochs. This is a significant decrease of more than 60%. While the training MSE turned out to be relatively low, the validation MSE increased with batch normalization. It increased from 467.07 to 725.96 after 60 epochs. The cause for this might be that our model does not generalize well enough to new samples and overfits. In order to solve this problem we will add regularization terms in chapter 8.

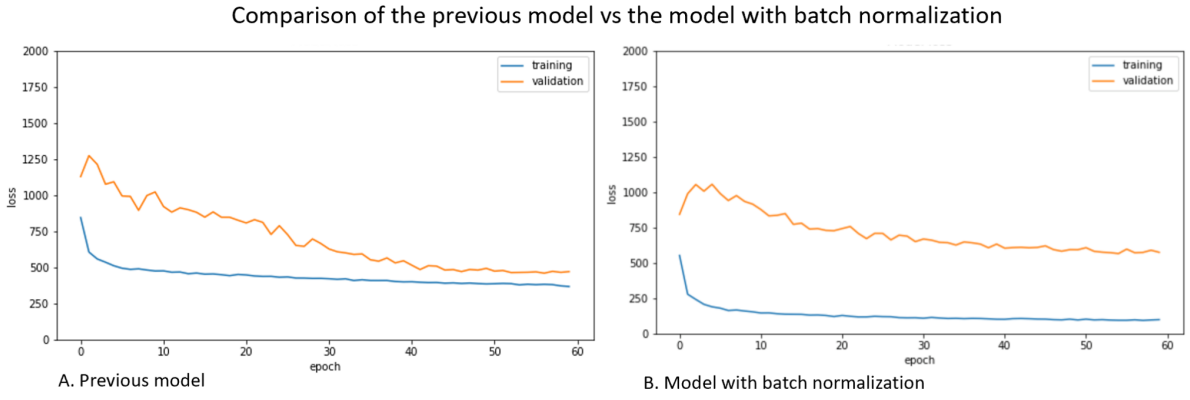


Figure 13: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model before applying batch normalization (A) - and model after applying batch normalization (B) for 60 epochs of training.

Chapter 7: Linear output

Introduction

We want to predict a 'Pawpularity' score between 0 and 100 with our model. Therefore, we want to restrict the output of our model between 0 and 100. In our previous model, we used a linear output without output constraints. In this version, we will add a clip function to our output layer, in order to set the output in the range of 0 to 100. Our goal is to improve the model output and lower the validation MSE.

Data analysis and preprocessing

Data analysis and preprocessing methods remain the same in this version.

Model pipeline and training

We customized an activation function for the output layer of the network, so the activation function now has a linear output between 0 and 100. This way, all the output lie between 0 and 100, which is the possible range for the 'Pawpularity' score. We tested this function on the model from chapter 5, so without batch normalization.

Evaluation and conclusions

The model using the clipped linear output gives a good result for the model MSE compared to our previous version. After 60 epochs, the new training MSE was 525.39 compared to 198.43 and the new validation MSE was 609.38 compared to 725.96. The results are displayed in figure 14. Because the model had a decreased training cost and decreased validation cost, the model overfits less. Hence, the clipping of the linear output improves the model and should therefore be used in the following models.

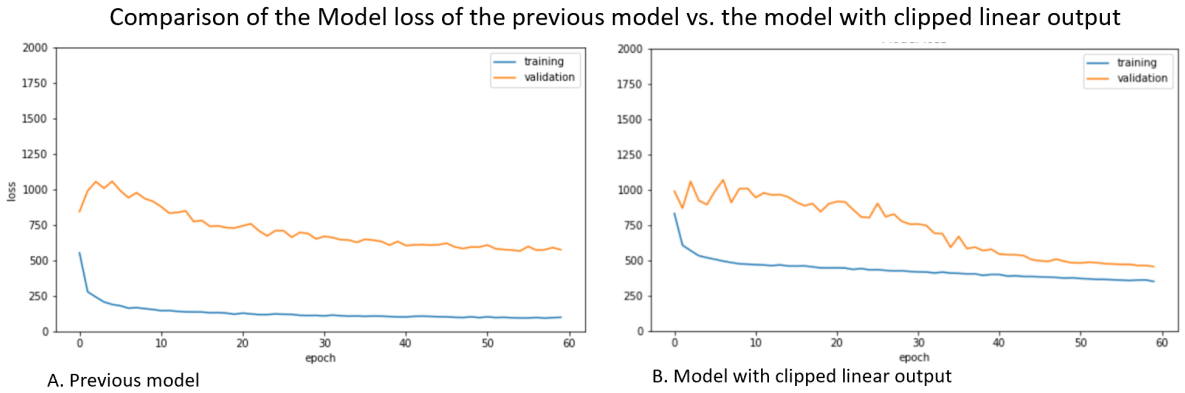


Figure 14: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model before clipping the output (A) - and model after clipping the output (B) for 60 epochs of training.

Chapter 8: Regularization term

Introduction
Applying batch normalization led to a notable decrease in training MSE, whereas the validation MSE increased a little. This means that batch normalization causes the network to learn features from the data better, but it does not generalize well to new samples yet. That is why we decided to apply more regularization by adding a regularization term. By adding a regularization term, the weights in the network have to be kept low to keep the cost low. Thus, there is a trade-off in the network between perfectly learning the features from training data (with large weights) and keeping the weights as low as possible, making the network more generalizable. The regularization parameter lambda decides the position of this trade-off.

Data analysis and preprocessing
Data analysis and preprocessing methods remain the same in this version.

Model pipeline and training
Different types of regularization will be tried to the model of chapter 7. In each layer, it is possible to add a kernel, bias and activity regularization. The options of regularization are L1, L2 or L1 and L2 combined.

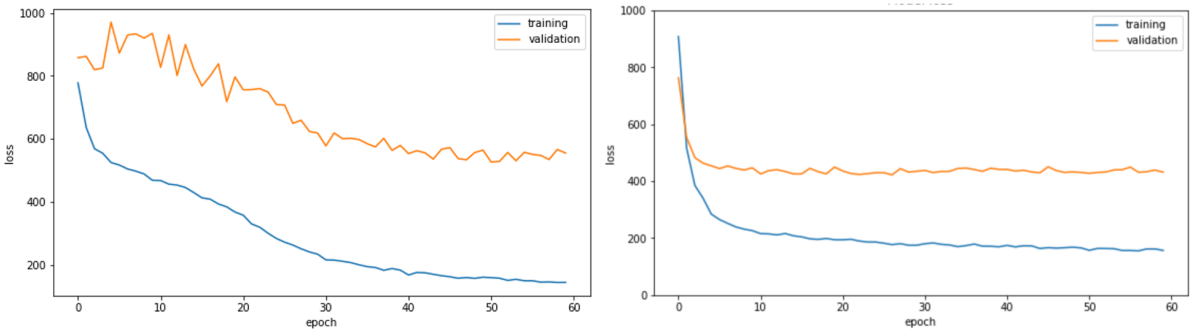
From table 1, it seems like overfitting arises in the concatenated model, so regularization terms will be added in these layers as a starting point. Regularization terms will also be applied in the dense layers of the CNN. Different values for the regularization parameter lambda will be used. We will use L2 regularization, since this is the most common type.

Evaluation and conclusions
After several tries of many different options we figured that it was not possible to get our model to the point where it would perform better on the validation data than before but still remain a low training MSE. All tries are visible in Appendix B. We eventually chose a regularization term where the training MSE was still relatively low and where the validation MSE was not too high. This model is still overfitting but less than the other options. It also seems to perform well before 30 epochs. We used an L2 kernel, bias and activity regularizer of 1e-3 on the dense layers of our CNN and changed lambda to 1e-1 for the dense layers of the concatenated model. This led to a training MSE of 156.82 and a validation MSE of 431.86 after 60 epochs.

Both the training and validation MSE decreased compared to the previous model, which had a training MSE of 525.39 and validation MSE of 609.38 after 60 epochs. Because of this decrease in both MSEes, this model outperforms the previous model. The comparison of the model MSE of the previous model compared to the model with the added regularization terms is visualized in figure 15.

Although this model is an improvement compared to the previous model, it still overfits. Since we already applied dropout and regularization terms, we will try to improve the model in different ways. For instance, we will look at outliers and data augmentation to prevent overfitting.

Comparison of the model loss with vs. without regularization term



A. Model without regularization term

B. Model with L2 regularization term

Figure 15: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model before adding the L2 regularization term (A) - after adding a L2 regularization term of 1e-3 on the 2 dense layers in the CNN and a term of 1e-1 on the dense layers of the concatenated model (B) for 60 epochs of training.

Chapter 9: Removing outliers using K-fold cross-validation

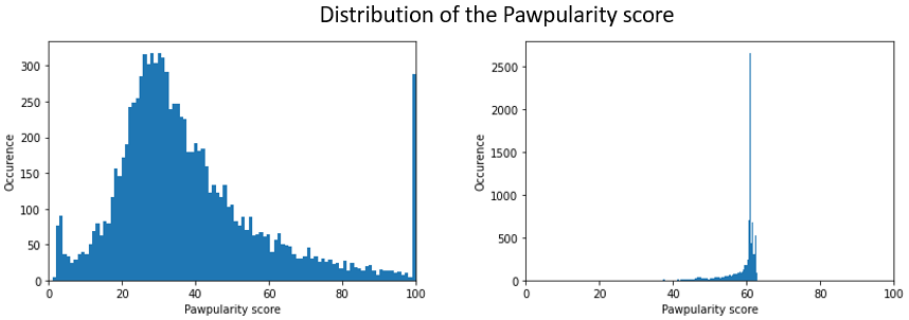
Introduction

When reviewing the distribution of the Pawpularity score, the high amount of samples that had a score of 100 stood out. Furthermore, the distribution was slightly right-skewed. It could be that when calculating the Pawpularity score, a cutoff was made at 100. So, the photos with a score higher than 100 were set to 100, causing the high amount of samples with this score. In this case, the samples at 100 are outliers and it could be that the model performs better without these outliers. In order to check this, we will first look at the distribution of the predicted Pawpularity scores. If this distribution is also right-skewed, the predictions are a good representation of the real Pawpularity scores. If this is not the case, the predictions do not properly reflect the real situation. Training the model without the outliers may then result in a better model. Hence, in this chapter we will first look at the distribution of the predicted Pawpularity score. If this distribution is not right-skewed we will train the model without the samples that have a score of 100 to see if that results in a better performance of the model. To compare the model with and without the outliers, we will use K-fold cross-validation to get more reliable results.

