

Report “Pawpularity”

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

Chapter 1: Basic model

Introduction:

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

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

The Project

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

First version model

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

Data analysis and preprocessing

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

Secondly, we looked at the image data and saw that not all images had the same size and orientation. In order to correct this, we reshaped each of the images to images of the size 64 x 64 pixels.

Furthermore, we sorted the dataframe based on the order of the images so they are indexed to the right tabular data. We used this combined data in our Convolutional Neural Network model. Thirdly, we splitted the training data into a training data set (80%) and a validation data set (20%). We started with shuffling the image - and tabular data for the reason that the validation - and training data can be chosen randomly. Then, we sorted the image data, in order that the image - and the corresponding tabular data had the same index. This way, we knew for certain that the same image - and tabular data was selected for the validation data. We choose the first 20% of the sorted data as validation data and the other 80% of the sorted data was assigned to be the training data.

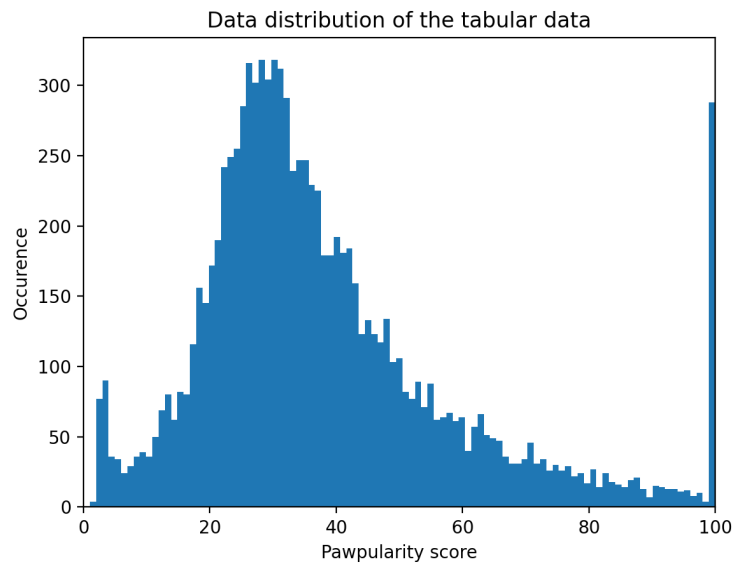


Figure 1: Histogram of the data distribution of the tabular data. On the x-axis is the Pawpularity score presented and on the y-axis the occurrence of each score. The graph is slightly right-skewed with an outlier at the score of 100.

