Report “Pawpularity”

Machine Learning Project

Cass Maes
Demi Soetens
Bente van Katwijk
Ilse Feenstra

07-02-2022

Index

Chapter 1: Basic model

Introduction

In western countries, owning pets is tremendously popular and pet owners would do anything to protect their pets. However, this is not so common everywhere around 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 Cuteness Meter to rank their pet photos. This tool is still quite basic, and it could be optimized by using artificial intelligence tools. The tool can be improved by predicting 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 9912 raw images and tabular data from the pet profiles of PetFinder.my to create a model that is able to accurately predict the Pawpularity of pet photos. An example of a photo is given in figure 1. 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. All features are presented in table 1. Using these features, the images and the given Pawpularity scores, the model will be able to predict this score for our test images.

Figure 1: An image of the train image data

Table 1: Photo Metadata. The train.csv and test.csv files contain metadata for photos in the training set and test set, respectively. Each pet photo is labeled with the value of 1 (Yes) or 0 (No) for each of the features presented below. Taken from https://www.kaggle.com/c/petfinder-pawpularity-score/data.

Feature	**Description**
Focus	Pet stands out against an uncluttered background, not too close / far.
Eyes	Both eyes are facing front or near-front, with at least 1 eye / pupil decently clear.
Face	Decently clear face, facing front or near-front.
Near	Single pet taking up a significant portion of photo (roughly over 50% of photo width or height).
Action	Pet in the middle of an action (e.g., jumping).
Accessory	Accompanying physical or digital accessory / prop (i.e. toy, digital sticker), excluding collar and leash.
Group	More than 1 pet in the photo.
Collage	Digitally-retouched photo (i.e. with digital photo frame, combination of multiple photos).
Human	Human in the photo.
Occlusion	Specific undesirable objects blocking part of the pet (i.e. human, cage or fence). Note that not all blocking objects are considered occlusion.
Info	Custom-added text or labels (i.e. pet name, description).
Blur	Noticeably out of focus or noisy, especially for the pet's eyes and face. For Blur entries, "Eyes" column is always set to 0.

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.

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 2). The distribution of the data was slightly right-skewed with the peak at 30. The mean was 38.04 and the standard deviation was 20.59. Noticeably was the high occurrence at a score of 100. Possibly, it would be better to remove these samples, which we will investigate in further versions.

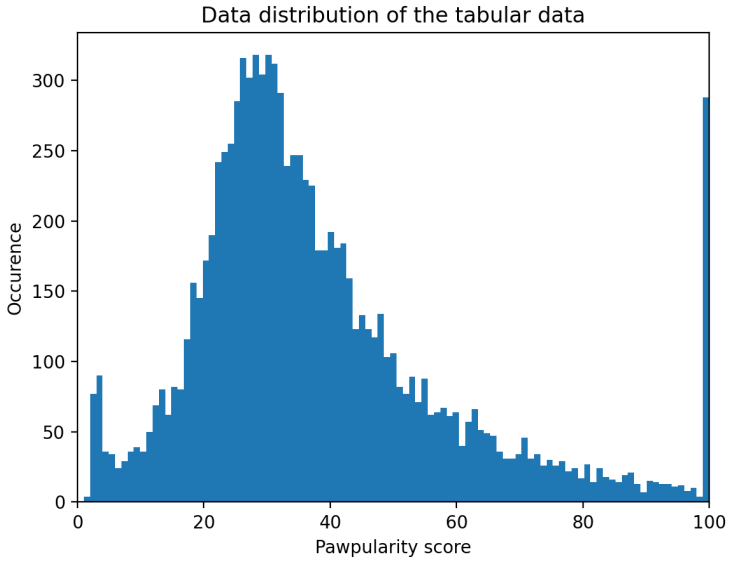


Figure 2: Histogram of the data distribution of the tabular data. On the x-axis is the Pawpularity score presented and on the y-axis the occurrence of each score. The graph is slightly right-skewed with an outlier at the score of 100.

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 the combined data in our concatenated neural network, as will be explained in the model pipeline section.

Thirdly, we splitted the training data into a training data set (80%) and a validation data set (20%). In order to do this, we started by shuffling the image and tabular data for the reason that the validation and training data could 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 were certain that the same image and tabular data was selected for the validation data. We chose the first 20% of the sorted data as validation data and the other 80% as training data.

Model pipeline and training

An overview of the model with all the layers is shown in figure 3. At first, we started with two separate models, a Convolutional Neural Network (CNN) for the images and a Dense Neural Network (DNN) for the tabular data. The input of the CNN was the reshaped images (64 x 64). We chose to implement a CNN, because this type of model is suited for image data as spatial information in the images remains preserved. We added two convolutional layers, where the first one has 64 filters and the second one 128. The kernel size for both is 3x3 and the convolutional layers have a ReLU activation function. After each of these layers we added a maximum pooling layer with a pool size of 2x2 and a stride of 2. After a flattening layer we added a Dense layer with 20 hidden nodes. This layer has a ReLU activation function as well.

The input of the DNN 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 tabular neural network contained 1 Dense layer of 20 nodes and a ReLU activation function. We chose a DNN for the tabular data, since this type of model 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 into a concatenated neural network. The model has the tabular neural network output and the CNN 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 80% training data and 20% validation data.

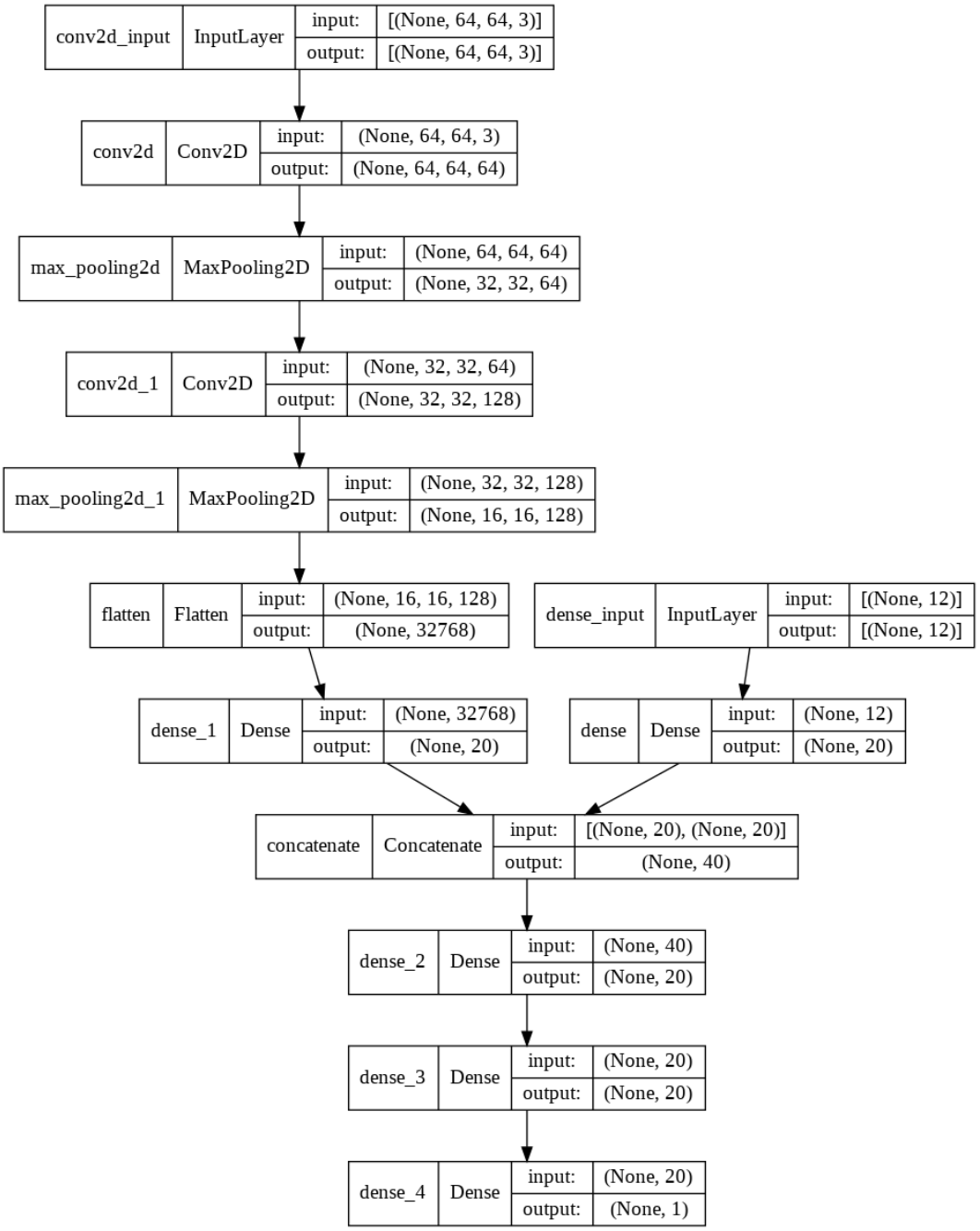


Figure 3: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below).

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

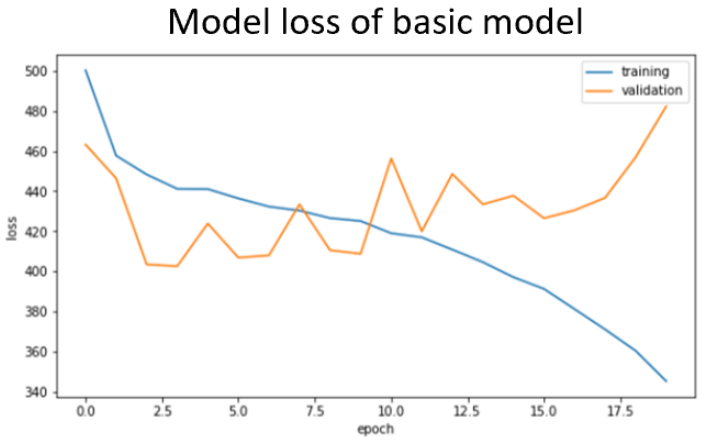


Figure 4: Graph showing the training (blue) and validation (orange) MSE (Mean Squared Error) of the concatenated model for 20 epochs.

The model did seem to overfit on the training data, since the training MSE was decreasing, while the validation MSE was increasing. This means that the model was fitting the training data too specifically and 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, is the Pawpularity score and ranges from 0 to 100. In later versions we could set a minimum and maximum 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 performing well on our training data and not on 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. Consequently, the model will 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 these methods to both the training and validation data, so the training and validation data are still comparable. We only applied the preprocessing methods to the image data, since the tabular data is binary and does not need mean centering and standard deviation normalization.

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 on just the CNN for the image data. We compared the MSE values of the concatenated network with the CNN and examined if the concatenated model indeed performed better than the CNN alone.

Evaluation and conclusions

Figure 5 shows the comparison of the concatenated model MSE and the CNN MSE with the training MSE in blue and the validation MSE in orange. After 20 epochs, the concatenated model had a final training MSE of 105.89 and a final validation MSE of 593.03. The CNN had a final training MSE of 179.02 and a final validation MSE of 603.06 after 20 epochs. As the concatenated model had a lower validation MSE we concluded that the concatenated model is performing better than the CNN.

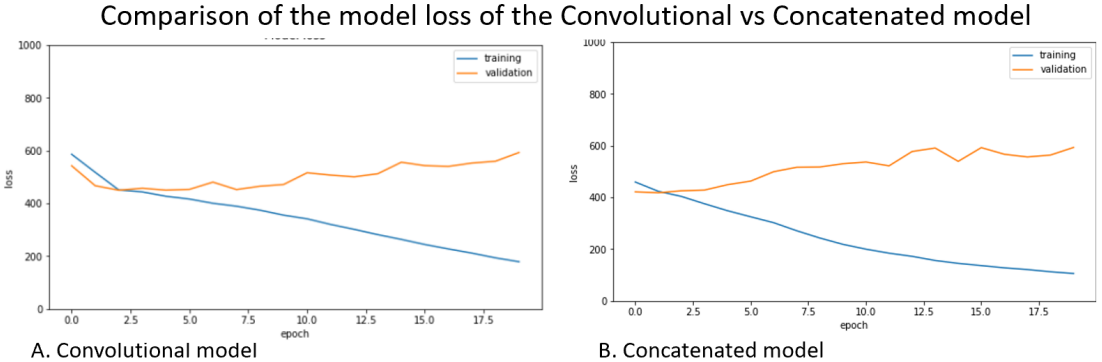
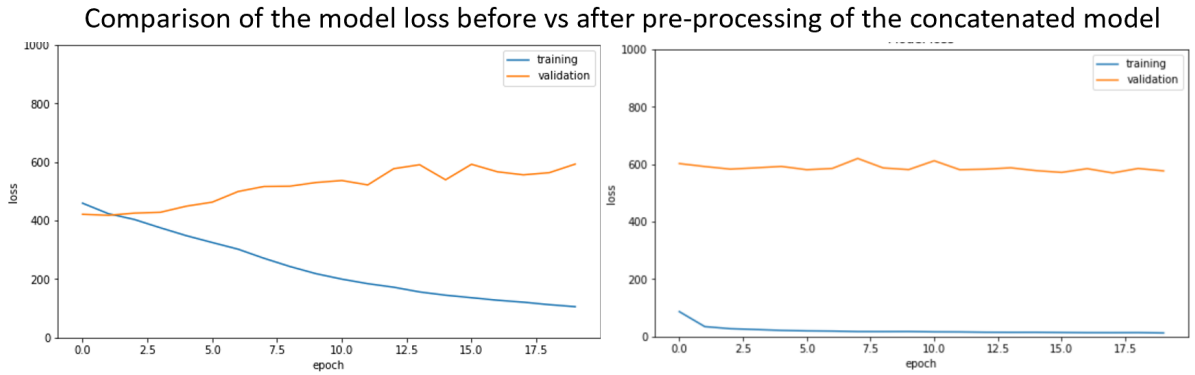


Figure 5: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) data of the convolutional (A) and concatenated model (B) for 20 epochs of training.

The concatenated model seemed to diverge less than the CNN and had a lower validation MSE (figure 5). 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 comparison of the model MSE between preprocessing and no preprocessing is shown in figure 6. The training MSE of the concatenated model after preprocessing was 12.83 after 20 epochs and 569.05 for the validation MSE. Thus, the training MSE after preprocessing decreased a lot whereas the validation MSE increased. From this, we concluded that the preprocessing methods did not solve the problem of overfitting. However, the validation MSE of the model after preprocessing is slightly lower.



A. Before pre-processing B. After pre-processing

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 evaluation, 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 methods in the model. We will try to reduce the overfitting with other means. For instance, by using dropout.

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. The model becomes more generalizable and it is therefore less likely to overfit.

Data analysis and preprocessing

Data analysis and preprocessing methods remained 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 then consisted of two hidden layers, with each 20 hidden nodes and ReLU activation (figure 7). The different probabilities tried and their outcome can be found in table 2. 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 whether dropout worked.