Each time the model runs, a different portion is chosen as the validation set. When the dataset is small, the validation set may be very different each time, so the model is variable. K-fold cross-validation is a method that trains and validates the model k times with each time a different training and validation set. This gives more clarity about the variability of the model; the variability is high if the learning curves of the model show a different pattern during each fold. The average learning curves of the different folds then mediates the variability. That is why k-fold cross-validation is useful when one wants to select the best parameters for a model. We will use k-fold cross-validation to choose between the model without the outliers being removed and the model with the outliers being removed.

Data analysis and preprocessing

After the model was trained, the distribution of the predicted values was plotted (Figure 16 B). This distribution is a bit left-skewed with a high peak at a score of 60. Since this does not reflect the distribution of the real Pawpularity scores (figure 16 A), we will remove the samples with a score of 100. This, because those samples could be outliers and worsen the performance of the model.



A. Distribution of Pawpularity score

B. Distribution of predicted Pawpularity score

Figure 16: Distribution of the Pawpularity score. A. The distribution of the Pawpularity score from the dataset. B. The distribution of the Pawpularity score predicted by our model.

Without k-fold cross-validation, the tabular and image data were split in a training (80%) and validation (20%) set. However, this is not needed when using k-fold cross-validation, so these steps are removed from the data preprocessing. The non-split tabular and image data are used for k-fold. The data is split in x- and y-values, with the x-values being the features in the tabular data and the images in the image data, and the y-values being the 'Pawpularity' scores. The k-fold function shuffles the data and splits the data in k equal parts.

Model pipeline and training

The number of folds (k) is set to 5, so the data is split in 5 parts. Each time the model is being trained, another part is used as the validation set and the remaining 4 parts are used as the training set. Thus, the model is being trained 5 times, with 5 different learning curves. The training and validation MSE are stored and at the end the average training and validation MSE is computed. The average learning curves over the 5 folds are displayed in a graph.

Evaluation and conclusions

The final training and validation MSE of the model that was trained on the data with the outliers using k-fold was 285.30 and 508.39, respectively. When training the model on the data without the outliers, the samples with a Pawpularity score of 100, the final training MSE was 209.08 and validation MSE was 382.99. The training and validation cost of both models are shown in figure 17. Since both the training and validation MSE decreased, the model performs better when the outliers are not present than when the outliers are in the dataset. So, for further versions the outliers should be excluded from the dataset.

Unfortunately, the model still overfits. Therefore, we will add data augmentation in the next chapter. We will first try various kinds of data augmentation without k-fold to save time. Then we will compare the best data augmentation method to this model with k-fold to see how much the model improved.

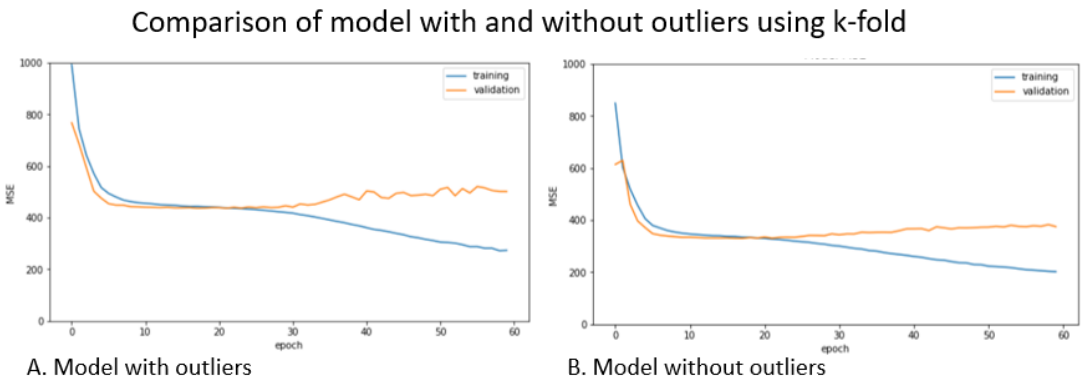


Figure 17: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model before removing the outliers (A) - after removing the outliers (B) for 60 epochs of training.

Chapter 10: Data augmentation

Introduction
The dataset that is used to train the model is rather small. Data augmentation is a method to artificially increase the dataset size. With data augmentation, the images are augmented by for example rotating the images, flipping the images horizontally and vertically, zooming in on images and shifting the colors. These modified images are added to the already existing dataset. The amount of training data increases and this prevents overfitting. The goal of data augmentation is to bring the final training and validation loss closer together.

Data analysis and preprocessing
We tried several data augmentation methods. These are all data preprocessing methods. We tried a rotation range of varying degrees, since pictures that are rotated still have the same features and these features can be extracted from the data. The same is true for a horizontal flip. We also tried a shear range of 0.2, making the images more 'stretched out'. We did not try zoom range, since the proximity of an animal to the camera tends to be a feature that determines the 'Pawpularity' score. As this experiment was done simultaneously to the experiment with removing the outliers (chapter 9), the data still contained the samples with a 'Pawpularity' score of 100.

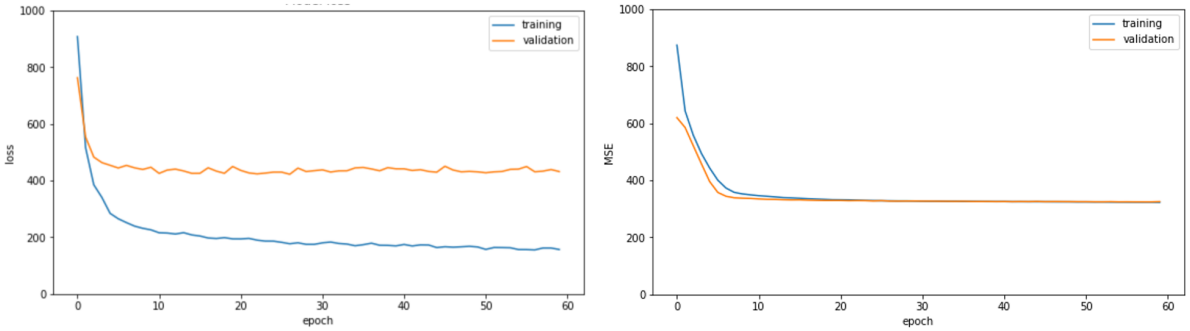
Model pipeline and training
The model is trained on the pile of already existing images and the augmented images. The data is validated on the non-augmented images, because we want to know whether the model performance improves when the variation in training data is higher. We will not use k-fold for this chapter as that will take up too much time with the different types of data augmentations that can be done.

Evaluation and conclusions
In appendix C, the different data augmentation methods with the corresponding final training - and validation MSE and graphs of the learning curves we tried are shown. The results suggest that adding a rotational range causes the model to not overfit anymore, whereas applying shear range leads to an overfitting model, but with a lower training cost.

The goal of data augmentation is to prevent overfitting. That is why we will continue using the following data augmentation methods in further versions: a horizontal flip, a rotation range of 90 degrees and a shear range of 0.2. We choose these methods, because the training MSE and validation MSE are the closest together (426.32 and 425.77, respectively). The previous version of the model had a training MSE of 156.82 and validation MSE of 431.86 after 60 epochs. Since the training MSE increased and the validation MSE decreased, the model does not overfit anymore. The training and validation MSE of both models are displayed in figure 18.

In the next version, we will combine these data augmentation methods with removing outliers while using k-fold cross-validation. With this we attempt to create a model that has a low cost, while not overfitting on the training data.

Comparison of the model loss with vs without data augmentation



A. Without data augmentation

B. With data augmentation

Figure 18: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model after adding regularization terms, before applying data augmentation (A) - after applying data augmentation (B) for 60 epochs of training.

Chapter 11: Data augmentation with removal of outliers

Introduction

In the model from chapter 9, we managed to get a model with low train and validation MSE with the removal of outliers. Nonetheless, this model did overfit. In chapter 10, we managed to get a model that did not overfit by adding data augmentation. However, here, the train and validation MSE were quite high. The goal is to have a model with a low train and validation cost that does not overfit. Hence, for this chapter we will combine the previous two chapters, to see if we can reduce the MSE, while preventing the model from overfitting. We will do this by combining the previous two chapters, so removing the outliers and applying data augmentation.

Data analysis and preprocessing

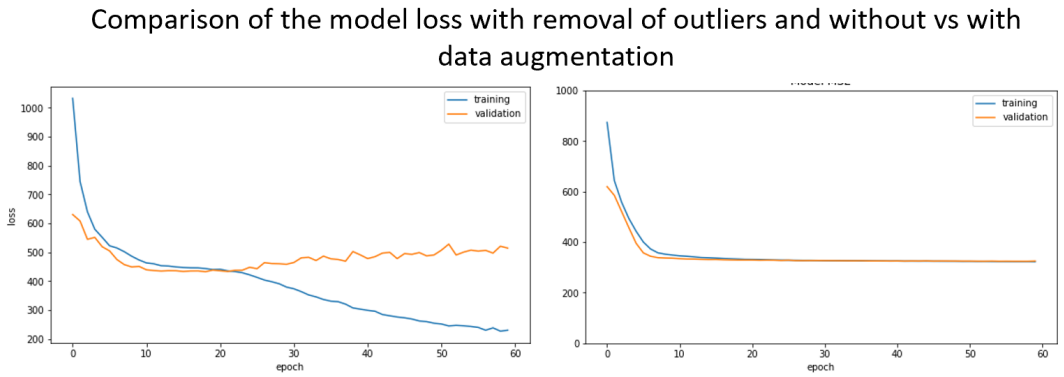
We removed the samples that had a 'Pawpularity' score of 100, as those are the outliers. Moreover, we applied data augmentation to the images. We applied a horizontal flip, a rotation range of 90 degrees and a shear range of 0.2, as this resulted in the least overfitting in the previous chapter.

Model pipeline and training

We used k-fold cross-validation in order to achieve more reliable results. The rest of the model pipeline remained the same as in chapter 9.

Evaluation and conclusions

After removing the outliers and applying data augmentation, the training MSE was 322.87 and the validation MSE was 325.14. The RMSE was 17.79 and 17.86 for training and validation, respectively. When comparing these results to the model from chapter 9, with only removal of outliers, we see that the training MSE increases (was 209.08) and the validation MSE (was 382.99) decreases. The comparison of these two models is depicted in figure 19. In conclusion, the model does not overfit anymore and has a relatively low MSE. Hence, our goal for this model has been achieved. Although the MSE of this model is already quite low, we can still try to reduce this even more. We will do this by adding more complexity to the model, so adding more layers or more hidden nodes. We will also try to make the model less complex to see if it still performs the same.



A. Without data augmentation B. With data augmentation

Figure 19: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model with the removal of the outliers, before applying data augmentation (A) - after applying data augmentation (B) for 60 epochs of training.