Model Pipeline and Training

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

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

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

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

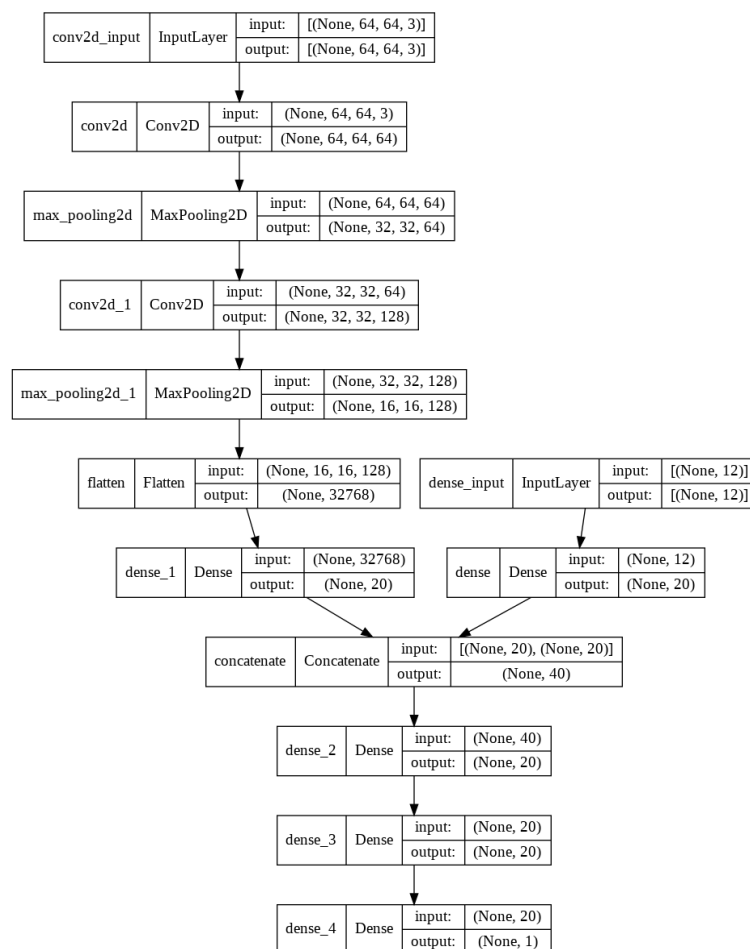


Figure 2: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below).

Evaluation and conclusions

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

loss in orange, while the model is being trained. The final training loss is 345.1967 and the final validation loss is 482.2912. Figure 4 shows the training metric in blue and the validation metric in orange. The final training metric is 18.57 and the final validation metric is 21.96.

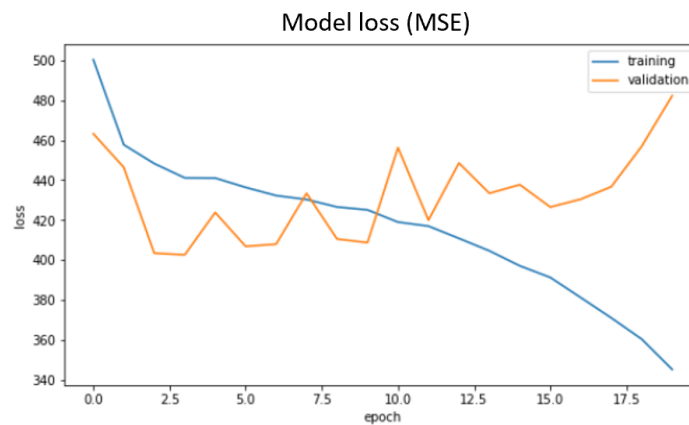


Figure 3: Graph showing the training (blue) and validation (orange) loss (mean squared error) of the concatenated model on each epoch during training the model.

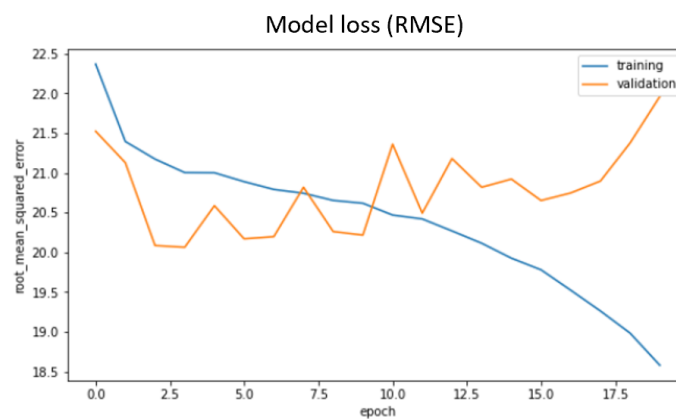


Figure 4: Graph showing the training (blue) and validation (orange) root mean squared error of the concatenated model on each epoch during training the model.

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

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

Chapter 2: Mean centering and standard deviation normalization

Introduction

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

Data analysis and preprocessing

We added the data preprocessing methods mean centering and standard deviation normalization, using the function `preprocessing.image.ImageDataGenerator(** preprocess)` from the Tensorflow Keras library. We applied this function to the train and validation data, by which the training - and validation data are still comparable. We chose the preprocessing features: 'featurewise_center' and 'featurewise_std_normalization'. We only applied this preprocessing to the image data because the tabular data already was normalized, since the tabular data is binary.