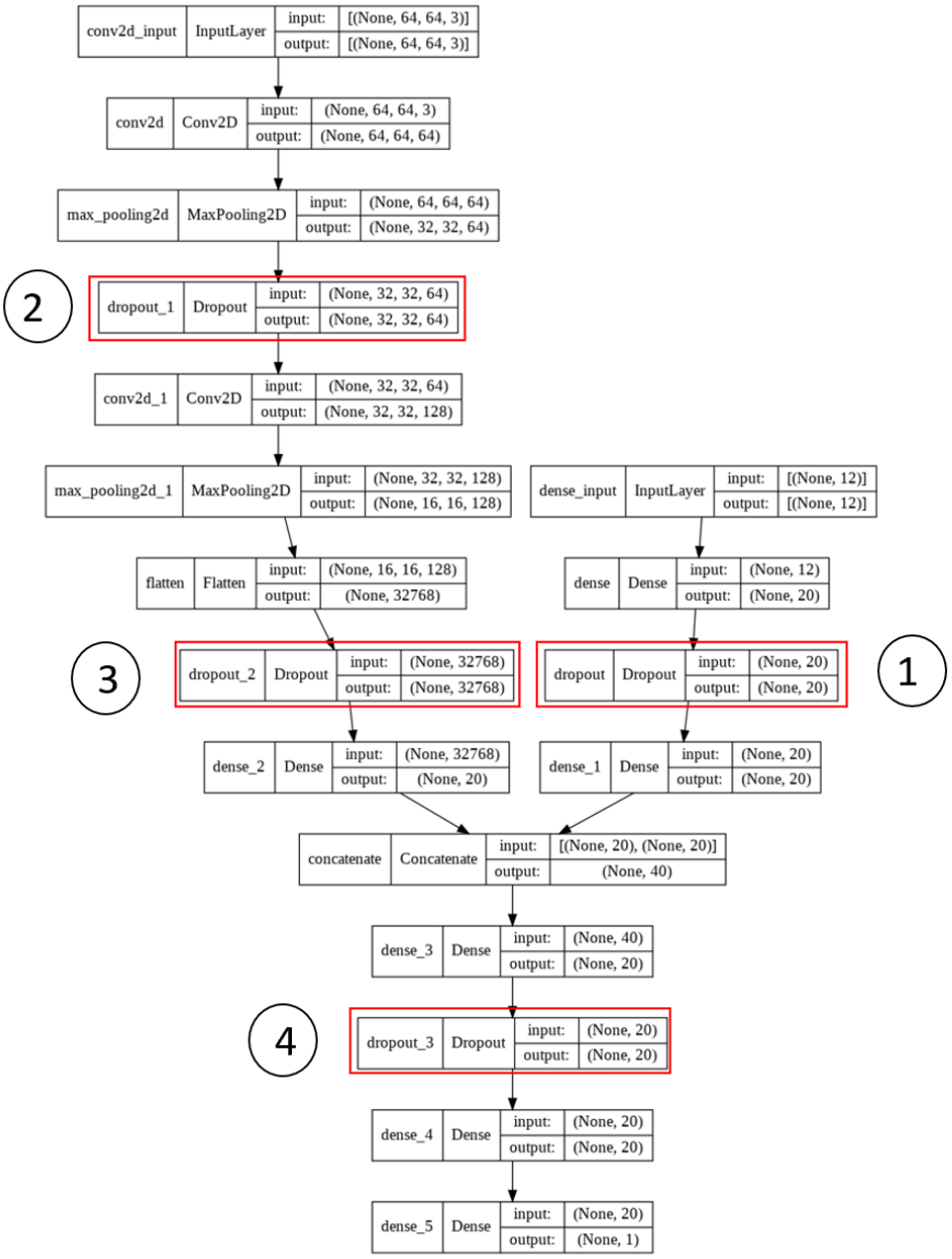


Figure 7: Overview of the layers in the CNN (Convolutional Neural Network) for images (left), neural network for tabular data (right) and concatenated network (below) with dropout layers added, indicated by red boxes. The numbers 1 until 4 will be used in table 2.

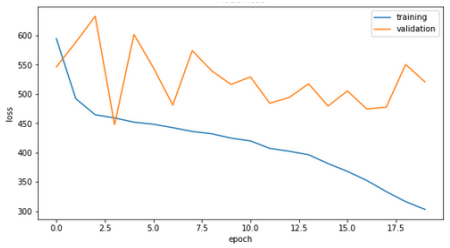
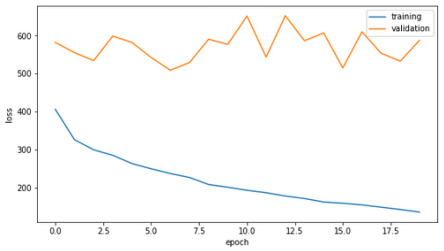
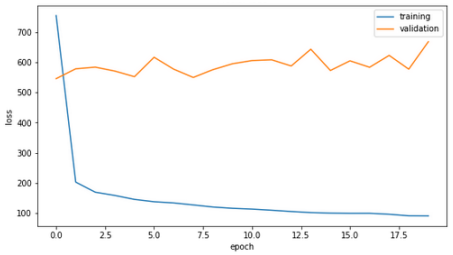
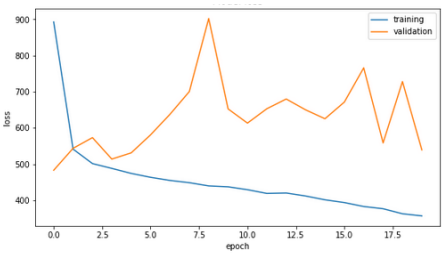
Evaluation and conclusions

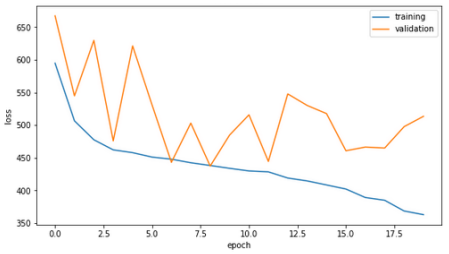
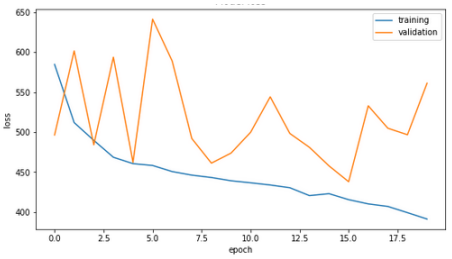
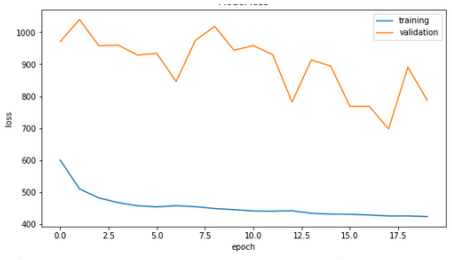
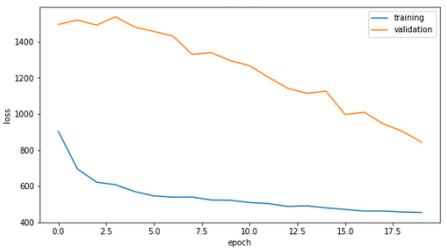
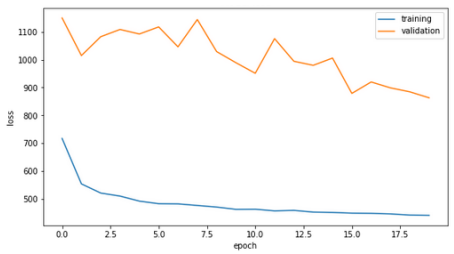
The results of the different dropout probabilities are given in table 2.

Table 2. 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, attention must be paid to the fact that 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.

CNN: Convolutional neural network, MSE: Mean Squared Error.

Dropout Layers	**Model MSE**	**Model MSE Graph**
Tabular Dense Layer: 0.4 CNN 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 CNN Conv2D 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 CNN 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 CNN Conv2D layer: 0.2 Dense layer: 0.2 Concatenated	Training MSE: 356.9272 Validation MSE: 539.2504 Difference: 182.3232	

Hidden layer: 0.4		
Tabular Dense layer: 0.4 CNN 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 CNN 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 CNN 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 CNN 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 CNN 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, CNN, 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 this increase, the final training MSE is 527.97 and the final validation MSE is 530.12. The results are shown in figure 8 below.

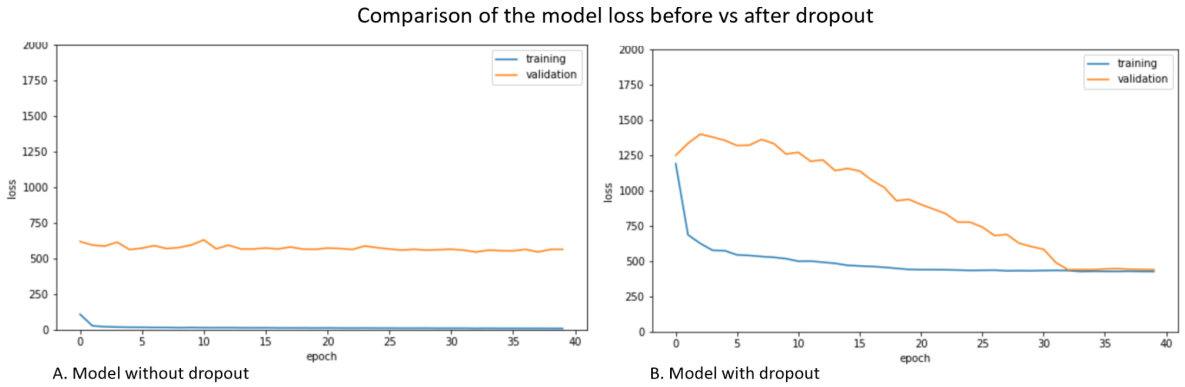


Figure 8: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) of the model before dropout (A) - and model after dropout (B) for 40 epochs of training.

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

The final model of this chapter did not overfit, but had a high MSE, meaning that our model does not learn the data perfectly yet. For our next 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 this 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 quite high. 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 the effect of adding extra layers.

Data analysis and preprocessing

Data analysis and preprocessing methods remained 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 Dense layer in the tabular neural network, with a dropout rate of 0.8 and 20 hidden nodes and ReLU activation. In the CNN we added one more convolutional layer with 256 filters, including an extra dropout and maximum pooling layer. We also added one extra Dense layer in the CNN, 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 with 512 filters and an extra maximum pooling layer.

Evaluation and conclusions

Table 3 shows the different versions of the network, showing the learning curves and the final model MSE (root Mean Squared Error). 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 was lower than the training MSE, meaning that the model did not overfit. The same was true for the model with 2 extra convolutional layers. Since both networks performed equally well, we chose the model with one extra convolutional layer because of its simplicity.

The comparison of the model's learning curves with the previous model is shown in figure 9. 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. Hence, we will continue with this model. An overview of the new model with the extra layers is displayed in figure 10. 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 when using a lower dropout rate.

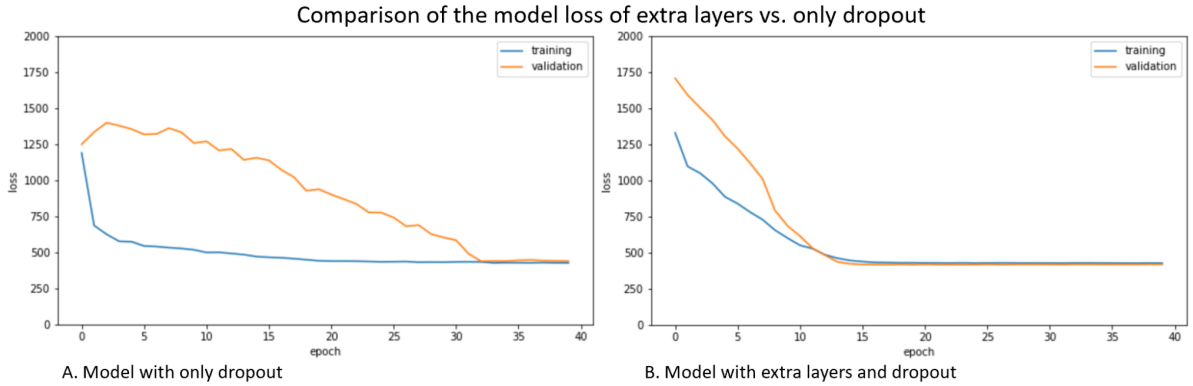
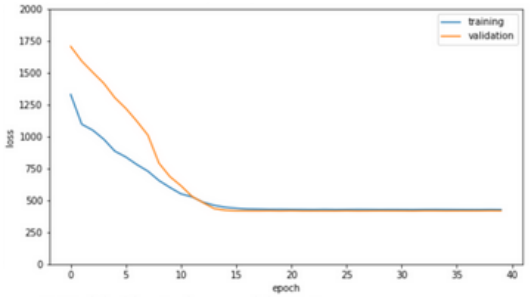
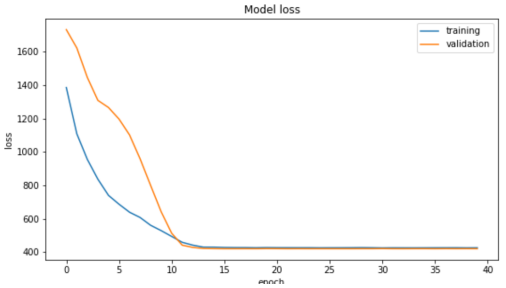


Figure 9: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) of the model with only dropout (A) and model after adding extra layers (B) for 40 epochs of training.

Table 3. 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.*
CNN: Convolutional Neural Network, MSE: Mean Squared Error.

Layers	**Model MSE**	**Model MSE graph**
Tabular 1 extra Dense layer CNN 1 extra Convolutional layer 1 extra Dense layer Concatenated 1 extra Dense layer	Training MSE 525.85 Validation MSE 521.31	
Tabular 1 extra Dense layer CNN 2 extra Convolutional layers 1 extra Dense layer Concatenated 1 extra Dense layer	Training MSE 525.84 Validation MSE 520.21	

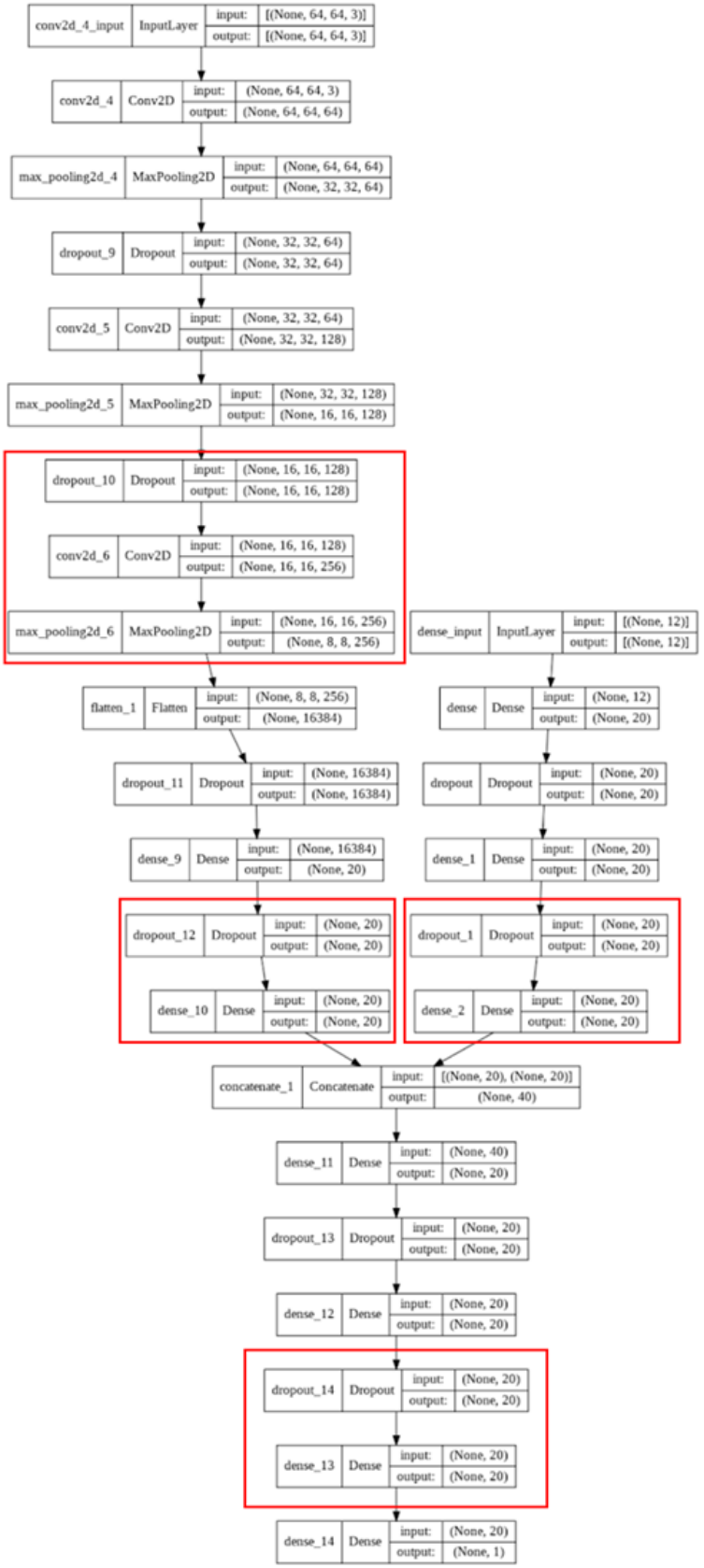


Figure 10: Overview of the layers in the CNN (Convolutional Neural Network) for images (left), neural network for tabular data (right) and concatenated network (below). The new layers are highlighted in red. This is the network we will use in later versions.