Chapter 12: Removing layers in tabular neural network

Introduction

The model from chapter 11 highly improved compared to previous versions. Now that the model does not overfit and performs well, we can try methods to further improve the model. We can make the model more or less complex. When critically looking at the current architecture of the model, it appears that the layers with 20 hidden nodes in the tabular neural network contain very little features compared to the many nodes and features in the convolutional neural network. This tabular neural network is a Dense neural network which analyzes the binary data of the features that are in the images. Besides, the features from the tabular data are not only processed in the tabular neural network itself, but also in the concatenated network. That is why we think that the tabular neural network does not contribute much to learning features from the tabular data and why we will remove the hidden layers in the tabular neural network in this version of the model. This will make sure that our model is not unnecessarily complex, and thus saving time while learning.

Data analysis and preprocessing

Data analysis and preprocessing remain the same in this version.

Model pipeline and training

In previous versions, we created a tabular neural network with three Dense layers. In this version we return to the basic tabular neural network of chapter 1 with just one Dense layer. The input of the Dense layer is the 12 input features in the tabular data and this is spread out over 20 output nodes with no activation function, which are then the input of the concatenated model. An overview of the model is shown in figure 20.

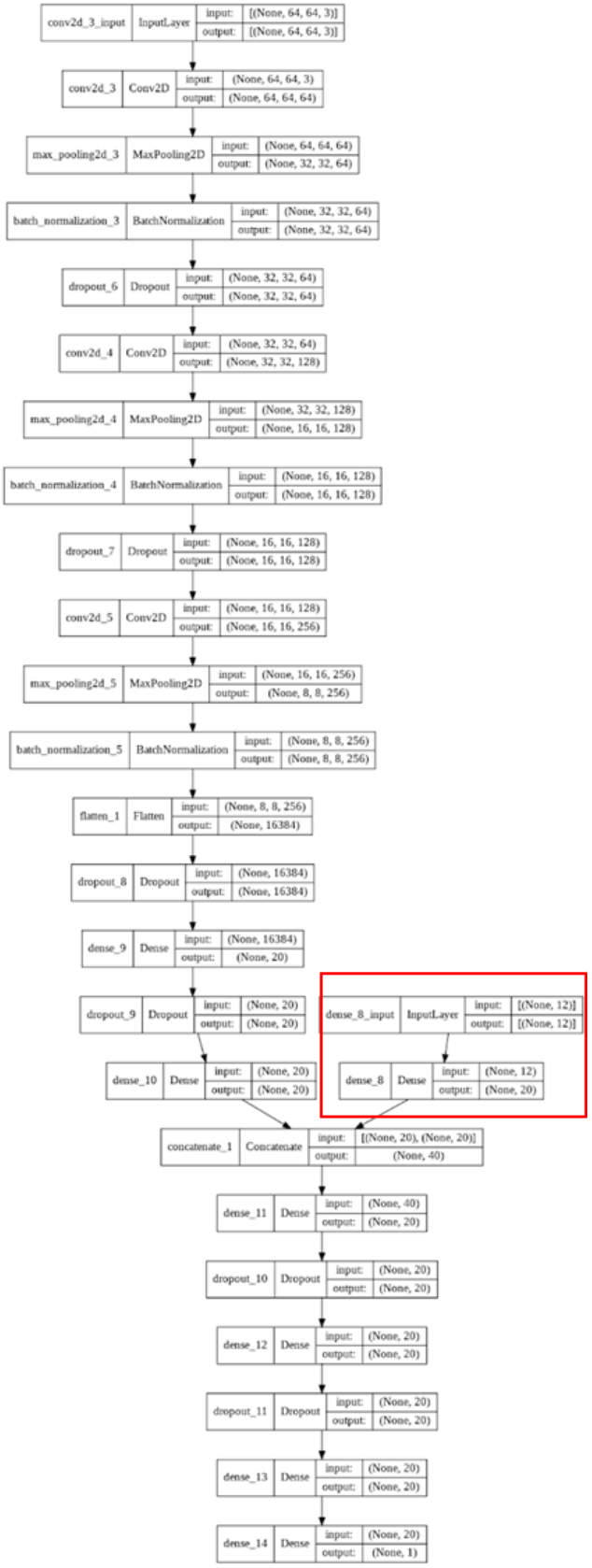


Figure 20: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below). Layers in the tabular network have been removed, as shown with the red rectangle. We will not continue with this network.

Evaluation and conclusions

The final average training MSE was 322.92, the final average training RMSE was 17.78, the final average validation MSE was 325.68 and the final average validation RMSE was 18.04. In the previous model, the train MSE was 322.87 and validation MSE was 325.14, with a RMSE of 17.79 and 17.86. The comparison between learning curves of the previous network and this network is shown in figure 21. Although the increase is small, our model does perform worse when the tabular network is omitted. Hence, we will continue using our previous model, which included the tabular neural network.

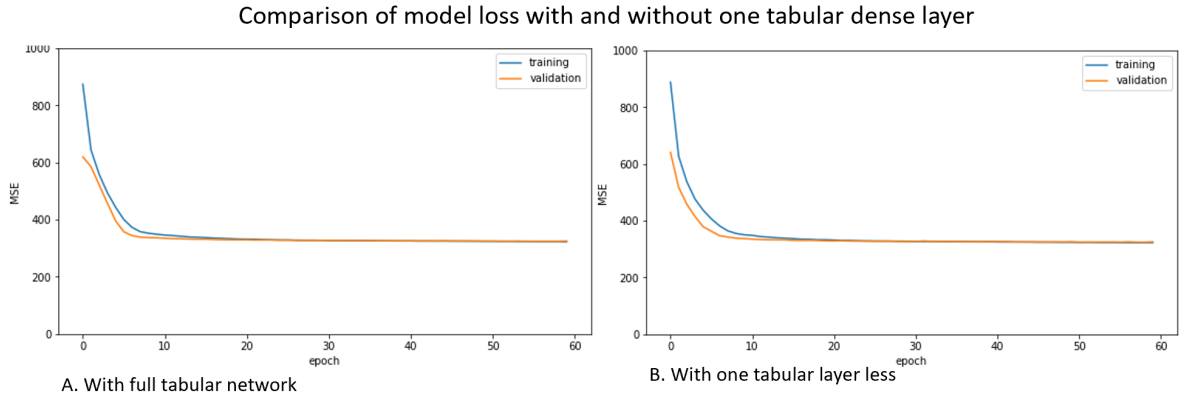


Figure 21: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model with all the layers of the tabular network (A) - with one tabular layer less (B) for 60 epochs of training.

Chapter 13: Extra convolutional layers

Introduction

Since our model does not overfit, we could try to make our model more complex. In this way, the model becomes more flexible. Consequently, the model can better learn the patterns in the data. This could result in better predictions of the model and thus decreasing the MSE. We will make our model more complex by adding extra convolutional layers. Now, there are more parameters which can be used to learn the features in the images. Consequently, the model might be able to perform better.

Data analysis and preprocessing

Data analysis and preprocessing methods remain the same in this version.

Model pipeline and training

After each convolutional layer, we now added one extra convolutional layer. This layer does not have Max pooling or dropout, and contains the amount of filters that are present in the previous layer. We used k-fold cross-validation to obtain more reliable results. Moreover, we tried a model where the last convolutional layer got two extra layers instead of one.

Evaluation and conclusions

For the model where each convolutional layer got one extra layer, the final training MSE was 322.77 after 60 epochs, with a RMSE of 17.78. The final validation MSE was 324.54, with a RMSE of 17.83. This is slightly better than the previous model, which had a train MSE of 322.87 and validation MSE of 325.14. The comparison of these two models is displayed in figure 22.

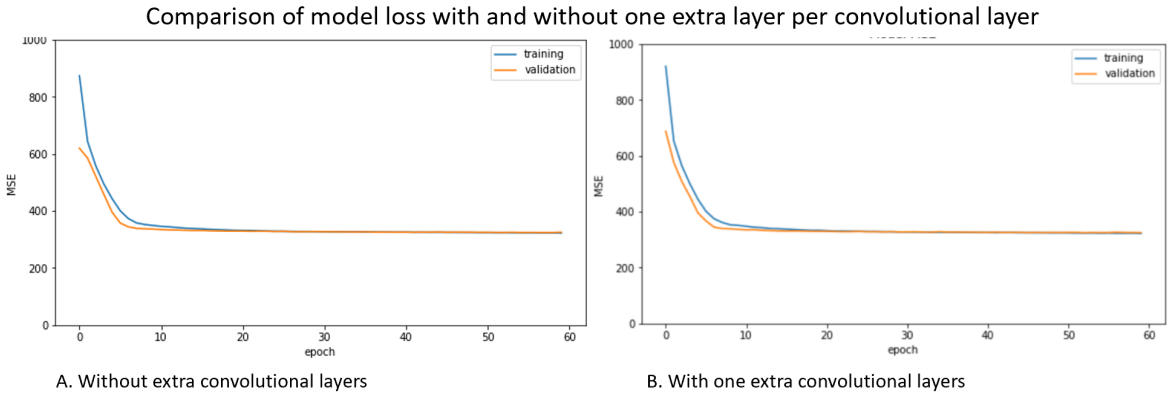
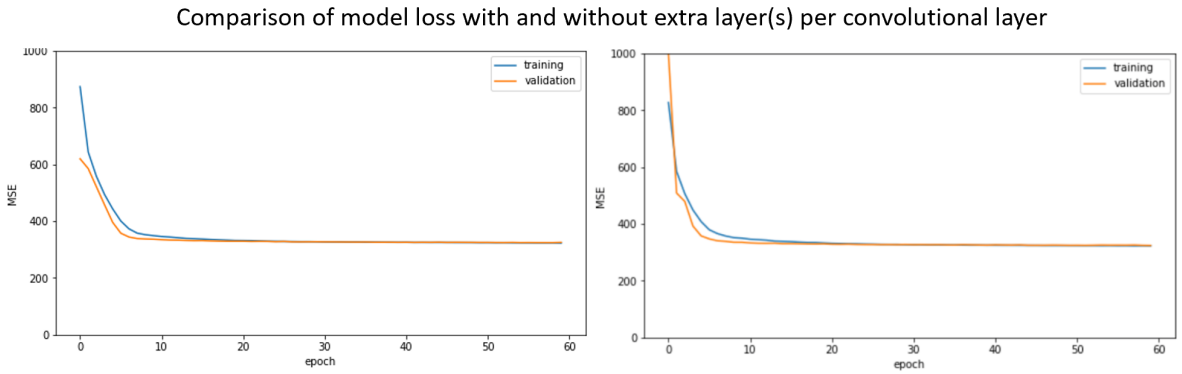


Figure 22: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model without one extra layer per convolutional layer (A) - with one extra layer per convolutional layer (B) for 60 epochs of training.