In order to add preprocessing, we needed to split the data in a training and validation set. To ensure that both the tabular and image data of the samples were split the same, we did the following. We first shuffled the image data, together with the names of the images using `zip`, to ensure the names still matched the images. We then sorted the dataframe with the tabular data based on the shuffled images. We reset the index of the dataframe, so the index matches the index of the images. After this we took the first 20% samples for the validation set and the last 80% for the training set. Because we shuffled at first, this split was randomly and not dependent on the sequence of the image data.

Model pipeline and training

In order to apply the preprocessing, we created a new function where we train and evaluate the model. This "train_and_evaluate" function applied the preprocessing before it fits the model. We fit the model on the combined data of the tabular and image data.

Since we created a new function, we wanted to confirm that it was working properly and handled all data correctly. Therefore, we applied the `train_and_evaluate` function only on the convolutional network for the image data. We compared the loss values of the concatenated network with the convolutional network and examined if the concatenated model indeed performed better than the convolutional NN only.

Evaluation and conclusions

The concatenated model has a final training loss of 86.53 and a final validation loss of 581.29. The convolutional model has a final training loss of 106.22 and a final validation loss of 603.06. Therefore we can conclude that the concatenated model is performing better than the convolutional model because the concatenated model has a lower validation loss. Figure 5 shows the comparison of the concatenated model loss and the convolutional model loss with the training metric in blue and the validation metric in orange.

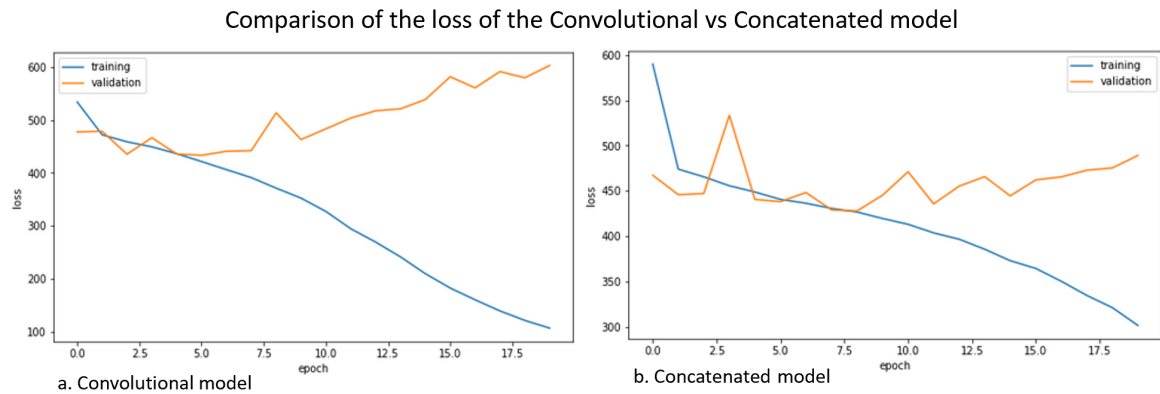


Figure 5: Graph showing comparison of the loss (mean squared error) of the training (blue) and validation (orange) of the convolutional (a) - and concatenated model (b) on each epoch during training the model.

The previous final training loss was 345.20 and the final validation loss was 482.29 for the model that did not use preprocessing. The current model has a final training loss of 301.49 and a final validation loss of 489.08. Concluding, the training loss decreased a lot, whereas the validation loss stayed the same. Therefore, the use of preprocessing methods did not solve the problem of overfitting. However, looking at figure 5b, the concatenated model seems to diverge less than the convolutional model. This means that the addition of the tabular data does help the model to learn. Since the model is still overfitting, other means should be used in the following versions. For instance, dropout and batch normalization could be used.