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, there is a high probability that a node is silenced and 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 remained the same in this version.

Model pipeline and training

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

Evaluation and conclusions

The table with all the attempts can be found in appendix A.

When using a dropout rate of 0.2 for one or more layers, the model started to overfit again. Looking at the learning curves and the final training and validation MSE, we chose to continue with the model that had each dropout rate set to 0.4, except for the dropout rate in the two Dense layers in the CNN, which were set to 0.3. An overview of the model with dropout is shown in figure 11. We chose this model, because the final validation MSE (467.07) was lower than the previous final validation MSE (525.17) and the learning curves showed that the model was not overfitting. The final training MSE of this model was 453.71, whereas the previous final training MSE was 525.85. The best attempt of the model MSE with decreasing the dropout rate is displayed in figure 12.

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

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

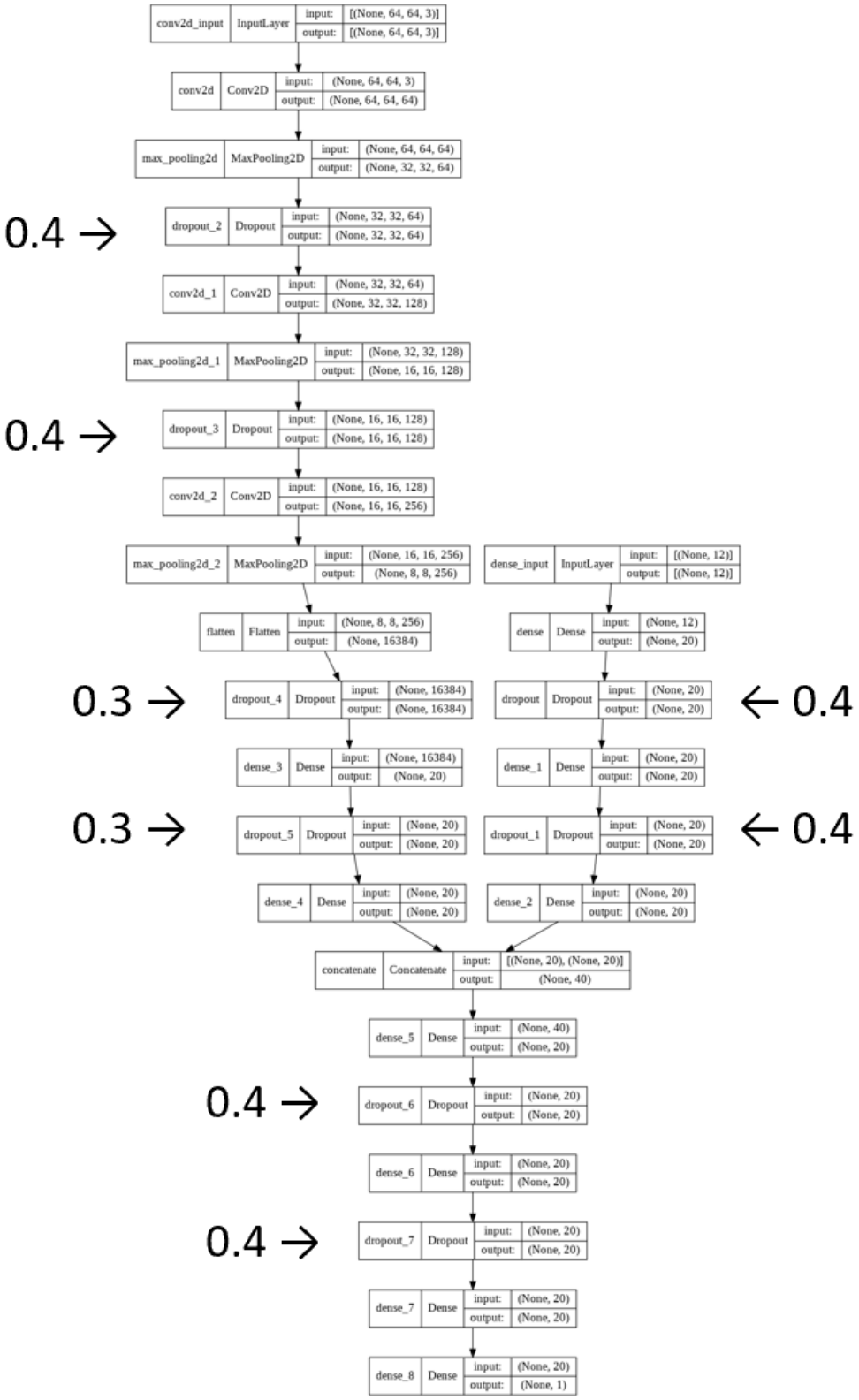
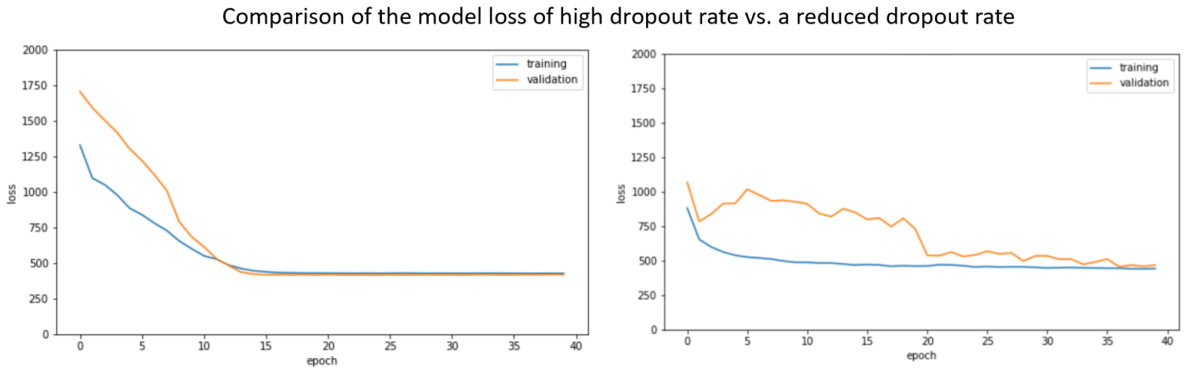


Figure 11: Overview of the layers in the CNN (Convolutional Neural Network) for images (left), neural network for tabular data (right) and concatenated network (below). The updated dropout rates per layer are displayed on the side. This is the network we will use in later versions.



A. Model with high dropout rate

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

We will apply batch normalization to increase the learning speed of the network. 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 remained 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 CNN. 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 was 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 had decreased from 453.71 to 198.43 after 60 epochs. Comparing this result to the result of the model from chapter 5, there was a significant decrease of training MSE 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 did not generalize well enough to new samples and overfitted. In order to solve this problem we will add regularization terms in chapter 8.

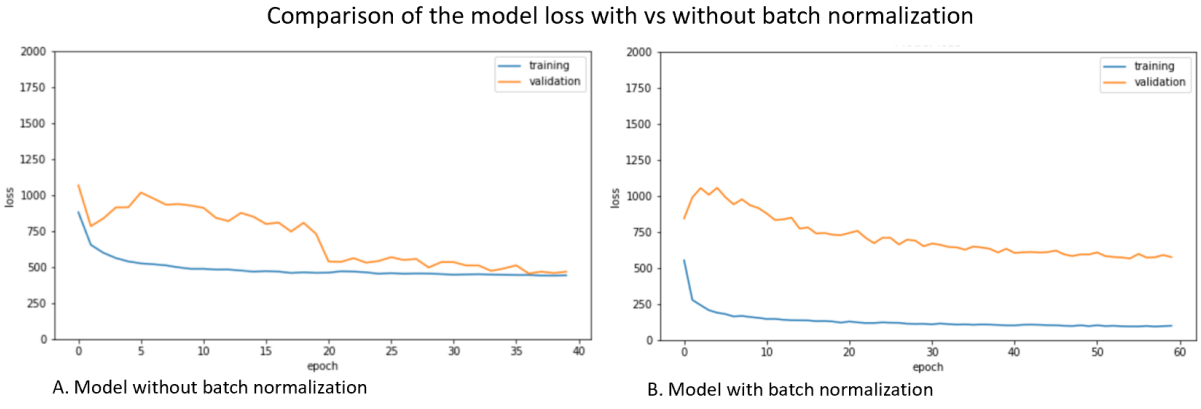


Figure 13: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) of the model before applying batch normalization (A) and model after applying batch normalization (B) for 60 epochs of training.

Chapter 7: Linear output

Introduction

With our model, we want to predict a Pawpularity score between 0 and 100. Therefore, we want to restrict the output of our model between 0 and 100. In our previous model, we used a linear output without any 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 by making it more realistic and thereby lower the validation MSE.

Data analysis and preprocessing

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

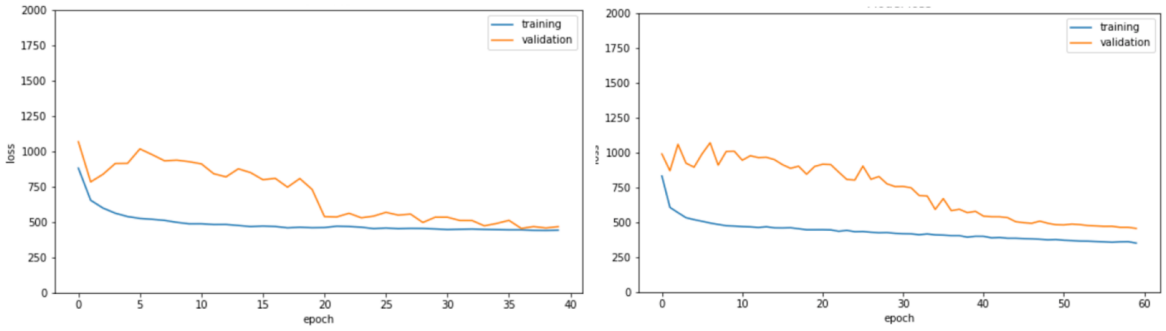
Model pipeline and training

We customized an activation function for the output layer of the network, so it had a linear output between 0 and 100. This way, all the output was 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 gave a better result for the model MSE compared to our previous version. After 60 epochs, the new training MSE was 525.39 compared to 453.71 and the new validation MSE was 609.38 compared to 467.07. The results are displayed in figure 14. Because the model had an increased training cost and increased validation cost, the model overfitted less. Hence, the clipping of the linear output improved the model and should therefore be used in the following models.

Comparison of the Model loss of the previous model vs. the model with clipped linear output



A. Previous model

B. Model with clipped linear output

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 remained the same in this version.

Model pipeline and training

Different types of regularization were tried to the model combining chapter 6 and 7. In each layer, it was possible to add a kernel, bias and activity regularization. The options of regularization were L1, L2 or L1 and L2 combined.

From table 2, it seemed like overfitting arised in the concatenated model, so regularization terms were added in these layers as a starting point. Regularization terms were also applied in the Dense layers of the CNN. Different values for the regularization parameter lambda were used. We used L2 regularization, since this is the most common type.

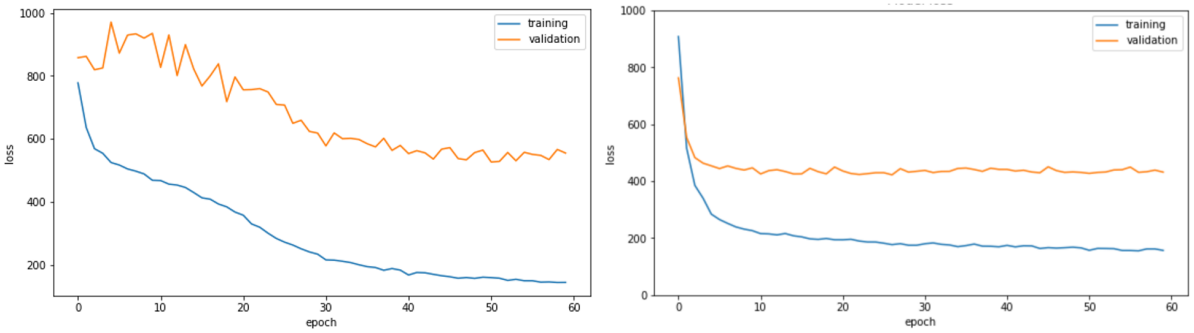
Evaluation and conclusions

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, while still remaining 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 was still overfitting, but less than the other options. The model also seemed to perform well before 30 epochs. This model had an L2 kernel, bias and activity regularizer of 1e-3 on the Dense layers of the CNN and a lambda of 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 model which combined chapter 6 and 7. The previous model had a training MSE of 525.39 and validation MSE of 609.38 after 60 epochs. Because of this decrease in both MSEs, 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 in the training data 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) and after adding a L2 regularization term of 1e-3 on the 2 Dense layers in the CNN (Convolutional Neural network) and a term of 1e-1 on the Dense layers of the concatenated model (B) for 60 epochs of training.

Chapter 9: Removing upper outliers

Introduction

When reviewing the distribution of the Pawpularity score, shown in figure 1 and figure 16A, 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 of the model. 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.

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 of chapter 8 was trained in 60 epochs, the distribution of the predicted values was plotted (Figure 16B). This distribution was a bit left-skewed with a high peak at a score of 60. Since this did not reflect the distribution of the real Pawpularity scores (figure 16A), we removed the samples with a score of 100. We did this, because those samples could be outliers and worsen the performance of the model. Figure 16A shows that Pawpularity scores of 5 and below also had a high occurrence. As the occurrence was much lower than a score of 100 and a score below 5 seemed closer to the median of the distribution, we first looked at the effect of removing only the samples with a value of 100 and will later look at the samples with a value lower than 5.

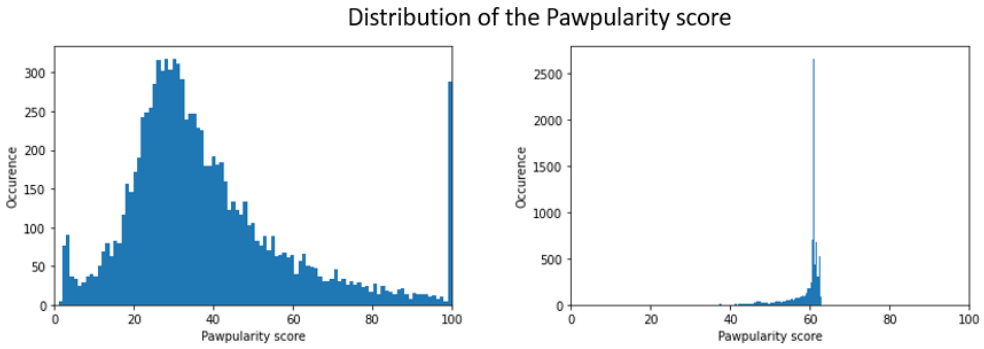


Figure 16: Distribution of the Pawpularity score. A. The distribution of the Pawpularity score from the dataset. B. The distribution of the Pawpularity score predicted by our model.