The next trained model had two extra convolutional layers at the last convolutional layer and one extra layer after the other convolutional layers. The training MSE after 60 epochs for this model was 320.71 and validation MSE was 331.03. The training RMSE after 60 epochs for this model was 17.73 and validation RMSE was 18.03. This is shown in figure 23. The MSE of this model is higher than that of the model with only one extra convolutional layer. Therefore, we will continue with the model that

only has one extra convolutional layer for all the convolutional layers. An overview of this model can be found in figure 24.



A. Without extra convolutional layers

B. With extra convolutional layers

Figure 23: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model without extra layer per convolutional layer (A) - with one extra layer for the first and second convolutional layer and two extra layers for the last convolutional layer (B) for 60 epochs of training.

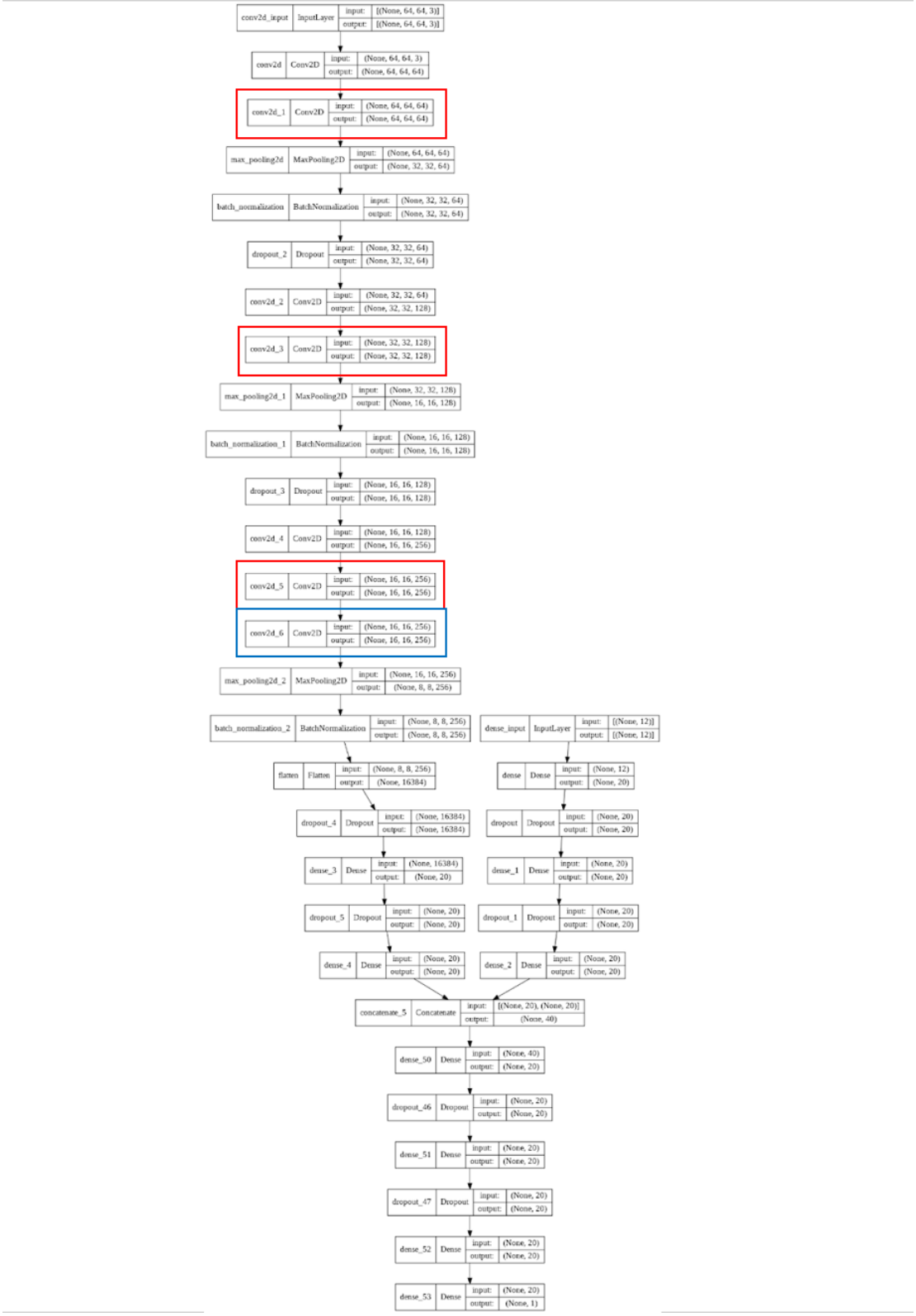


Figure 24: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below). The new layers are highlighted in red. The layer in blue was tested, but we will not use this layer in later versions. This is the network (without the blue layer) we will use in later versions.

Chapter 14: Adding hidden nodes

Introduction

Since our model does not overfit, we could try to make our model more complex. In this way, the model becomes more flexible and the model can better learn the patterns in the data. This could result in better predictions of the model and thus decreasing the MSE. We will add more hidden nodes in the convolutional dense layers and in the concatenated dense layers. In this way, there are more parameters which can be used to learn the features in the images. Consequently, the model might be able to perform better.

Data analysis and preprocessing

Data analysis and preprocessing methods remain the same in this version.

Model pipeline and training

As this experiment was done simultaneously to the experiment of chapter 13, we increased the number of hidden nodes in the model of chapter 12. So, there are no extra convolutional layers in this model. We did not increase the hidden nodes of the tabular neural network, as this network only contained 12 input features, and if we would increase the number of hidden nodes it would not be in proportion to the input nodes. We increased the number of nodes in the hidden dense layers of the convolutional neural network from 20 to 100, as this dense neural network has a lot of input nodes from the convolutional part of the network. Moreover, we increased the number of hidden nodes from the dense concatenated neural network from 20 to 40 nodes.

Evaluation and conclusions

After increasing the number of nodes, the final training MSE was 322.07 and training RMSE was 17.78. The final validation MSE was 324.19 and RMSE was 17.84. When comparing this to the model with no increase in hidden nodes (from chapter 11), the validation RMSe was 17.86 (figure 25). So, this model performs slightly better. Since both this model and the model with the extra convolutional layers performed better than the model from chapter 11, in the next chapter we will check whether a model with both extra layers and extra hidden nodes performs even better.

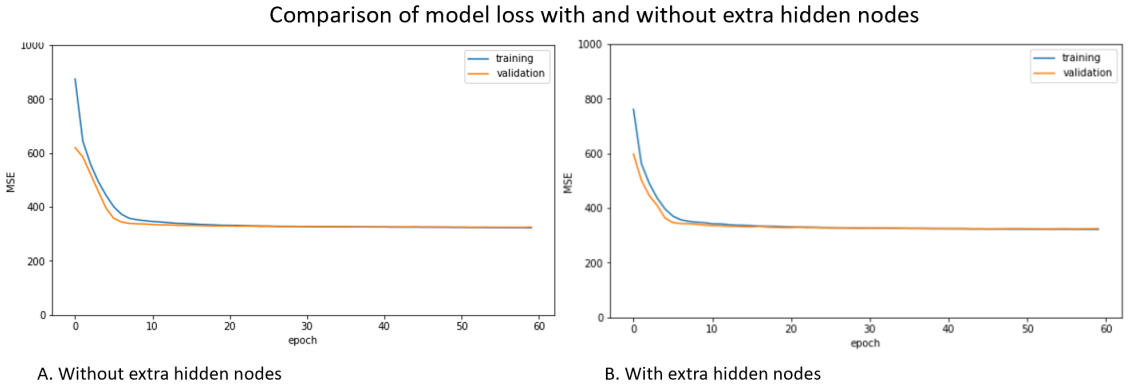


Figure 25: Graph showing the comparison of the MSE (mean squared error) of the training (blue) and validation (orange) of the model without extra hidden nodes (A) - with extra hidden nodes (B) for 60 epochs of training.