Chapter 3: Dropout

Introduction

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

Data analysis and preprocessing

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

Model pipeline and training

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

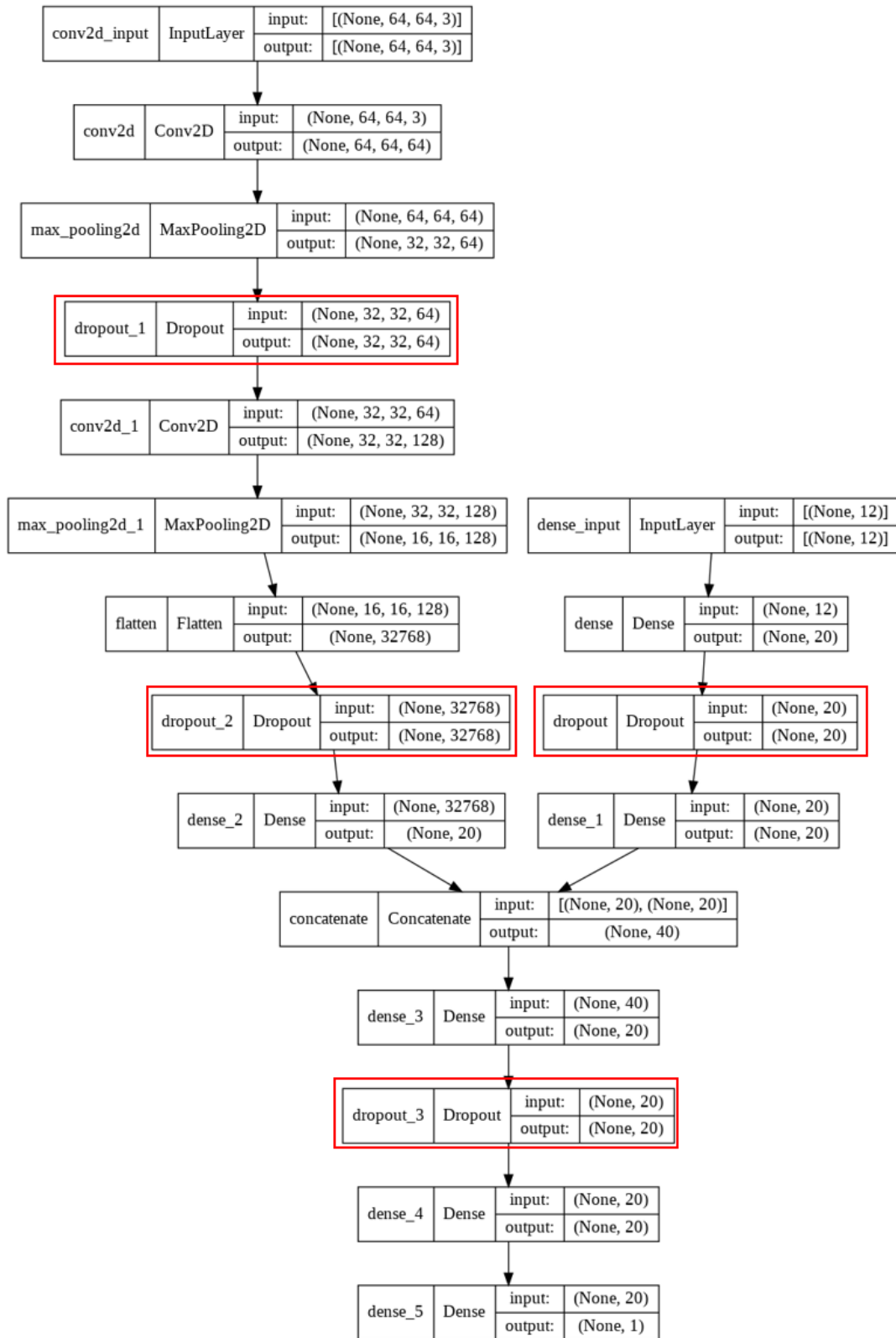


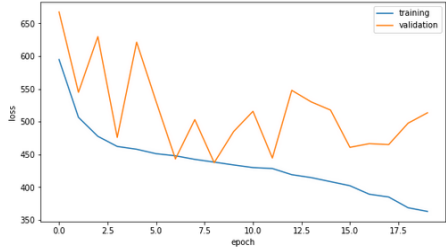
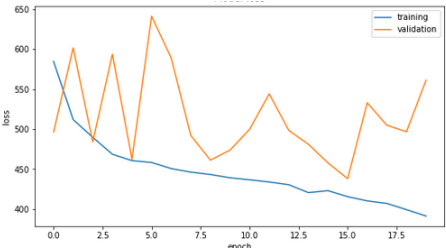
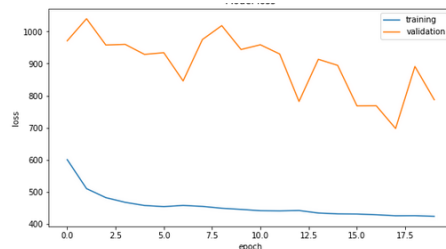
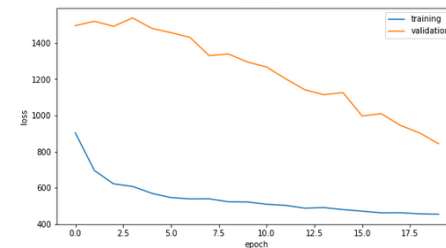
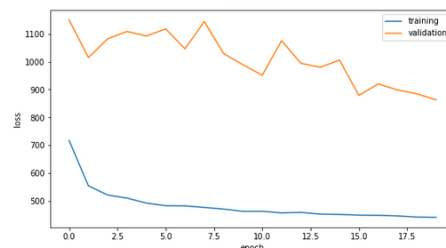
Figure 6: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below) with dropout (red).

Evaluation and conclusions

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

Table 1. MSE loss for different probabilities of Drop out Layers. A dropout of 0.8 gave the best trend for the validation loss.

Dropout Layers (before)	MSE loss	Model Loss Graph
<u>Tabular</u> Dense Layer: 0.4 <u>Convolutional</u> Conv2D layer: 0.4 Dense layer: 0.2 <u>Concatenated</u> Hidden layer: 0.2	<u>Training loss:</u> 303.0349 <u>Validation loss :</u> 520.6554 <u>Difference:</u> 217.6205	