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 were removed from the data preprocessing. The non-split tabular and image data were 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.

Model pipeline and training

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

Evaluation and conclusions

The training and validation learning curves of the model without outliers and the model with outliers are shown in figure 17. The final training and validation MSE of the model that was trained on the data with the outliers still included using k-fold was 285.30 and 508.39, respectively. When training the model on the data where the outliers were removed, the final training MSE was 209.08 and validation MSE was 382.99. Since both the training and validation MSE decreased, the model performed better when the outliers are not present than when the outliers are in the dataset.

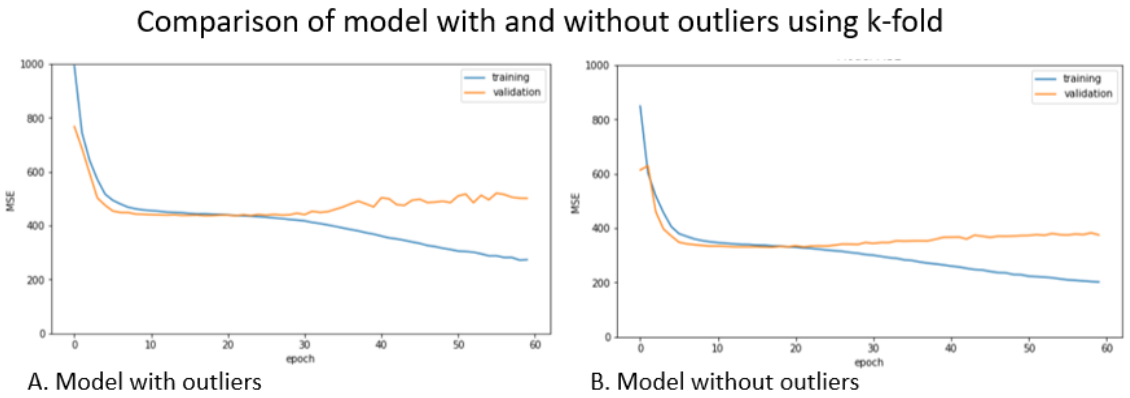


Figure 17: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) of the model before removing the outliers (A) and after removing the outliers (B) for 60 epochs of training.

A distribution of predictions was created after outliers were removed. This is shown in figure 18. The Pawpularity scores are distributed between 25 and 60. It looked a lot more like the distribution of the training scores, as shown in figure 16A. Since the model performed better when outliers are removed and since the distribution looked more like the distribution from the training data, we will exclude outliers from the dataset in further versions of the model.
The model was still overfitting. Therefore, we will add data augmentation in the next chapter.

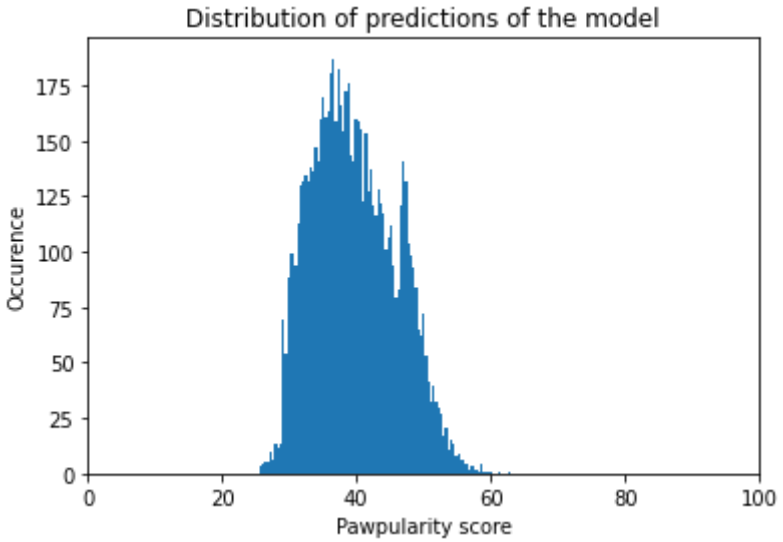


Figure 18: Distribution of predictions of the model after removing outliers with a Pawpularity score of 100.

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, so the amount of training data increases. Moreover, the variation between the samples increases, so a wider range of features is learned. This makes it more likely that the features that are present in the validation set are learned and this prevents overfitting. The goal of data augmentation is to bring the final training and validation MSE 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 (figure 19B). 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. When applying rotation or shear range to an image, this results in empty pixels around the borders of the image, as can be seen in figure 19C and 19D. Since the background of images may be important in predicting the Pawpularity score, empty pixels could lead to incorrect learning of the model. We chose to fill in an empty pixel with the nearest pixel of the image, so the pixels at the border to prevent the model from incorrect learning.

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.

A. B. C. D.

Figure 19: Data augmentation methods that are implemented in the model. A. The original photo. B. Horizontal flip. C. Rotation range (50 degrees). D. Shear range (20 degrees).

Model pipeline and training

The model was trained on the pile of already existing images and the augmented images. The data was validated on the non-augmented images, because we wanted to know whether the model

performance improved with a higher variation in training data. We did not use k-fold as that would have taken up too much time with the different types of data augmentations.

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 goal of data augmentation was 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 chose 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 did not overfit anymore. The training and validation MSE of both models are displayed in figure 20.

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

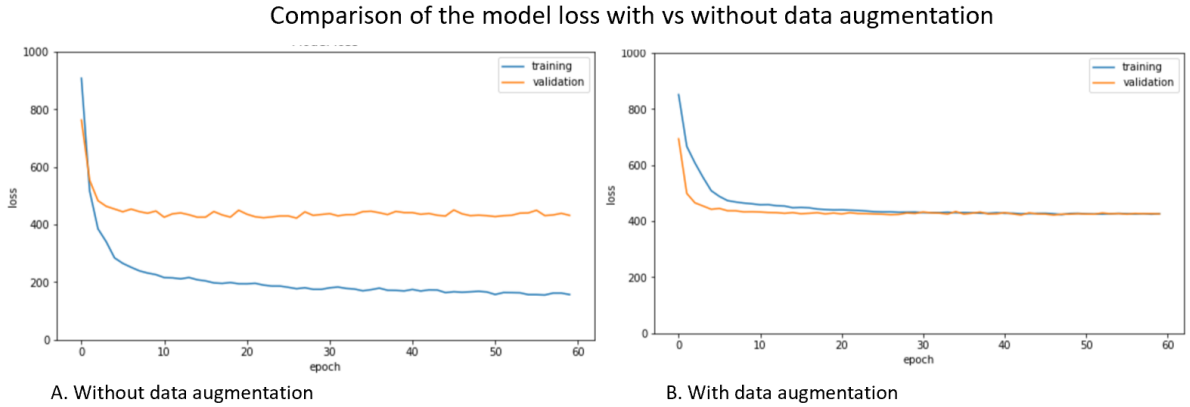


Figure 20: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) data of the model after adding regularization terms, before applying data augmentation (A) and after applying data augmentation (B) for 60 epochs of training.

Chapter 11: Data augmentation combined with removal of outliers

Introduction

In the model from chapter 9, we managed to get a model with a low train and validation MSE by 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

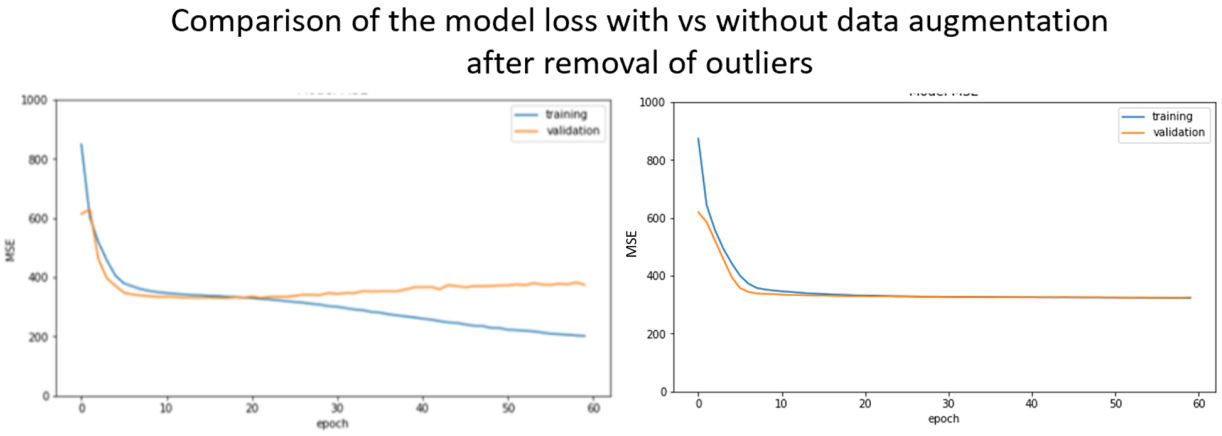
We removed the samples that had a Pawpularity score of 100, as those are the outliers. Moreover, we applied data augmentation to the images. We applied a horizontal flip, a rotation range of 90 degrees and a shear range of 0.2, as this resulted in the least overfitting in the previous chapter.

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. 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 21. In conclusion, the model did not overfit anymore and had a relatively low training and validation MSE. Hence, our goal for this model has been achieved. Although the MSEs of this model were already quite low, we could still try to reduce them even more. We will do this by adding more complexity to the model, so by 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 21: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) data of the model with the removal of the outliers, before applying data augmentation (A) and 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. The two options that we have are making 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 CNN. This tabular neural network is a DNN 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 remained 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 returned to a tabular network with just one Dense layer. The input of the Dense layer was the 12 input features in the tabular data and this was spread out over 12 output nodes with no activation function, which were then the input of the concatenated model. An overview of the model is shown in figure 22.

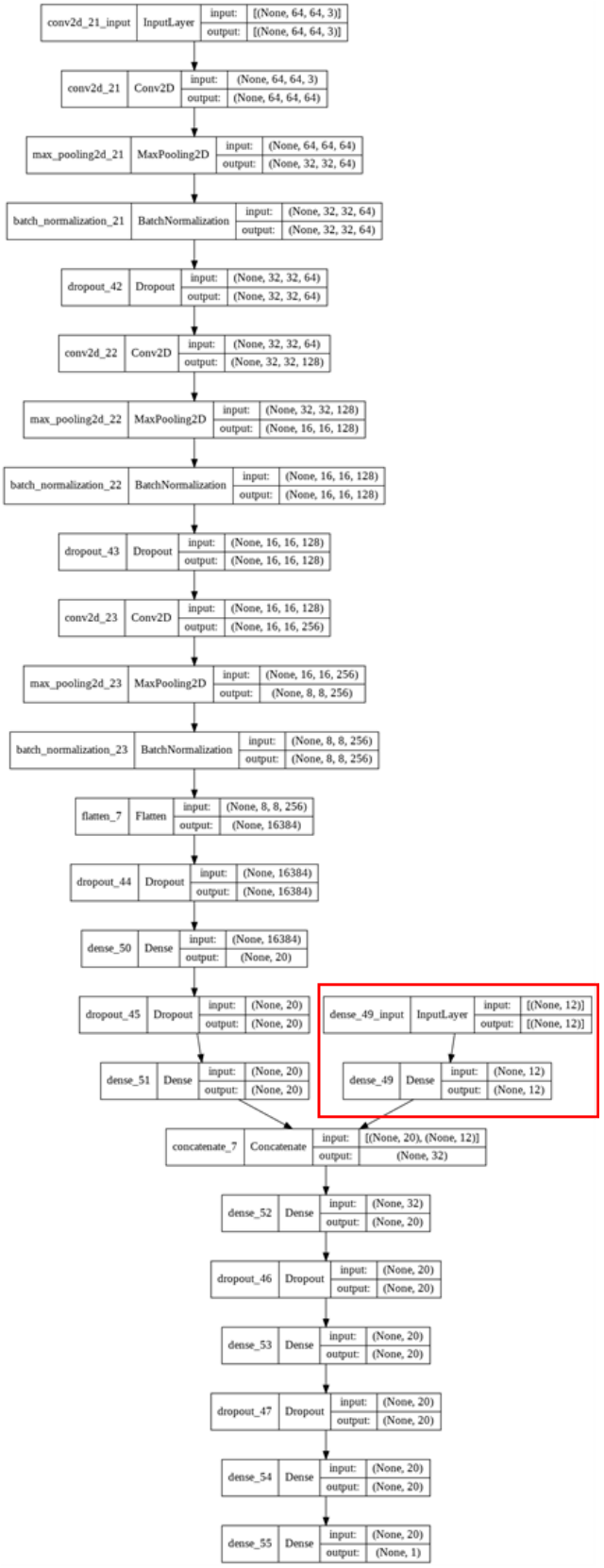


Figure 22: Overview of the layers in the CNN (Convolutional Neural Network) for images (left), neural network for tabular data (right) and concatenated network (below). Layers in the tabular network have been removed, as shown with the red rectangle. We will not continue with this network.

Evaluation and conclusions

The comparison between learning curves of the previous network and this network is shown in figure 23. The final average training MSE was 322.92 and the final average validation MSE was 325.68 after 60 epochs. In the previous model, the train MSE was 322.87 and validation MSE was 325.14. 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.

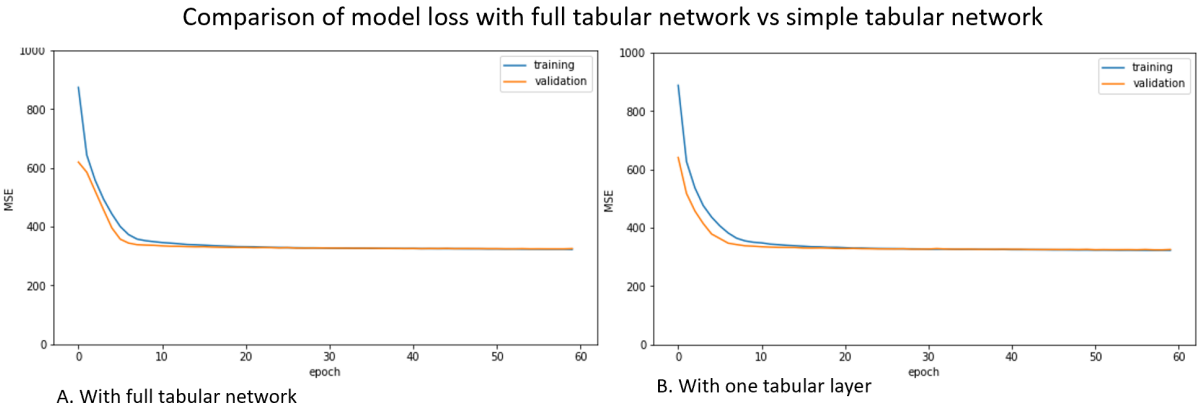


Figure 23: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) of the model with all the layers of the tabular network (A) and with one tabular layer (B) for 60 epochs of training.

Chapter 13: Extra convolutional layers

Introduction

Since our model does not overfit, we could try to make our model more complex. 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. Then, 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 remained the same in this version.

Model pipeline and training

After each convolutional layer, we now added one extra convolutional layer. This layer did not have maximum pooling or dropout, and contained 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 has two extra layers instead of one.

Evaluation and conclusions

For the model where each convolutional layer had one extra layer, the final training MSE was 322.77 and the final validation MSE was 324.54, after 60 epochs. This is slightly better than the previous model, which had a train MSE of 322.87 and validation MSE of 325.14.

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 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. The comparison of these two models is shown in figure 24 and an overview of this model is depicted in figure 25.