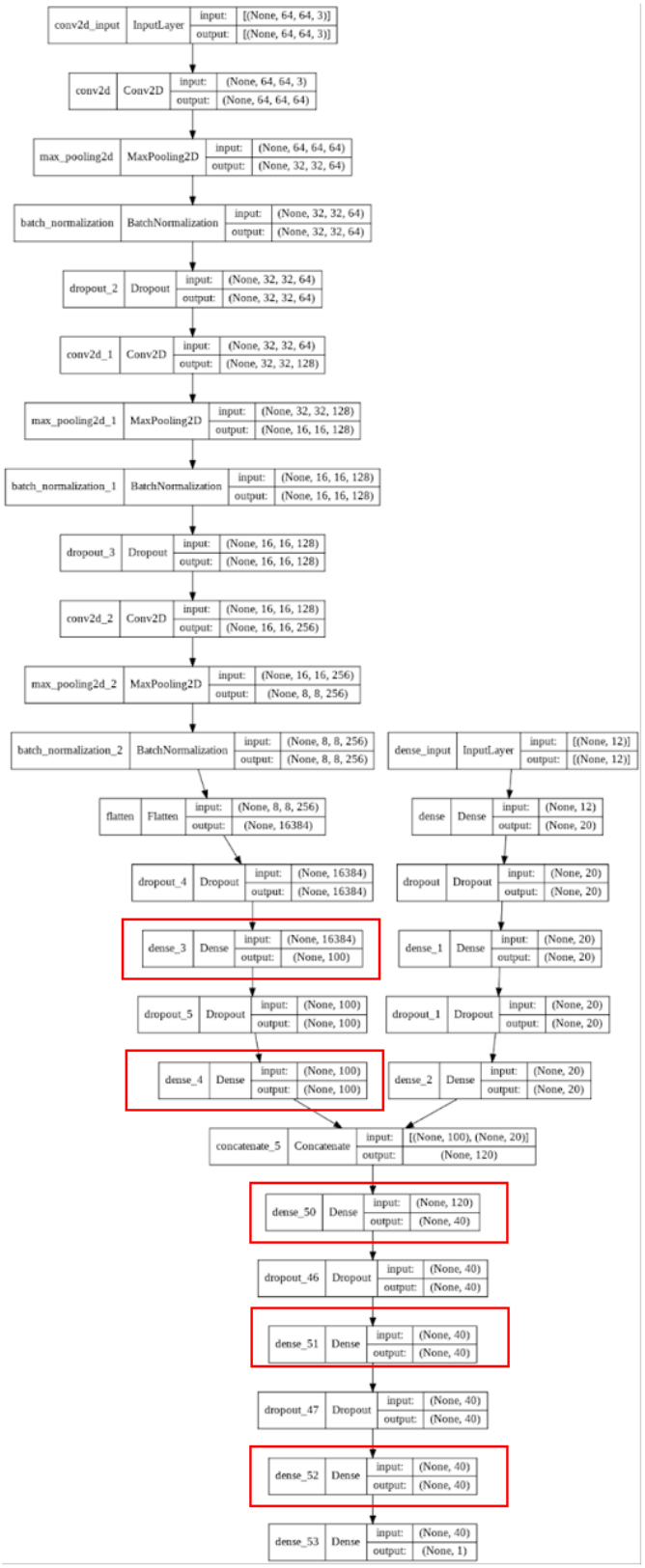
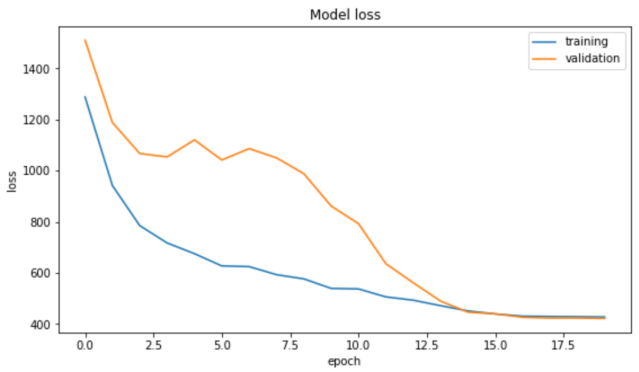
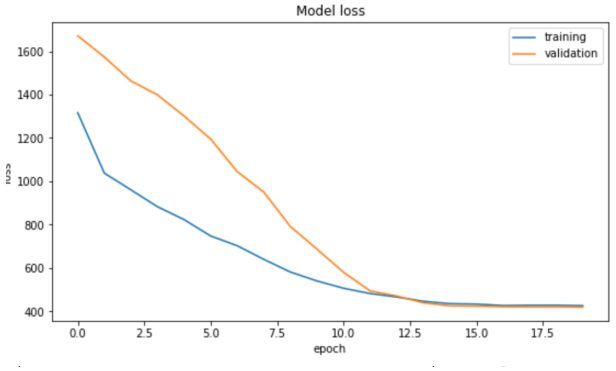
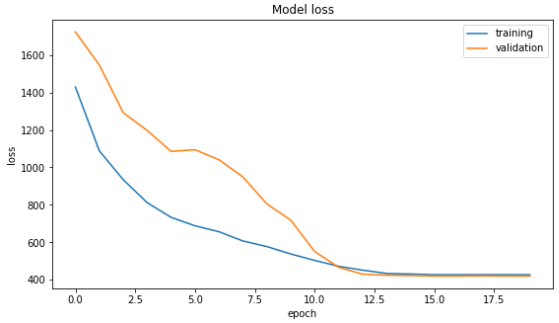
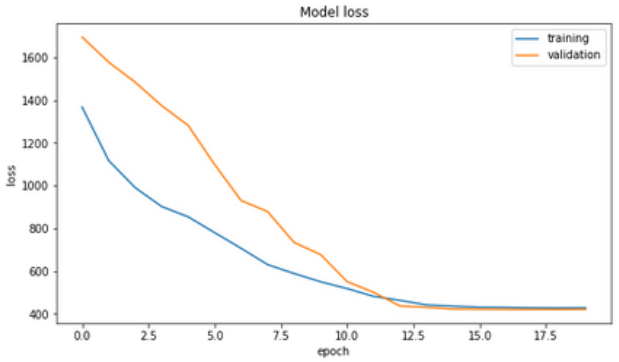
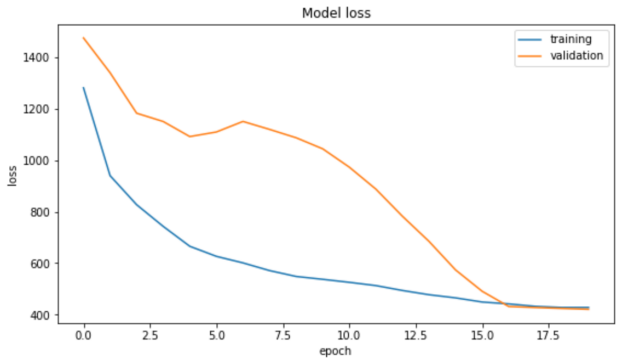
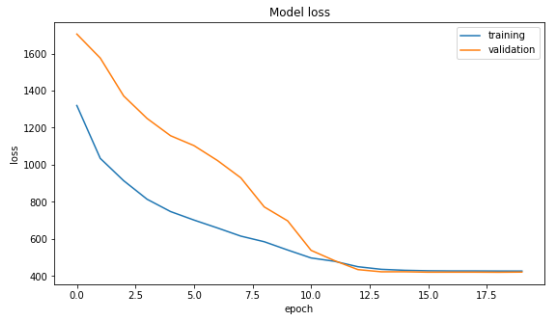
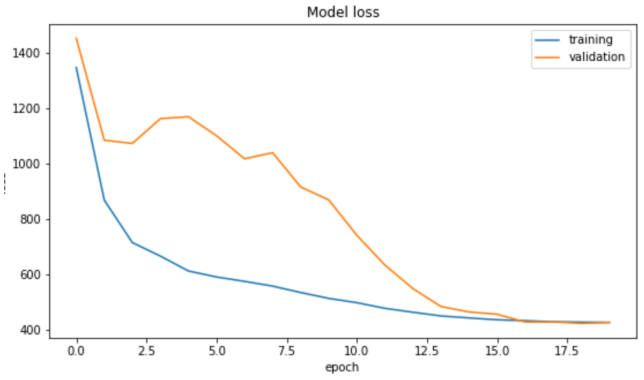
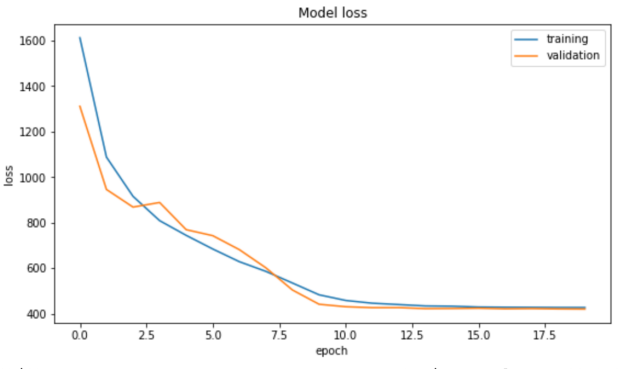
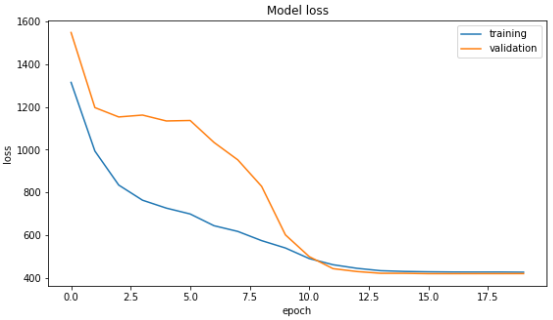


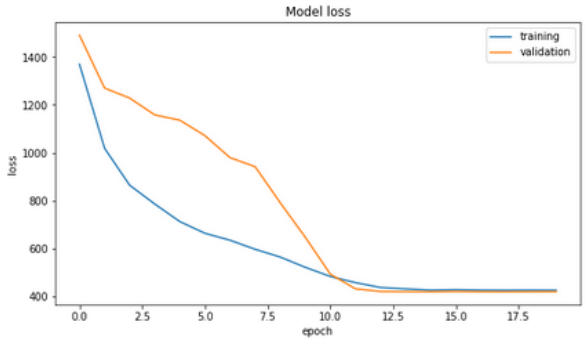
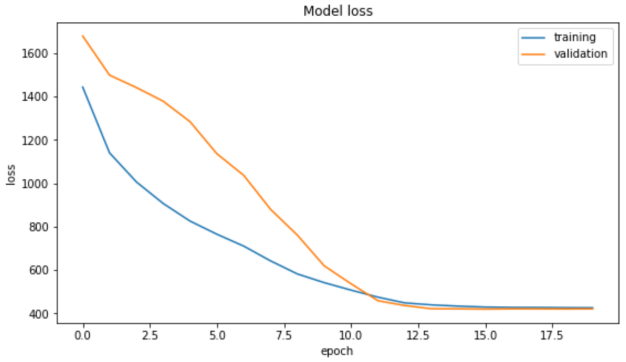
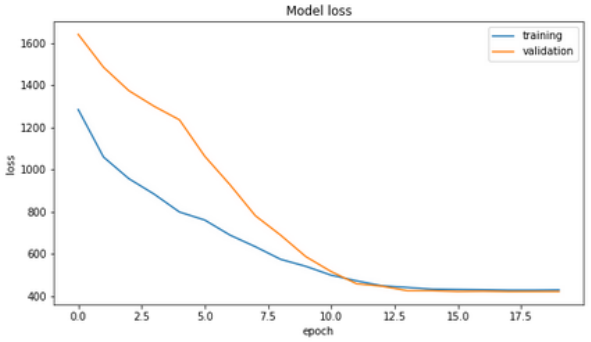
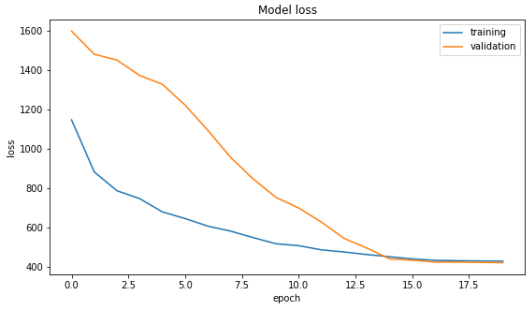
Figure 26 : Overview of the model with the extra hidden nodes in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below). The layers with the adjusted hidden nodes are highlighted in red.

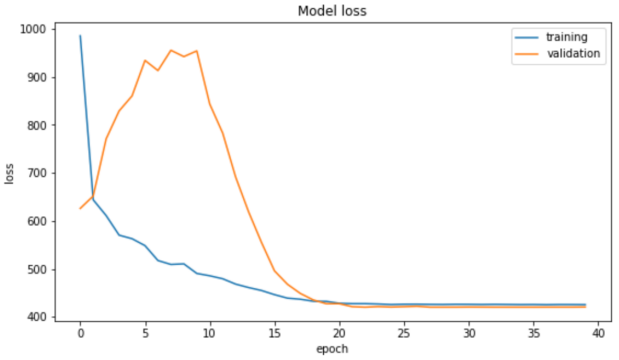
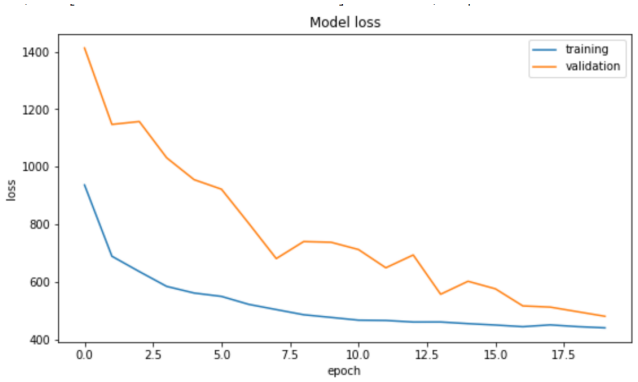
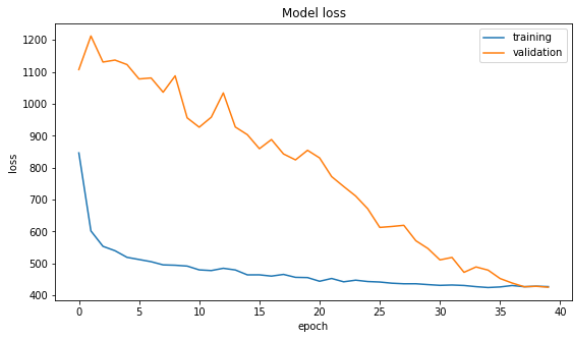
Appendix A: Model training and validation MSE for the combinations tried to decrease the dropout. The left column shows to which layers which dropout probability is added, the second column the model MSE (MSE) of training and validation data and the third column the learning curve of the model. The y-axis of the model MSE graphs vary.

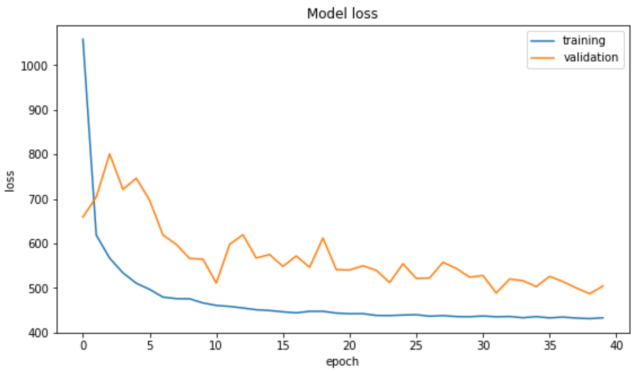
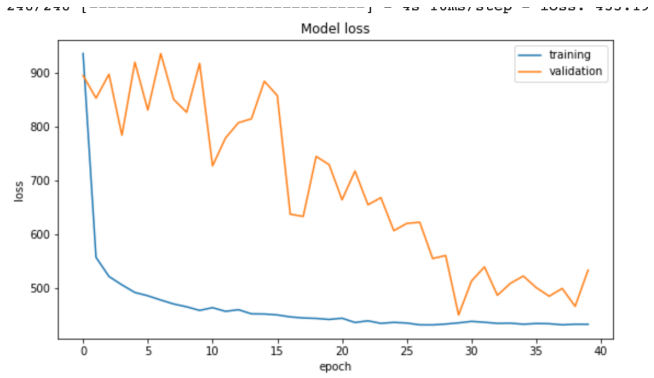
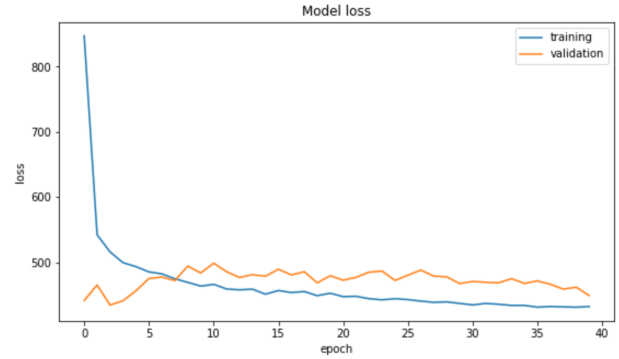
Layers	Model MSE (MSE)	Model MSE graph
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.7	Training MSE 427.39 Validation MSE 421.44	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.7 Hidden layer 2: 0.8	Training MSE 426.84 Validation MSE 420.23	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.7 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training MSE 427.08 Validation MSE 420.13	

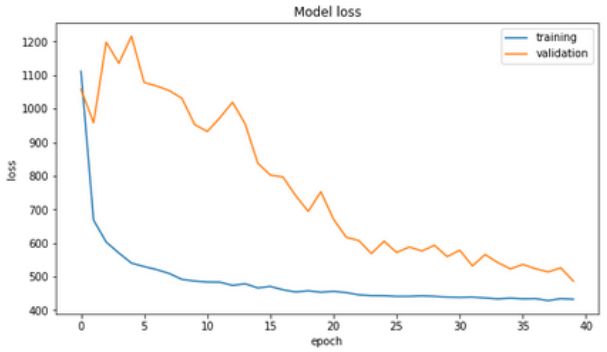
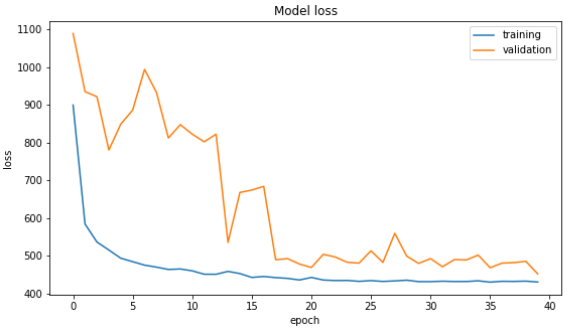
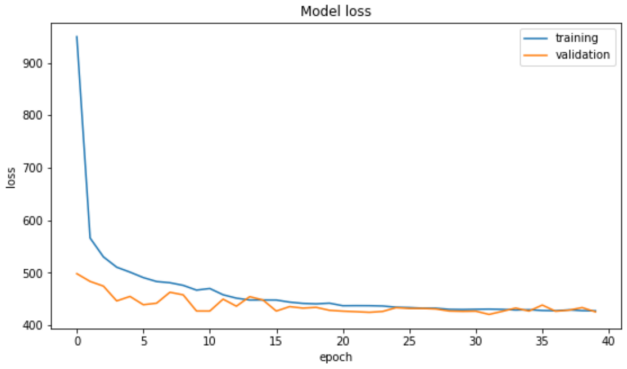
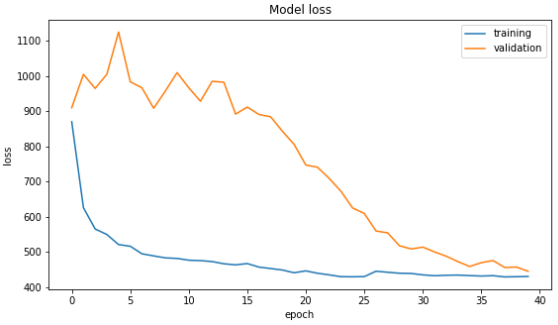
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.7 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Train MSE 427.81 Validation MSE 420.66	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.6 Hidden layer 2: 0.8	Training MSE 427.96 Validation MSE 420.84	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.6 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training MSE 426.75 Validation MSE 421.12	

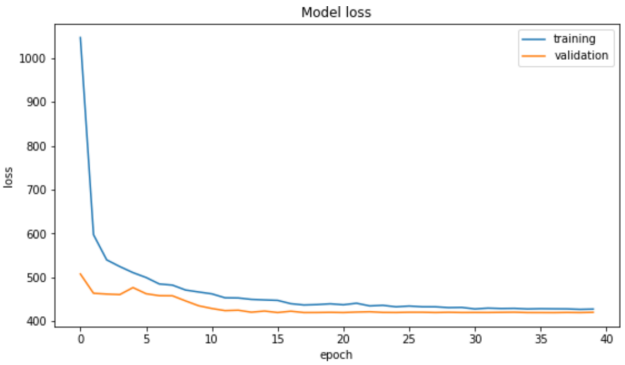
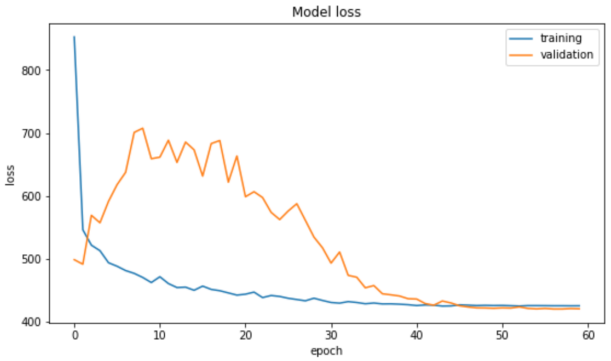
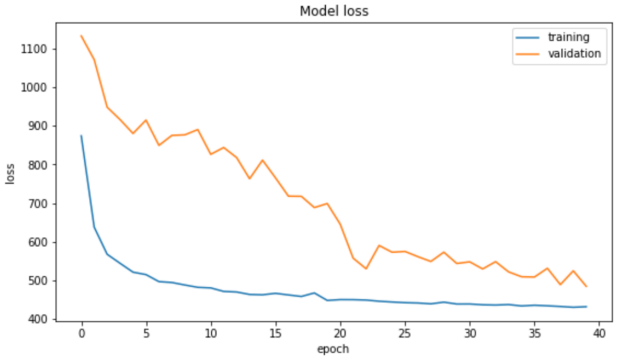
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.6 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.6	Training MSE 427.87 Validation MSE 427.61	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.6 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training MSE 427.14 Validation MSE 420.22	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.6 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training MSE 426.98 Validation MSE 420.41	

Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.6 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training MSE 426.49 Validation MSE 420.75	
Tabular Dense layer 1: 0.6 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training MSE 426.33 Validation MSE 421.38	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.6 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training MSE 428.17 Validation MSE 420.38	
Tabular Dense layer 1: 0.7 Dense layer 2: 0.7 Convolutional Conv2D layer 1: 0.7 Conv2D layer 2: 0.7 Dense layer 1: 0.7 Dense layer 2: 0.7	Training MSE 427.55 Validation MSE 420.15	