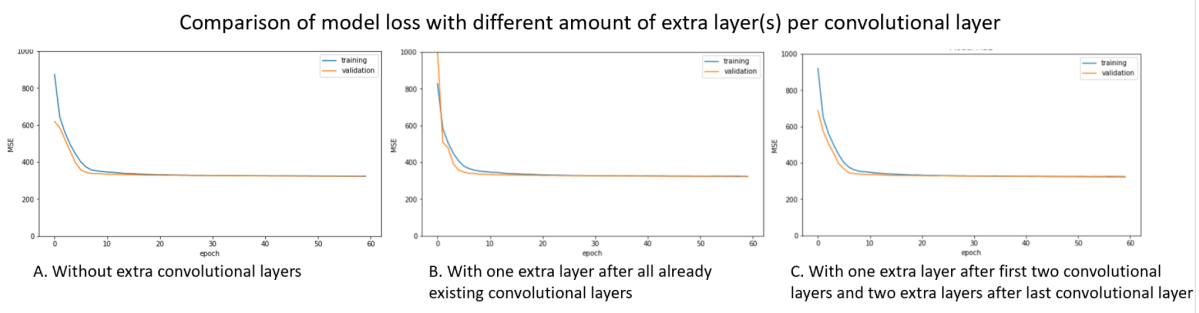


Figure 24: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) data of the model without extra convolutional layers after the already existing convolutional layers (A), with one extra convolutional layer after all already existing convolutional layers (B) and with one extra convolutional layer after the first and second convolutional layer and two extra convolutional layers after the last convolutional layer (C) for 60 epochs of training.

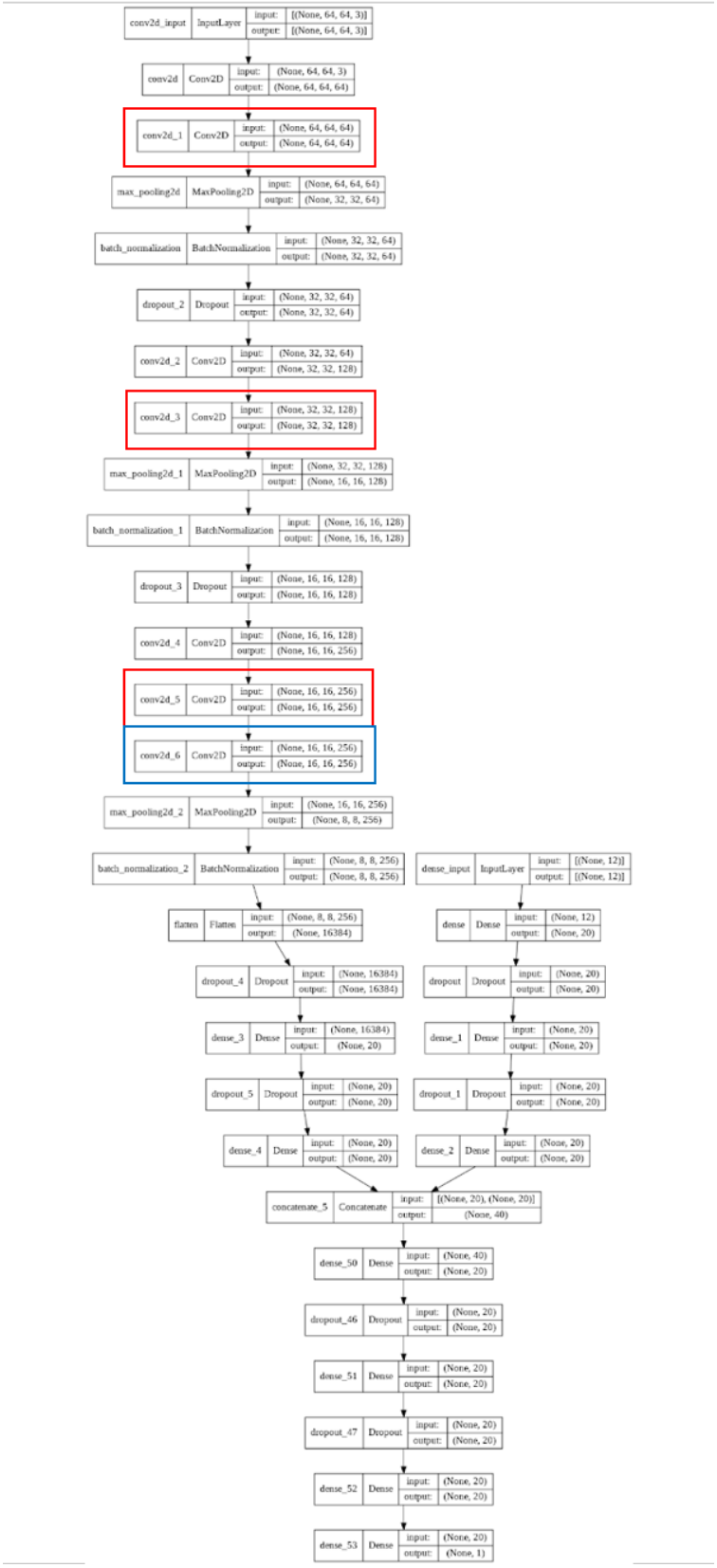


Figure 25: Overview of the layers in the CNN (Convolutional Neural Network) for images (left), neural network for tabular data (right) and concatenated network (below). The new layers are highlighted in red. The layer in blue was tested, but we will not use this layer in later versions. This is the network (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. 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 a decrease in both MSEs. We will add more hidden nodes in the Dense layers of the CNN and in the Dense layers of the concatenated network. By doing this, more parameters 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 remained 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. Therefore, there are no extra convolutional layers in this model. An overview of the adjustments are presented in figure 27. We did not increase the number of 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 CNN from 20 to 100, as these Dense layers have a lot of input nodes from the convolutional part of the network. Moreover, we increased the number of hidden nodes from the Dense layers in the 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 the final validation MSE was 324.19, after 60 epochs. We compared this result to the model with no increase in hidden nodes (from chapter 11) with a validation MSE of 325.14 in figure 26. It shows that our current model performed slightly better. In the next chapter we will check whether a model with both extra layers and extra hidden nodes performs even better, since both this model and the model with the extra convolutional layers performed better than the model from chapter 11,.

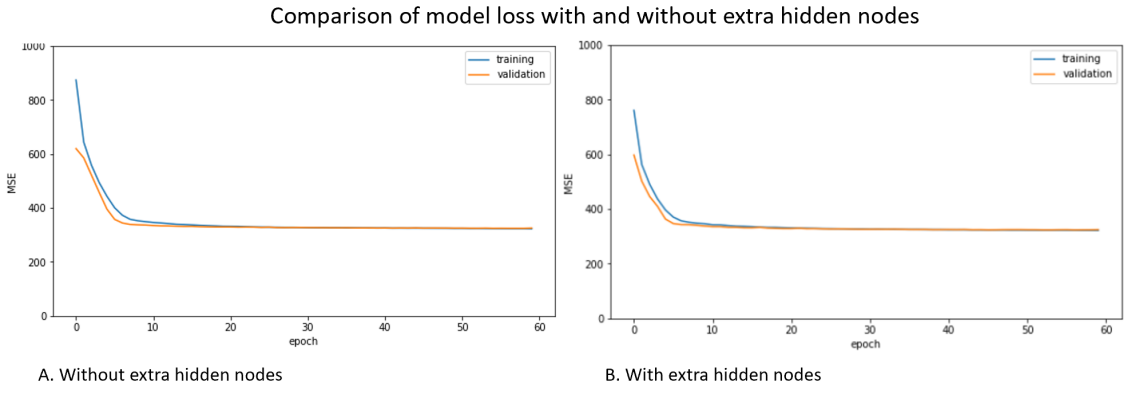


Figure 26: Graph showing the comparison of the MSE (Mean Squared Error) of the training (blue) and validation (orange) data of the model without extra hidden nodes (A) and with extra hidden nodes (B) for 60 epochs of training.

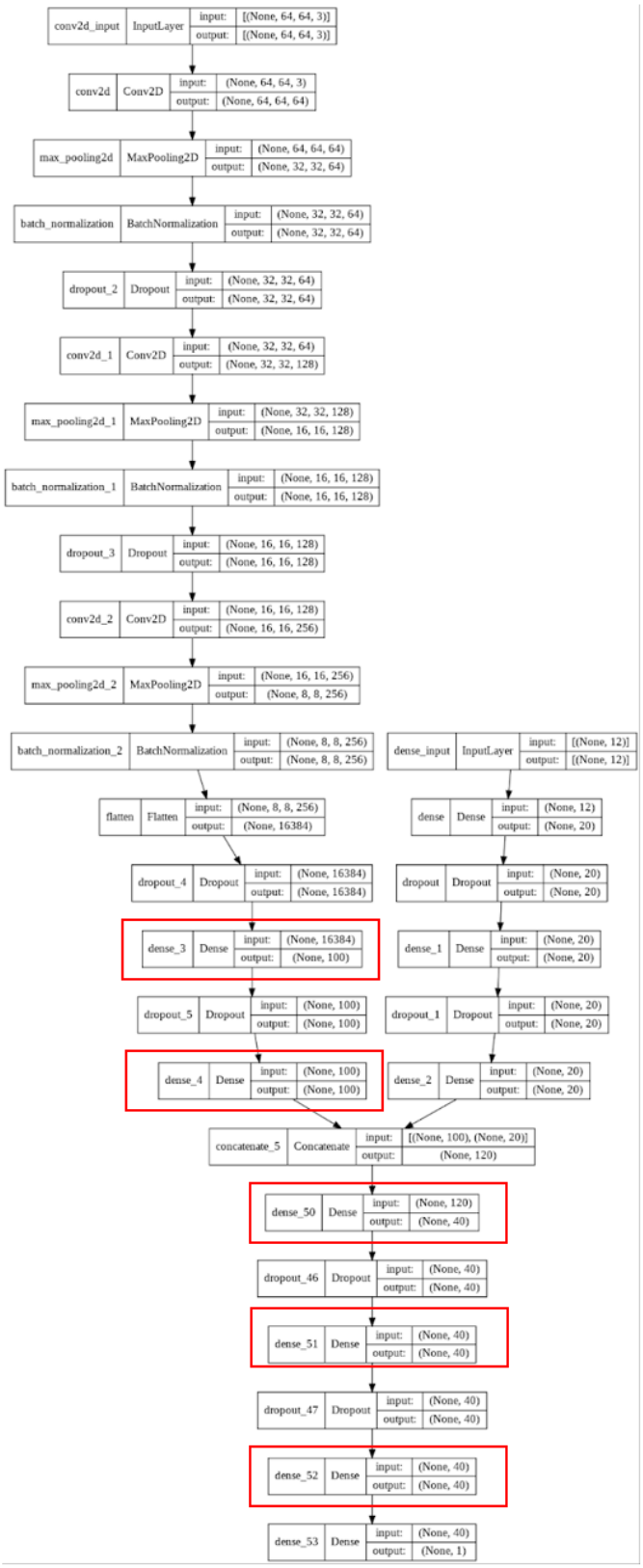


Figure 27: Overview of the model with the extra hidden nodes in the CNN (Convolutional Neural Network) for images (left), neural network for tabular data (right) and concatenated network (below). The layers with the adjusted hidden nodes are highlighted in red.

Chapter 15: Extra convolutional layers and hidden nodes

Introduction

In the previous two chapters we increased the complexity of the model by adding more convolutional layers or more hidden nodes. This resulted in a small decrease in the MSE, while the model was not overfitting. As our goal is to minimize the cost, we will combine the previous two chapters in this chapter. When adding more convolutional layers and increasing the number of hidden nodes, the model becomes more complex. Consequently, the model might be able to better learn the data and thus perform better. However, as the complexity increases, it might be that the model starts to overfit. If that is the case, a less complex model should be used.

Data analysis and preprocessing

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

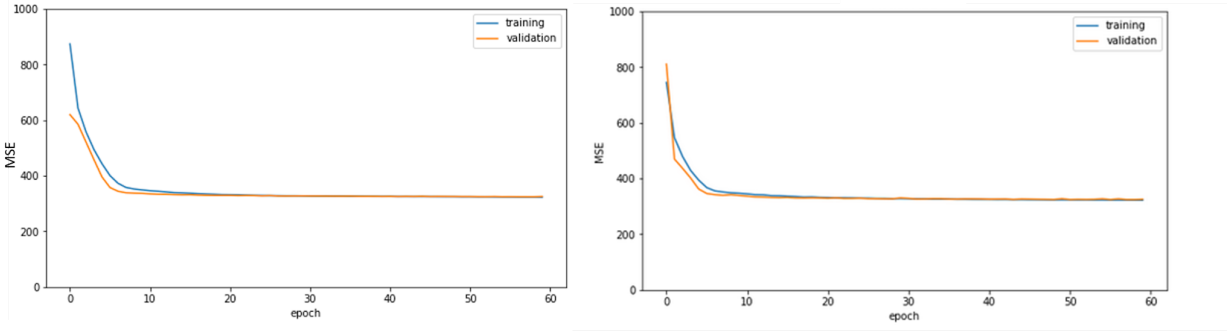
Model pipeline and training

For this version we combined the models of chapter 13 and 14, resulting in a model that contained two convolutional layers before each max pooling layer. The Dense layers of the tabular network had 20 hidden nodes, the Dense layers in the CNN had 100 hidden nodes and the Dense layers of the concatenated network had 40 hidden nodes.

Evaluation and conclusions

Figure 28 shows the model MSE with and without the extra convolutional layers and hidden nodes. When combining the extra layers and more hidden nodes, the train MSE was 322.21 and the validation MSE was 325.24, after 60 epochs. So, the model did not overfit. The model with only extra convolutional layers had a validation MSE of 324.54. Although the current model does not overfit, it does not perform better than the model with only more convolutional layers. Hence, the model is unnecessarily complex and should not be used in further versions.

Comparison of the model MSE with and without extra convolutional layers and hidden nodes



A. Model without extra layers and nodes

B. Model with extra layers and nodes

Figure 28: Graphs showing the training and validation MSE (Mean Squared Error) of the model with and without extra convolutional layers and hidden nodes. A. Model MSE of chapter 11. B. Model MSE, combining chapter 13 and 14, with two convolutional layers before each pooling layer, 100 hidden nodes in the Dense layers in the convolutional network and 40 hidden nodes in the concatenated network.

Chapter 16: Different activation functions

Introduction

In the previous model, we used the ReLU activation function for all the layers except for the output layer. One disadvantage of the ReLU activation function is that it can result in 'dead' nodes. When the activation of a node is lower than 0, the activation is set to 0 with a gradient of 0. If this happens for all the samples, this node becomes a dead node and no longer contributes to the model. To prevent this from happening, we will use the Leaky ReLU activation function instead of ReLU, where the activation is set to a small, negative number instead of 0. Moreover, the gradient is now a small number instead of 0. Therefore, the model is less likely to get dead nodes. For the tabular network, we will use a Sigmoid activation function. Since the tabular data is binary, a Sigmoid activation function will suit this data as this function results in an output between 0 and 1. With these different activation functions we hope to improve the performance of our model.

Data analysis and preprocessing

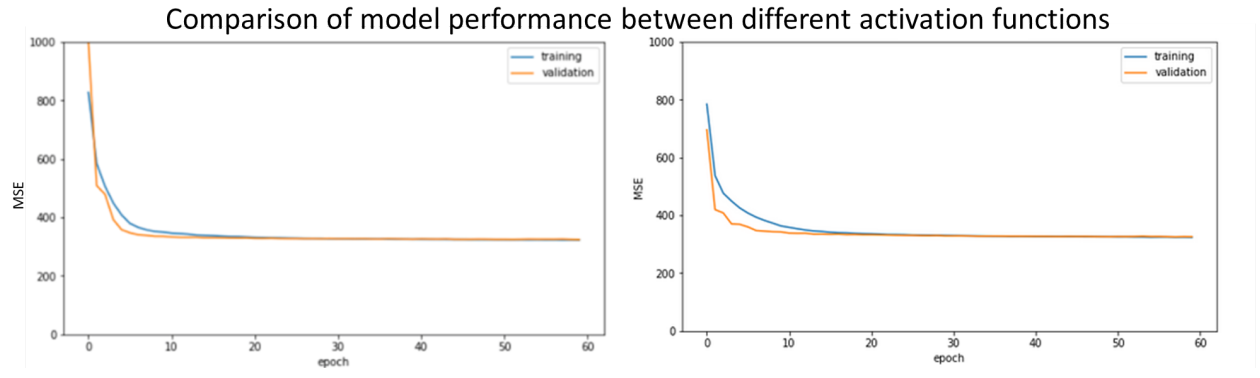
Data analysis and preprocessing methods remained the same in this version.