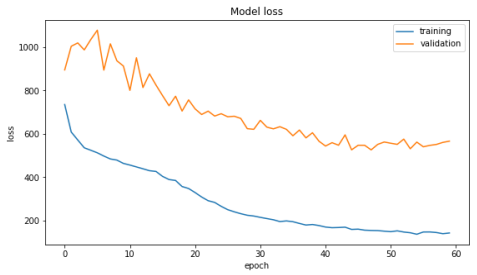
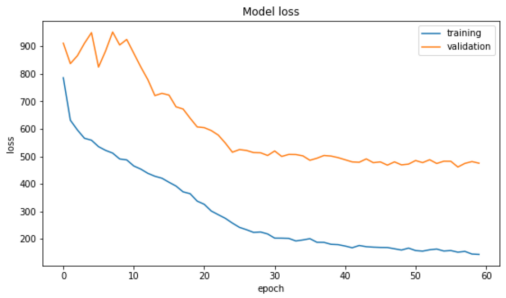
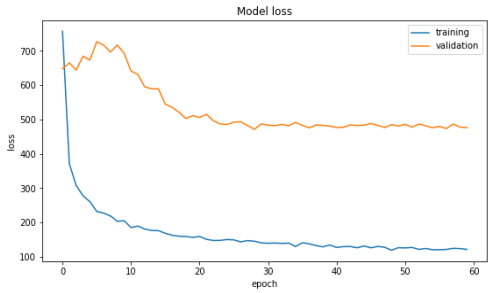
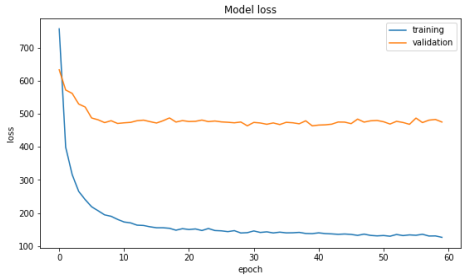
<u>Concatenate</u> Hidden layer 1: 0.7 Hidden layer 2: 0.7		
<u>Tabular</u> Dense layer 1: 0.6 Dense layer 2: 0.6 <u>Convolutional</u> Conv2D layer 1: 0.6 Conv2D layer 2: 0.6 Dense layer 1: 0.6 Dense layer 2: 0.6 <u>Concatenate</u> Hidden layer 1: 0.6 Hidden layer 2: 0.6	<u>Training MSE</u> 425.39 <u>Validation MSE</u> 420.45	
<u>Tabular</u> Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Convolutional</u> Conv2D layer 1: 0.5 Conv2D layer 2: 0.5 Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Concatenate</u> Hidden layer 1: 0.5 Hidden layer 2: 0.5	<u>Training MSE</u> 440.42 <u>Validation MSE</u> 480.54	
<u>Tabular</u> Dense layer 1: 0.4 Dense layer 2: 0.4 <u>Convolutional</u> Conv2D layer 1: 0.4 Conv2D layer 2: 0.4 Dense layer 1: 0.4 Dense layer 2: 0.4 <u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4	<u>Training MSE</u> 427.62 <u>Validation MSE</u> 425.83	

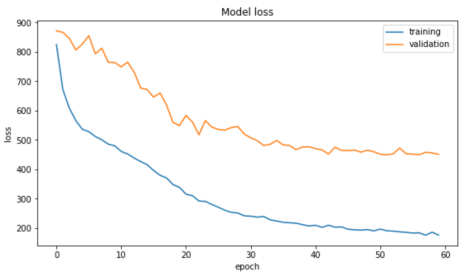
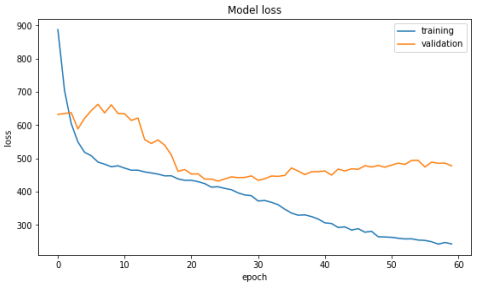
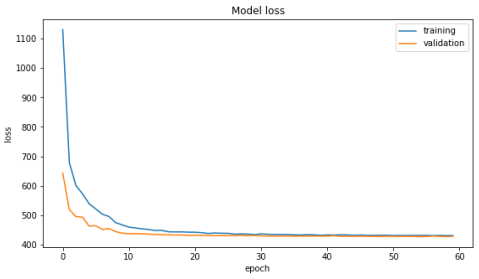
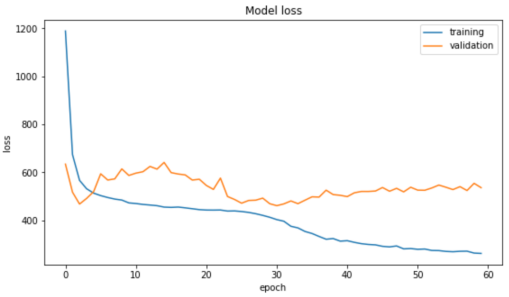
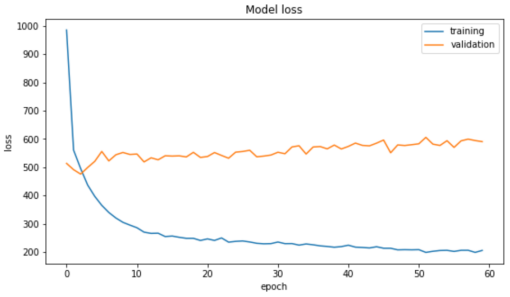
Tabular Dense layer 1: 0.5 Dense layer 2: 0.4 Convolutional Conv2D layer 1: 0.5 Conv2D layer 2: 0.5 Dense layer 1: 0.4 Dense layer 2: 0.4 Concatenate Hidden layer 1: 0.3 Hidden layer 2: 0.3	Training MSE 433.03 Validation MSE 504.42	
Tabular Dense layer 1: 0.4 Dense layer 2: 0.4 Convolutional Conv2D layer 1: 0.4 Conv2D layer 2: 0.4 Dense layer 1: 0.4 Dense layer 2: 0.4 Concatenate Hidden layer 1: 0.2 Hidden layer 2: 0.2	Training MSE 433.20 Validation MSE 533.93	
Tabular Dense layer 1: 0.4 Dense layer 2: 0.4 Convolutional Conv2D layer 1: 0.4 Conv2D layer 2: 0.4 Dense layer 1: 0.4 Dense layer 2: 0.4 Concatenate Hidden layer 1: 0.3 Hidden layer 2: 0.3	Training MSE 432.48 Validation MSE 449.33	

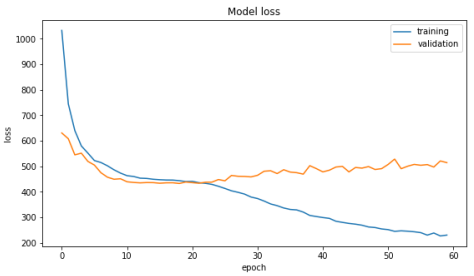
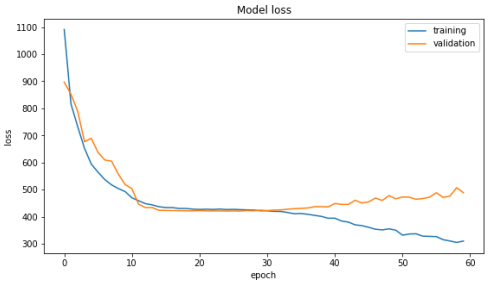
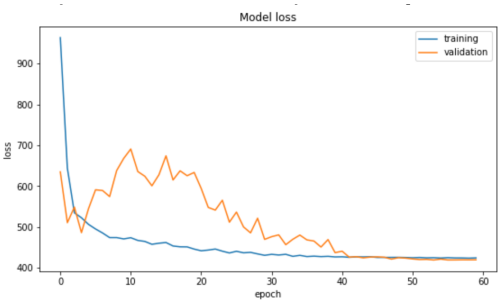
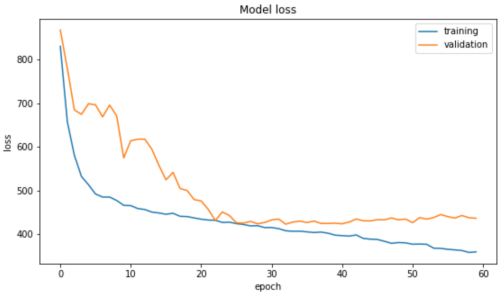
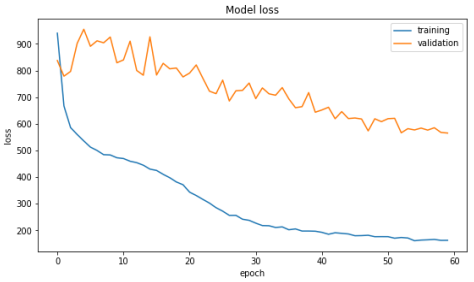
<u>Tabular</u> Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Convolutional</u> Conv2D layer 1: 0.5 Conv2D layer 2: 0.5 Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4	<u>Training MSE</u> 432.71 <u>Validation MSE</u> 486.46	
<u>Tabular</u> Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Convolutional</u> Conv2D layer 1: 0.5 Conv2D layer 2: 0.5 Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Concatenate</u> Hidden layer 1: 0.3 Hidden layer 2: 0.3	<u>Training MSE</u> 430.19 <u>Validation MSE</u> 451.66	
<u>Tabular</u> Dense layer 1: 0.4 Dense layer 2: 0.4 <u>Convolutional</u> Conv2D layer 1: 0.4 Conv2D layer 2: 0.4 Dense layer 1: 0.2 Dense layer 2: 0.2 <u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4	<u>Training MSE</u> 427.59 <u>Validation MSE</u> 425.60	
<u>Tabular</u> Dense layer 1: 0.4 Dense layer 2: 0.4 <u>Convolutional</u> Conv2D layer 1: 0.2 Conv2D layer 2: 0.2 Dense layer 1: 0.4 Dense layer 2: 0.4	<u>Training MSE</u> 429.78 <u>Validation MSE</u> 444.34	

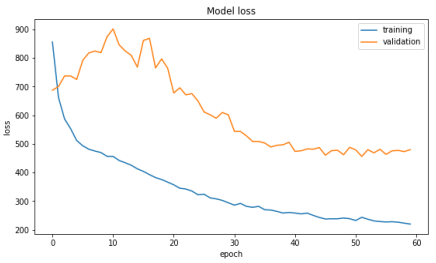
<u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4		
<u>Tabular</u> Dense layer 1: 0.4 Dense layer 2: 0.4 <u>Convolutional</u> Conv2D layer 1: 0.4 Conv2D layer 2: 0.4 Dense layer 1: 0.3 Dense layer 2: 0.3 <u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4	<u>Training MSE</u> 427.99 <u>Validation MSE</u> 420.66	
<u>Tabular</u> Dense layer 1: 0.4 Dense layer 2: 0.4 <u>Convolutional</u> Conv2D layer 1: 0.4 Conv2D layer 2: 0.4 Dense layer 1: 0.3 Dense layer 2: 0.3 <u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4 <u>Epochs</u> 60	<u>Training MSE</u> 425.17 <u>Validation MSE</u> 420.33	
<u>Tabular</u> Dense layer 1: 0.3 Dense layer 2: 0.3 <u>Convolutional</u> Conv2D layer 1: 0.4 Conv2D layer 2: 0.4 Dense layer 1: 0.4 Dense layer 2: 0.4 <u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4	<u>Training MSE</u> 432.06 <u>Validation MSE</u> 484.75	

Appendix B: Model training and validation MSE when adding regularization terms. The left column shows the regularization terms that are added, the second column the model MSE (MSE) of training and validation data and the third column the learning curve of the model. The y-axis of the model MSE graphs vary.