Model pipeline and training

For this chapter we will use the network from chapter 13, so with one extra convolutional layer. For the tabular neural network we will use a Sigmoid activation function and for the other layers Leaky ReLU. The threshold for this Leaky ReLU is the default value 0.3. The last layer of the concatenated network has the custom linear activation function, which clips the output between 0 and 100 (see chapter 7).

Evaluation and conclusions

Figure 29 shows the comparison of the model's learning curve with the ReLU activation function and the model with Sigmoid and Leaky ReLU activation functions. With the different activation functions, after 60 epochs the final training MSE was 323.76 and a validation MSE was 325.77. Compared to the model of chapter 13, this model did not perform better. Hence, the model of chapter 13 will be used further.



A. Model with ReLU activation function

B. Model with Sigmoid and Leaky ReLU activation functions

Figure 29: Comparing the training and validation model MSE (Mean Squared Error) between models with different activation functions. A. Model with ReLU activation function in all layers. B. Model with Sigmoid activation function in tabular neural network and Leaky ReLU activation function in the convolutional and concatenated neural networks.

Chapter 17: Removing lower outliers

Introduction

In Chapter 9, we saw that removing the outliers with a Pawpularity score of 100 resulted in a decrease of the MSE. Moreover, we saw that there was a high occurrence of samples with a score lower than 5. Since removing the outliers at a score of 100 considerably improved the model, we will look at the effect of removing the lower outliers in this chapter.

Data analysis and preprocessing

We removed the samples that had a Pawpularity score below 5. The remaining analysis was the same as the model of chapter 13. Moreover, we plotted the distribution of the predictions of this model after it was trained, to compare it to the actual Pawpularity scores in the data.

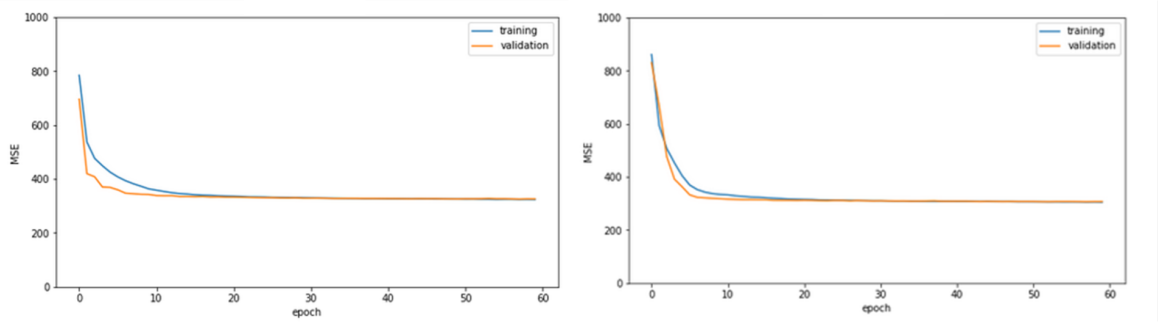
Model pipeline and training

Model pipeline and training remained the same in this version.

Evaluation and conclusions

Figure 30 shows the learning curves of the model with and without removing the lower outliers. The training MSE after removing the lower outliers was 305.08 and the validation MSE was 307.09 after 60 epochs, respectively. The validation MSE of the previous model was 324.54, which is higher than the validation MSE of this model. In addition, with the current model the distribution of the predictions were more comparable to the distribution of the data (figure 1), since there is no peak of predictions around 50. The comparison of the distribution of the predictions of this model and the model of chapter 9 is shown in figure 31. These results suggest that this new model performed far better than the previous ones. Therefore, this model will be our final model.

Comparison of the model performance with and without lower outliers removed

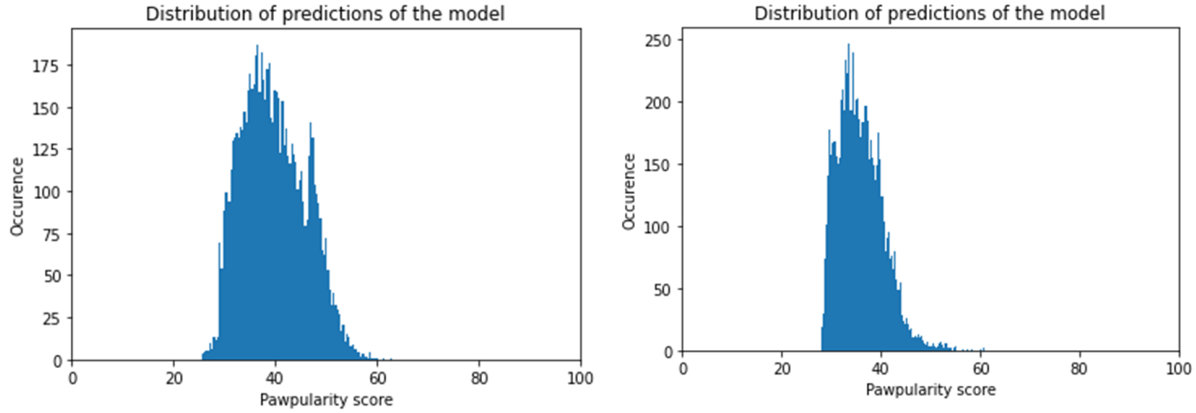


A. Model without lower outliers removed

B. Model with lower outliers removed

Figure 30: Comparing the training and validation model MSE (Mean Squared Error) of the model with and without lower outliers removed. A. Model MSE without lower outliers removed. B. Model MSE with lower outliers (Pawpularity below 5) removed.

Comparing the model predictions with and without lower outliers removed



A. Lower outliers not removed

B. Lower outliers removed

Figure 31: Comparing the distributions of predictions of the model where only the upper outliers (Pawpularity score of 100) are removed (A) and where also the lower outliers (Pawpularity scores below 5) are removed (B).

Chapter 18: The final model

Introduction

During the project, we have tried various versions of our model. To summarize, we used different preprocessing methods, which were input normalization, data augmentation and outlier removal. We made a model that combined a neural network with tabular data as input with a CNN with image data as input into a concatenated model. In this model, we used dropout, batch normalization, regularization terms and a clipped linear output. When we applied these methods to our model, we experimented with the number of layers in each network, the number of hidden nodes in each network, the value of the dropout rate for various layers and different data augmentations. After all our experiments, we concluded that our best model was the model from chapter 17. Hence, we will execute our final evaluation with this model. We will evaluate our model with the provided test set by looking at the MSE. We will also use the RMSE, since this will make it easier to interpret how well our model is performing.

Data analysis and preprocessing

We first removed the samples that had a Pawpularity score of 100 or a score below 5. Then, we reshaped the images to 64x64x3 pixels. We preprocessed the images further by mean and standard deviation normalization. Moreover, we augmented the images with a horizontal flip, a rotation range of 90 degrees and a shear range of 0.2.

Model pipeline and training

Our model had a DNN for the tabular data which had 3 hidden layers, each with 20 hidden nodes and dropout rate of 0.4. The model also had a CNN for the image data. This model had two convolutional layers with 64 filters and kernel size 3x3, followed by a maximum pooling layer with a pool size of 2x2 and stride 2. Batch normalization and a dropout of 0.4 were applied before the next convolutional layer. Next, there were two convolutional layers with 128 filters and kernel size 3x3. After a maximum pooling layer with pool size 2x2 and stride 2, batch normalization and dropout of 0.4, there were two convolutional layers with 256 filters. The last layer was flattened and followed by two Dense layers with 20 nodes. These Dense layers had a dropout of 0.3 and regularization terms (kernel, bias and activity) of 1e-3. The DNN and CNN were combined into the concatenated network with 3 hidden layers with 20 nodes. These Dense layers had a dropout of 0.3 and regularization terms (kernel, bias and activity) of 1e-1. All layers had a ReLU activation function. The output layer contained one node with the clipped linear activation function.

We first started with training the model from chapter 17, while using 10-fold cross-validation, which gives a validation split of 0.1. We chose to use a validation set, because the provided test set is a 'fake' test set of 10 samples where the images and binary features are not meaningful. This is because the real test set can only be obtained if you enter the Pawpularity contest. We used 10-fold because the validation split is then lower, leaving more data to train the model. We trained the model for 60 epochs in each fold. The results from this version will be used to conclude how well our final model performs. We also compared this to other outcomes of the Pawpularity contest.

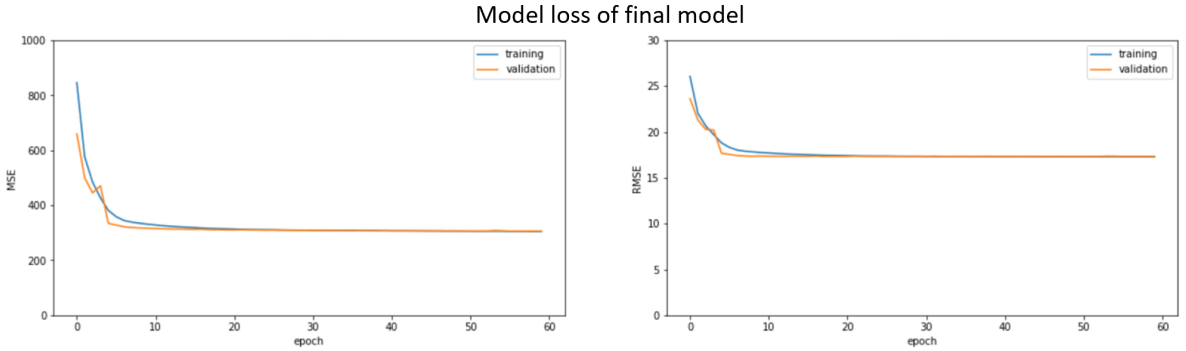
Then, we used the whole training set to train the model and tested the model with the testing images. We did not expect a low MSE and RMSE for the test set, since the images are merely random

pixels. We did make this model so we could hypothetically evaluate the performance of this model, if we had the right test data. Furthermore, we would use this model if we decided to compete in the Pawpularity contest.

Evaluation and conclusion

After letting our final model run with 60 epochs and a 10-fold cross validation, we reached a final training MSE of 305.19 and a final validation MSE of 302.78. In comparison to the previous model, where the training MSE was 305.08 and the validation MSE was 307.09, this model performed better on the validation data. The results are shown in figure 32.

With a final validation root mean squared error of 17.24 we would have been placed first in the public kaggle competition.



A. MSE B. RMSE

Figure 32: A. *Graph showing the training (blue) and validation (orange) MSE of the final model for 60 epochs. B. Graph showing the training (blue) and validation (orange) RMSE of the final model for 60 epochs.*

When we used the whole training set to train the model, the training MSE was 220.61 and the training RMSE was 14.51, after 60 epochs. The results of the test set were a test MSE of 1215.87 and a test RMSE of 34.74. As expected, these values are extremely high because we used a 'fake' test set. Therefore, we are curious what the results would be if we had the right test data.

In the end, we are content with our end results. However, since we had a time limit for this project, we could not test every possible improvement of our model. The experiments that potentially could improve our model are using a different optimizer than the gradient descent, ensemble methods, transfer learning, hyperparameter optimization, and a log transformation on the data.

An optimizer is a method used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses. For our current network we used plain momentum, which reduces the oscillations and high variance of the parameters. However, the disadvantage of gradient descent is that it may trap at local minima and it can take a long time to converge to the minima, since the weights are changed after calculating gradient on the whole dataset. Furthermore, it requires large memory to calculate the gradient on the whole dataset. The optimizer we could try is mini batch gradient descent. The advantage of mini batch gradient descent over gradient descent is that mini batch gradient descent updates the model parameters after every batch, has less variance

and requires less memory. However, with this optimizer you would still have the problem that you may get into local minima.

Another improvement to our model could be to make use of an ensemble of models. Neural networks are influenced majorly by the initialisation of certain parameters, such as the weights. Because this initialization happens randomly, there can be a lot of variance in training the same model. During this project we used k-fold to reduce this variance in results. However, with k-fold you cannot make specific predictions, as you take the average of different models for the loss, but cannot take the average of the weights to make predictions. Another way to reduce this variance is using an ensemble of models. Here, you train different models on a subset of the data. In this way, you have multiple imperfect models. However, combining these models may give a more reliable result. So, in this case you would let each model predict the Pawpularity and the final prediction would be the average of the different predictions.

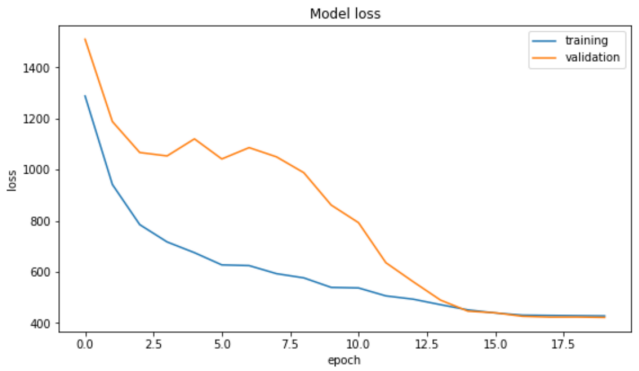
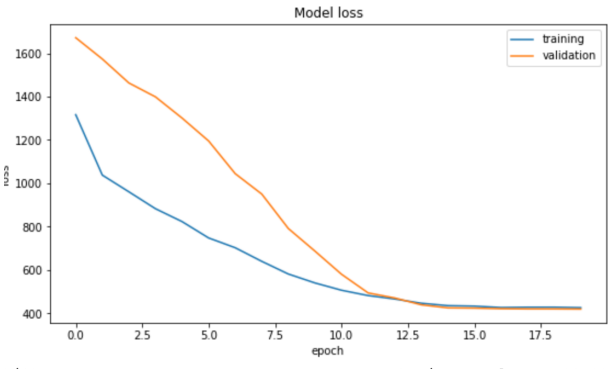
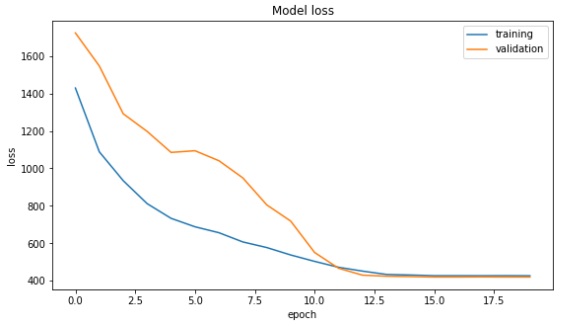
Furthermore, one could make use of transfer learning. Hereby, you use a model that has been trained on a similar problem. You could fix the weights of this pre-trained model or slightly adjust this. In any case, using a pre-trained model will speed up training your own model as the model has less parameters to learn. This could make your model more efficient and thereby better. Another method to improve the efficiency of training the model could be to make use of a different type of hyperparameter optimization. Now, we did a manual optimization. However, this takes up a lot of time and there is only a limited amount of parameter values that can be tried. Using, for instance, random search could speed things up as more (combinations of) random values can be tried and this could eventually lead to a better model.