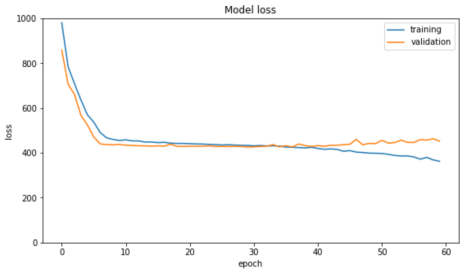
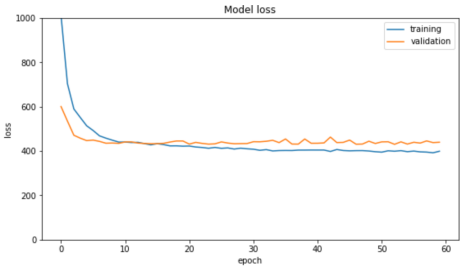
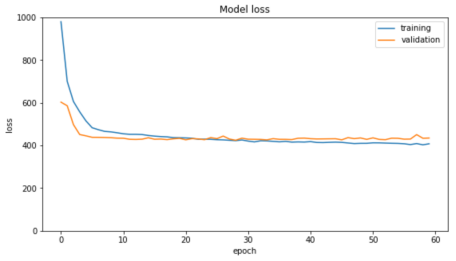
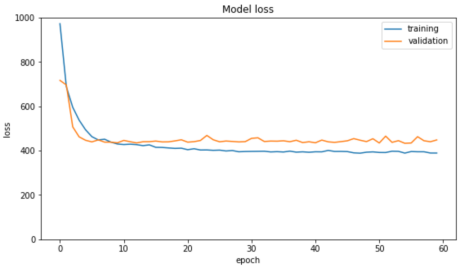
Regularization	**Model MSE (MSE)**	**Model MSE graph**
Concatenated layers Kernel regularizer: L2 (ƛ=1e-4) Bias regularizer: L2 (ƛ=1e-4) Activity regularizer: L2 (ƛ=1e-5)	Training MSE 142.23 Validation MSE 566.46	
Concatenated layers Kernel regularizer: L2 (ƛ=1e-3) Bias regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3)	Training MSE 143.76 Validation MSE 475.27	
Concatenated layers Kernel regularizer: L2 (ƛ=1e-2) Bias regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2)	Training MSE 121.60 Validation MSE 476.53	
Concatenated layers Kernel regularizer: L2 (ƛ=1e-1) Bias regularizer: L2 (ƛ=1e-1) Activity regularizer: L2 (ƛ=1e-1)	Training MSE 126.76 Validation MSE 475.16	

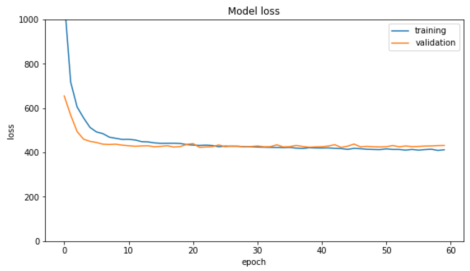
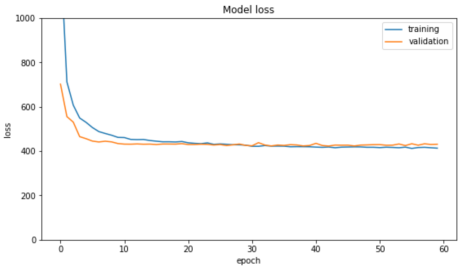
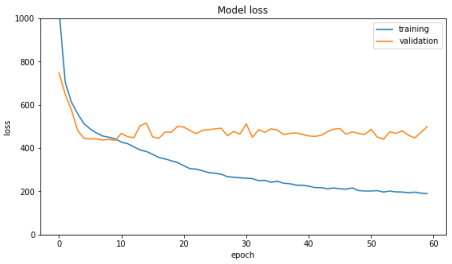
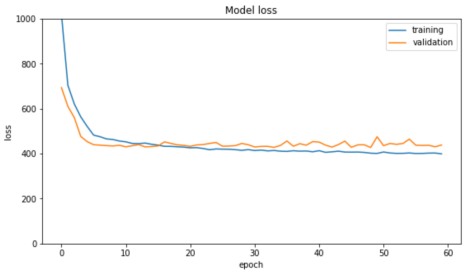
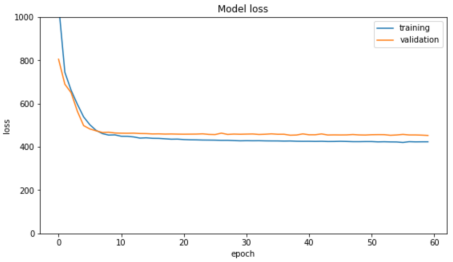
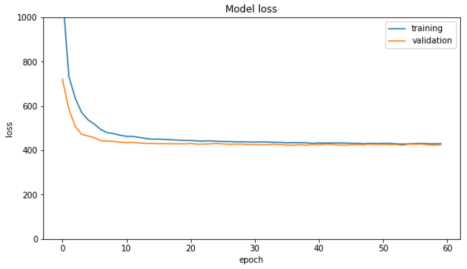
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-3) Bias regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-3) Bias regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3)	<u>Training MSE</u> 175.96 <u>Validation MSE</u> 451.23	
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-2) Bias regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-2) Bias regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2)	<u>Training MSE</u> 242.83 <u>Validation MSE</u> 477.41	
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-1) Bias regularizer: L2 (ƛ=1e-1) Activity regularizer: L2 (ƛ=1e-1) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-1) Bias regularizer: L2 (ƛ=1e-1) Activity regularizer: L2 (ƛ=1e-1)	<u>Training MSE</u> 430.36 <u>Validation MSE</u> 427.87	
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-1) Bias regularizer: L2 (ƛ=1e-1) Activity regularizer: L2 (ƛ=1e-1) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-3) Bias regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3)	<u>Training MSE</u> 262.07 <u>Validation MSE</u> 536.07	
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-1) Bias regularizer: L2 (ƛ=1e-1) Activity regularizer: L2 (ƛ=1e-1) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-2) Bias regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2)	<u>Training MSE</u> 205.78 <u>Validation MSE</u> 590.64	

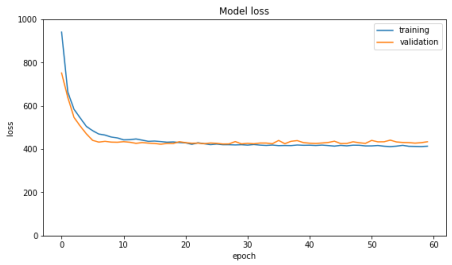
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-3) Bias regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-1) Bias regularizer: L2 (ƛ=1e-1) Activity regularizer: L2 (ƛ=1e-1)	<u>Training MSE</u> 230.49 <u>Validation MSE</u> 514.46	
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-2) Bias regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-2) Bias regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2) <u>Dropout</u> Concatenated layers: 0.6	<u>Training MSE</u> 310.57 <u>Validation MSE</u> 488.90	
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-1) Activity regularizer: L2 (ƛ=1e-1) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3)	<u>Training MSE</u> 424.51 <u>Validation MSE</u> 419.86	
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2)	<u>Training MSE</u> 359.52 <u>Validation MSE</u> 436.45	
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-2) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-2)	<u>Training MSE</u> 162.95 <u>Validation MSE</u> 565.34	

<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3)	<u>Training MSE</u> 220.07 <u>Validation MSE</u> 479.80	

Appendix C: Model training and validation MSE when adding data augmentation methods. The first column shows the tried data augmentation methods, the second column shows the training and validation MSE of the model after 60 epochs and the third column shows the learning curves of the model during training the model. Some augmentations result in a model that does not overfit anymore, whereas other augmentations do not.

Augmentation methods	**Model MSE (MSE)**	**Model MSE graph**
Horizontal flip: True	<u>Training MSE:</u> 362.53 <u>Validation MSE:</u> 452.63	
Rotation range: 50 degrees Horizontal flip: True	<u>Training MSE:</u> 399.09 <u>Validation MSE:</u> 439.70	
Rotation range: 60 degrees	<u>Training MSE:</u> 407.90 <u>Validation MSE:</u> 434.88	
Rotation range: 60 degrees Horizontal flip: True	<u>Training MSE:</u> 389.25 <u>Validation MSE:</u> 448.35	

Rotation range: 70 degrees	<u>Training MSE:</u> 412.13 <u>Validation MSE:</u> 431.68	
Rotation range: 70 degrees Horizontal flip: True	<u>Training MSE:</u> 413.26 <u>Validation MSE:</u> 430.90	
Shear range: 0.2	<u>Training MSE:</u> 189.58 <u>Validation MSE:</u> 499.56	
Rotation range: 70 degrees Shear range: 0.2	<u>Training MSE:</u> 399.07 <u>Validation MSE:</u> 438.49	
Rotation range: 70 degrees Shear range: 0.2 Horizontal flip: True	<u>Training MSE:</u> 423.53 <u>Validation MSE:</u> 452.83	
Rotation range: 90 degrees	<u>Training MSE:</u> 430.56 <u>Validation MSE:</u> 427.23	

Rotation range: 90 degrees Horizontal flip: True	<u>Training MSE:</u> 413.26 <u>Validation MSE:</u> 433.95	
Rotation range: 90 degrees Shear range: 0.2 Horizontal flip: True	<u>Training MSE:</u> 426.32 <u>Validation MSE:</u> 425.77