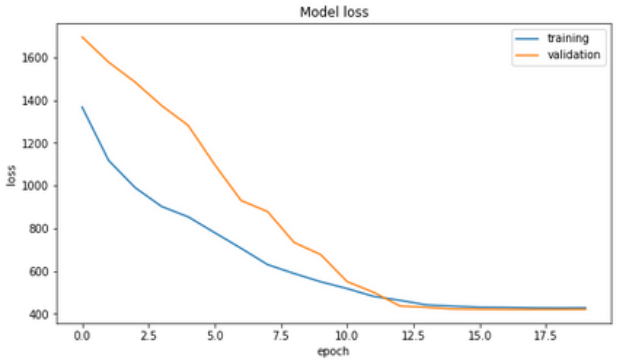
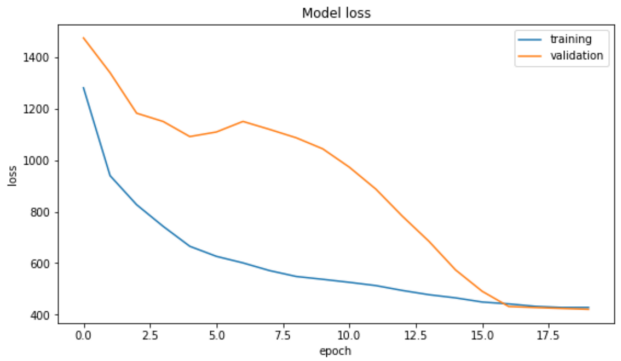
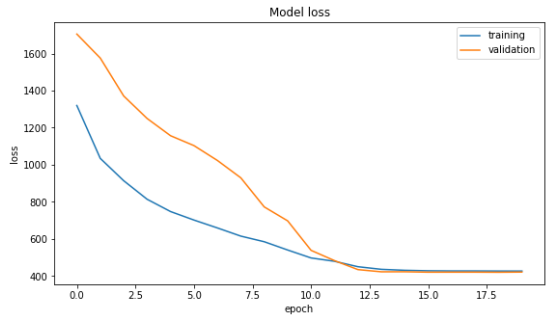
Finally, since the predictions of the training are not normally distributed and right-skewed, we could use a log transformation on our training data to make the distribution less skewed. It is easier to learn patterns from normally distributed data, so the model performance may improve. The reason why we have not tried this yet, is because we would also have to rescale the predictions of the model back by taking the inverse transformation to get the actual predictions of the model. These are difficult steps and were not our priority in creating the model, but it is worth looking into in further Pawpularity models.

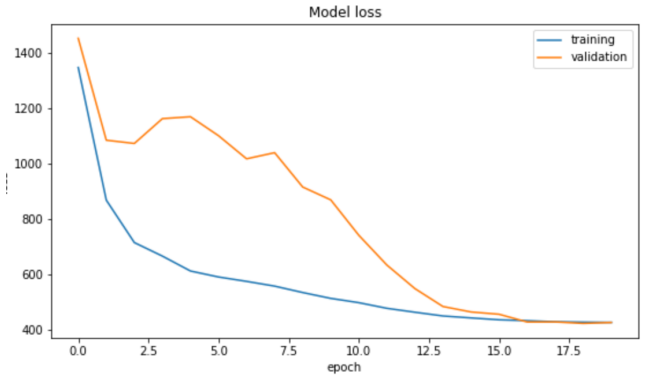
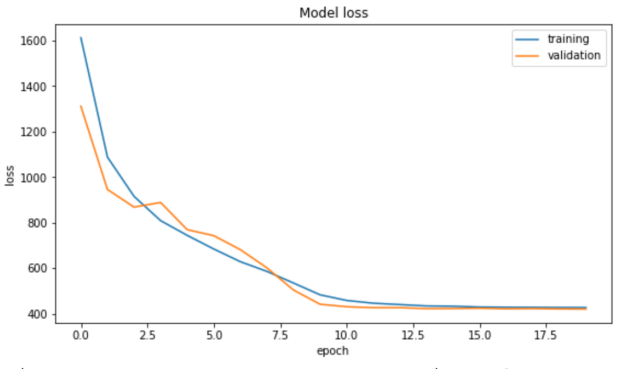
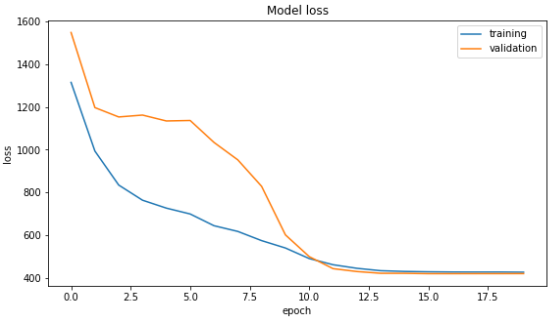
To conclude, we created a model that can predict the Pawpularity score of stray animals which can be helpful in creating better pet profiles so more stray animals can be rescued and adopted.

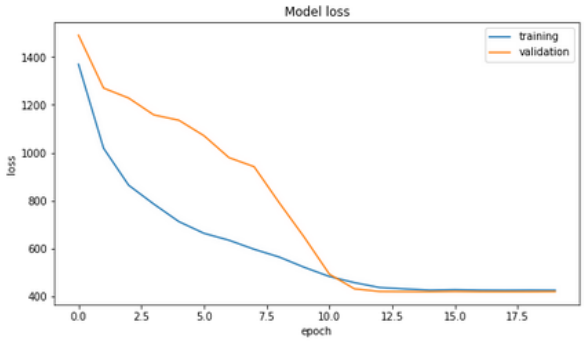
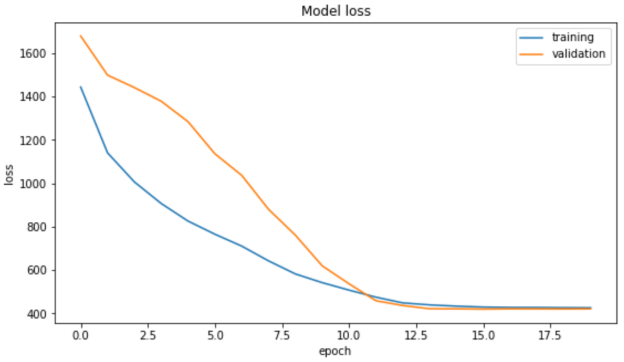
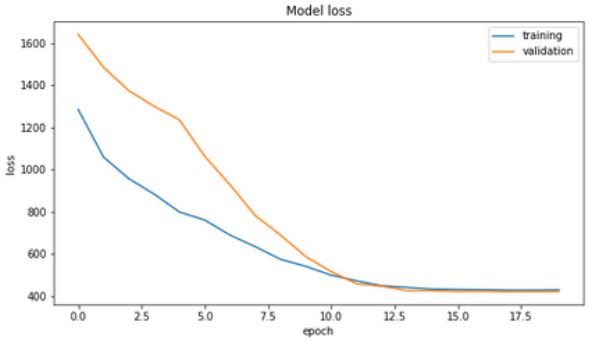
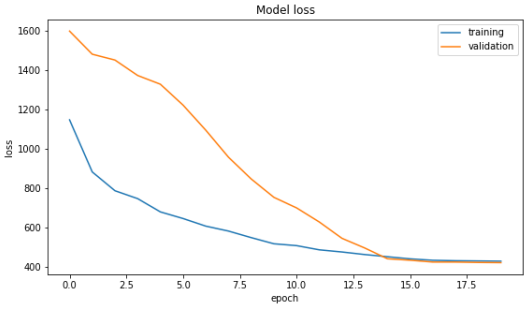
Appendices

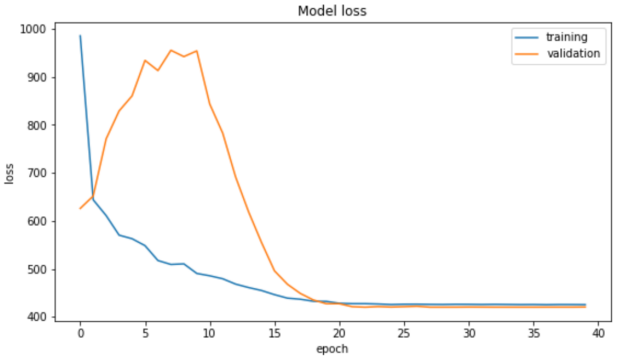
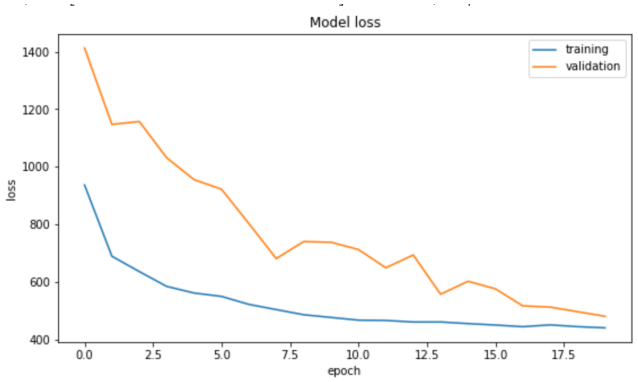
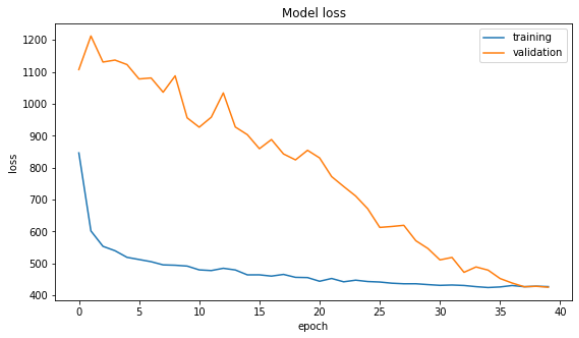
Appendix A: Model training and validation MSE for the combinations tried to decrease the dropout. *The left column shows to which layers which dropout probability is added, the second column the model MSE (MSE) of training and validation data and the third column the learning curve of the model. The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

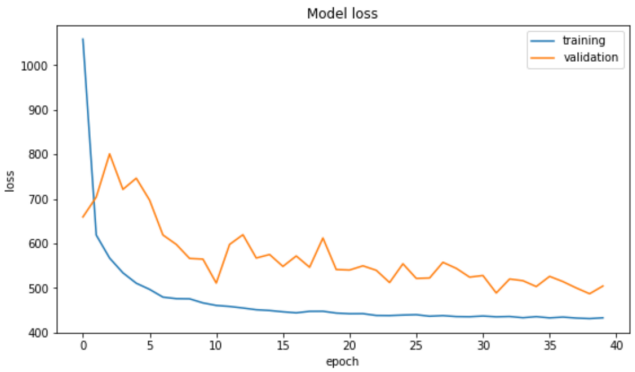
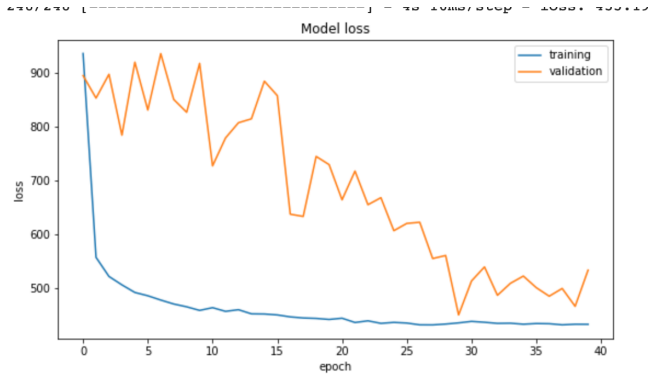
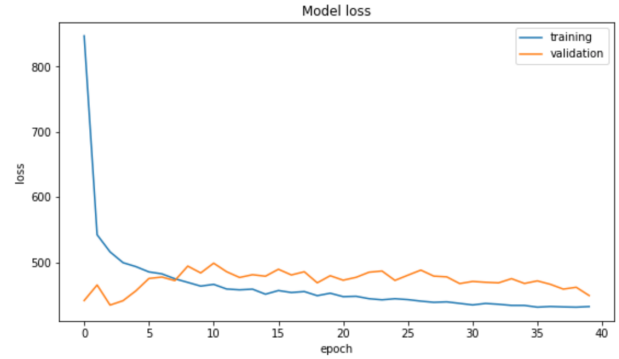
Layers	Model MSE (MSE)	Model MSE graph
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.7	Training MSE 427.39 Validation MSE 421.44	
Tabular Dense layer 1: 0.8 Dense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

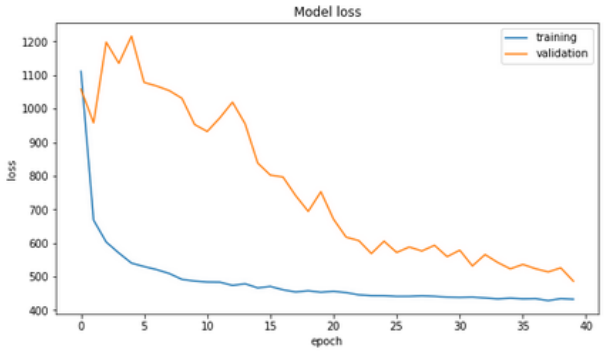
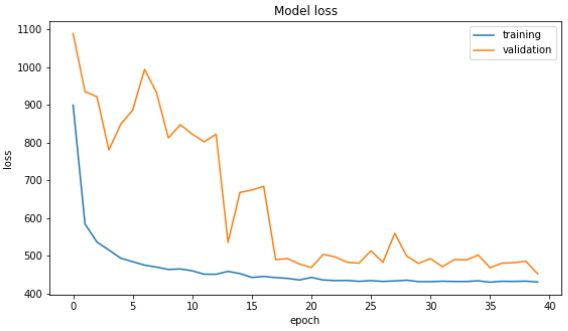
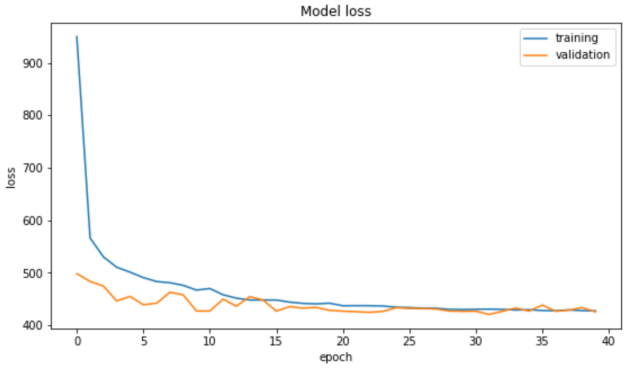
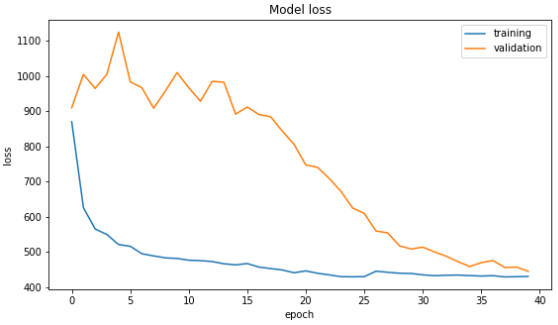
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.7 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Train MSE 427.81 Validation MSE 420.66	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 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	

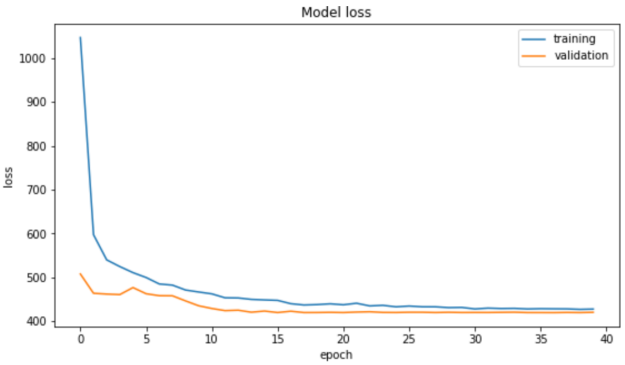
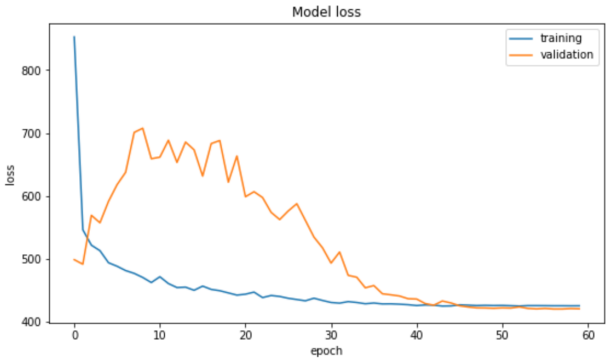
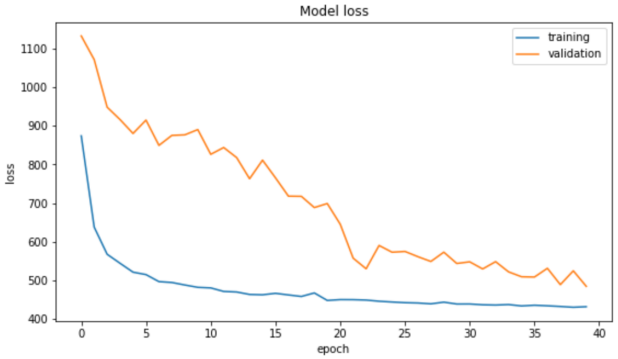
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.6 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.6	Training MSE 427.87 Validation MSE 427.61	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer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

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

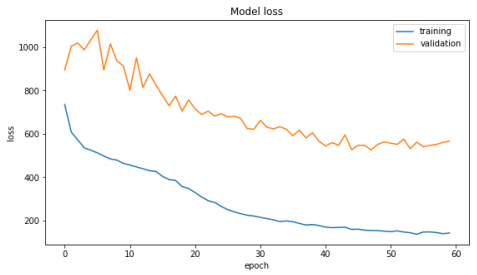
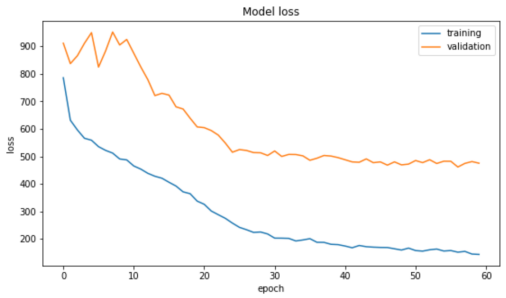
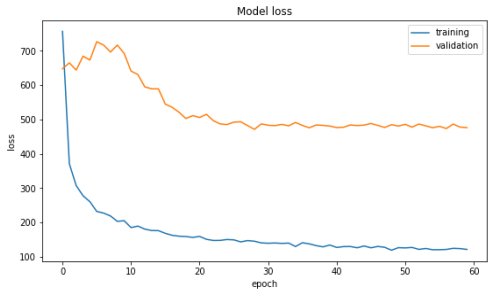
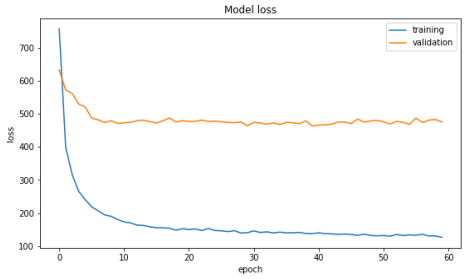
<u>Concatenate</u> Hidden layer 1: 0.7 Hidden layer 2: 0.7		
<u>Tabular</u> Dense layer 1: 0.6 Dense layer 2: 0.6 <u>Convolutional</u> Conv2D layer 1: 0.6 Conv2D layer 2: 0.6 Dense layer 1: 0.6 Dense layer 2: 0.6 <u>Concatenate</u> Hidden layer 1: 0.6 Hidden layer 2: 0.6	<u>Training MSE</u> 425.39 <u>Validation 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	

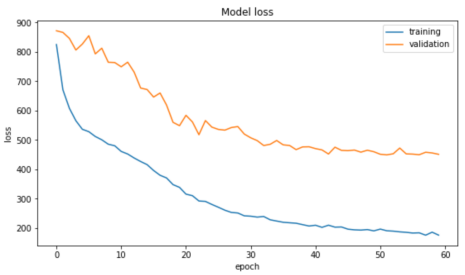
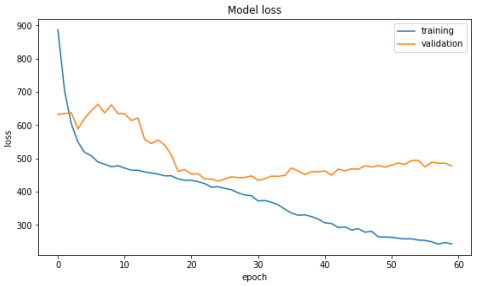
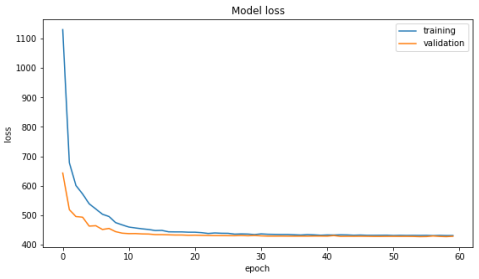
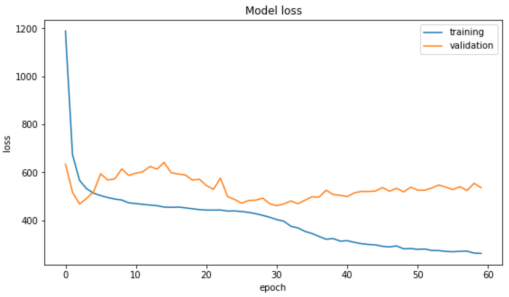
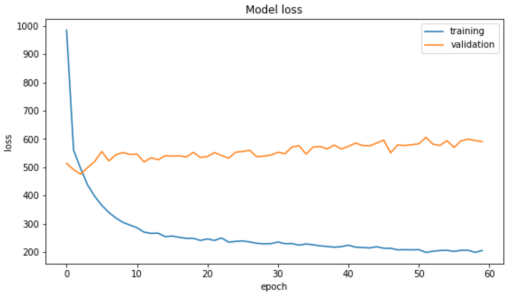
Tabular Dense layer 1: 0.5 Dense layer 2: 0.4 Convolutional Conv2D layer 1: 0.5 Conv2D layer 2: 0.5 Dense layer 1: 0.4 Dense layer 2: 0.4 Concatenate Hidden layer 1: 0.3 Hidden layer 2: 0.3	Training MSE 433.03 Validation MSE 504.42	
Tabular Dense layer 1: 0.4 Dense layer 2: 0.4 Convolutional Conv2D layer 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	

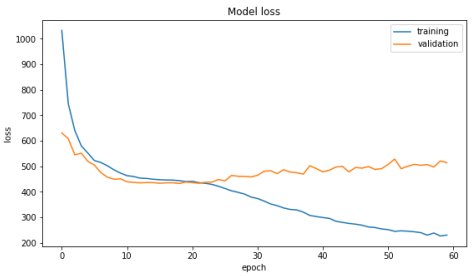
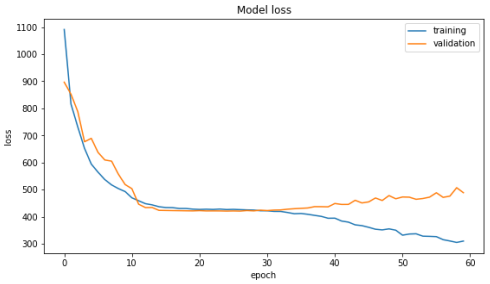
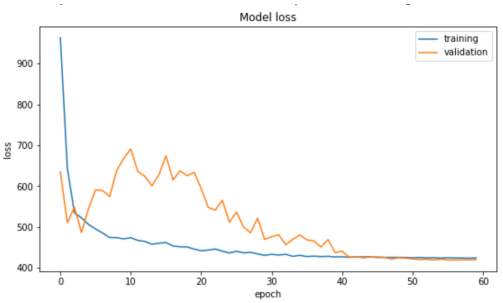
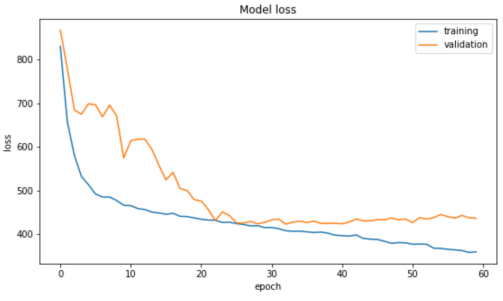
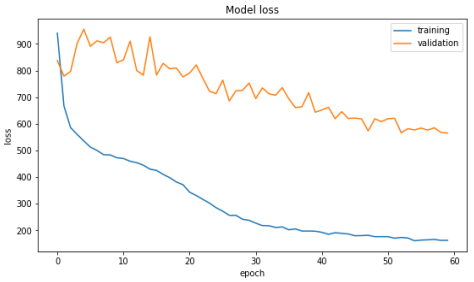
<u>Tabular</u> Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Convolutional</u> Conv2D layer 1: 0.5 Conv2D layer 2: 0.5 Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4	<u>Training MSE</u> 432.71 <u>Validation MSE</u> 486.46	
<u>Tabular</u> Dense layer 1: 0.5 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	

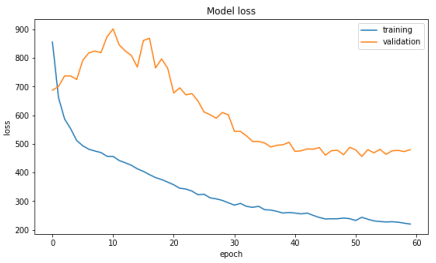
<u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4		
<u>Tabular</u> Dense layer 1: 0.4 Dense layer 2: 0.4 <u>Convolutional</u> Conv2D layer 1: 0.4 Conv2D layer 2: 0.4 Dense layer 1: 0.3 Dense layer 2: 0.3 <u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4	<u>Training MSE</u> 427.99 <u>Validation 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.*

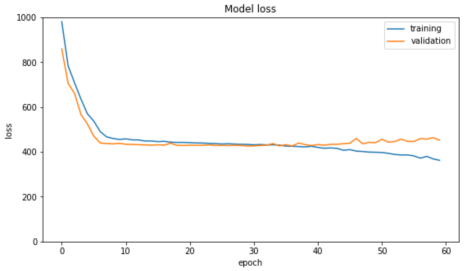
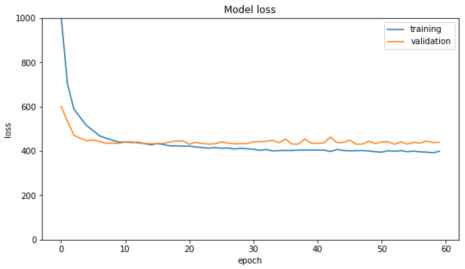
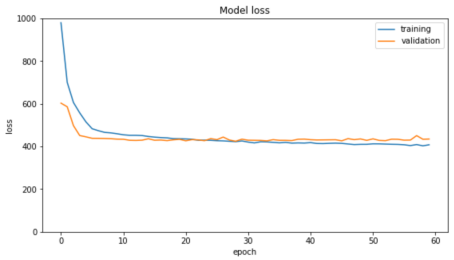
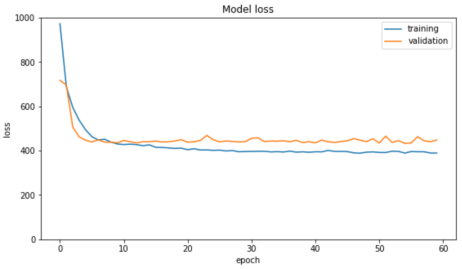
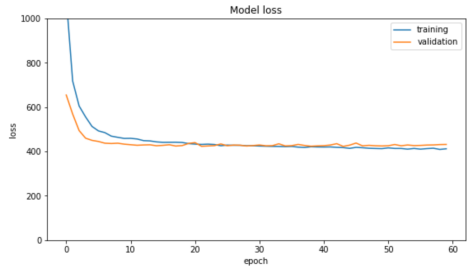
Regularization	**Model MSE (MSE)**	**Model MSE graph**
Concatenated layers Kernel regularizer: L2 (ƛ=1e-4) Bias regularizer: L2 (ƛ=1e-4) Activity regularizer: L2 (ƛ=1e-5)	Training MSE 142.23 Validation MSE 566.46	
Concatenated layers Kernel regularizer: L2 (ƛ=1e-3) Bias regularizer: L2 (ƛ=1e-3) Activity 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	

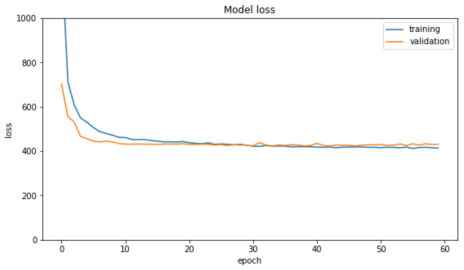
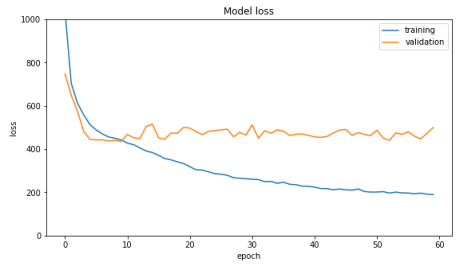
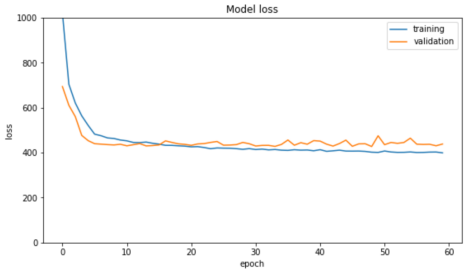
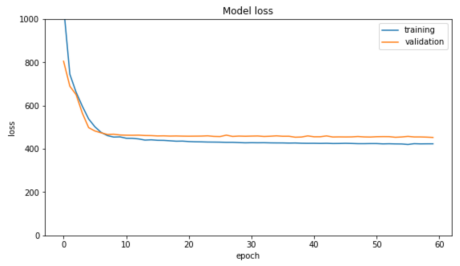
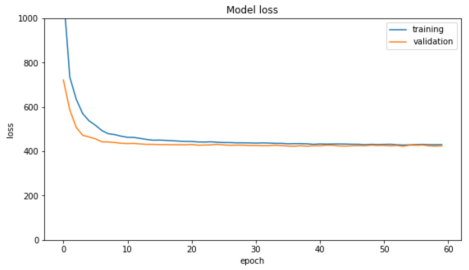
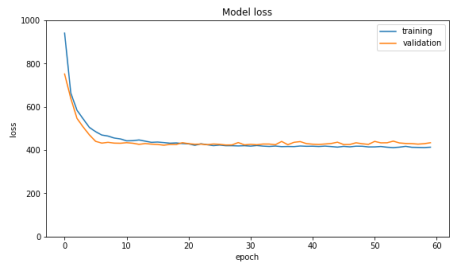
<u>Dense layers CNN</u> Kernel regularizer: L2 (ƛ=1e-3) Bias regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3) <u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-3) Bias regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3)	<u>Training MSE</u> 175.96 <u>Validation MSE</u> 451.23	
<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	Training MSE: 362.53 Validation MSE: 452.63	
Rotation range: 50 degrees Horizontal flip: True	Training MSE: 399.09 Validation MSE: 439.70	
Rotation range: 60 degrees	Training MSE: 407.90 Validation MSE: 434.88	
Rotation range: 60 degrees Horizontal flip: True	Training MSE: 389.25 Validation MSE: 448.35	
Rotation range: 70 degrees	Training MSE: 412.13 Validation MSE: 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