<u>Tabular</u> Dense layer: 0.2 <u>Convolutional</u> Conv2DI layer: 0.4 Dense layer: 0.2 <u>Concatenated</u> Hidden layer: 0.2	<u>Training loss:</u> 136.8040 <u>Validation loss:</u> 587.4723 <u>Difference:</u> 450.6683	
<u>Tabular</u> Dense layer: 0.5 <u>Convolutional</u> Conv2d layer: 0.4 Dense layer: 0.2 <u>Concatenated</u> Hidden layer: 0.2	<u>Training loss:</u> 90.8209 <u>Validation loss:</u> 668.6118 <u>Difference:</u> 577.7909	
<u>Tabular</u> Dense layer: 0.2 <u>Convolutional</u> Conv2D layer: 0.2 Dense layer: 0.2 <u>Concatenated</u> Hidden layer: 0.4	<u>Training loss:</u> 356.9272 <u>Validation loss:</u> 539.2504 <u>Difference:</u> 182.3232	

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

The graph of the dropout rate of 0.8 for the tabular, convolutional and concatenated network showed the best trend downwards for the validation loss. Therefore, we chose this dropout rate and increased the number of epochs to 40 to investigate whether this trend will continue. After the increase in the number of epochs, the training loss is 427.9708 and the validation loss is 423.1221. The results are shown in figure 7.

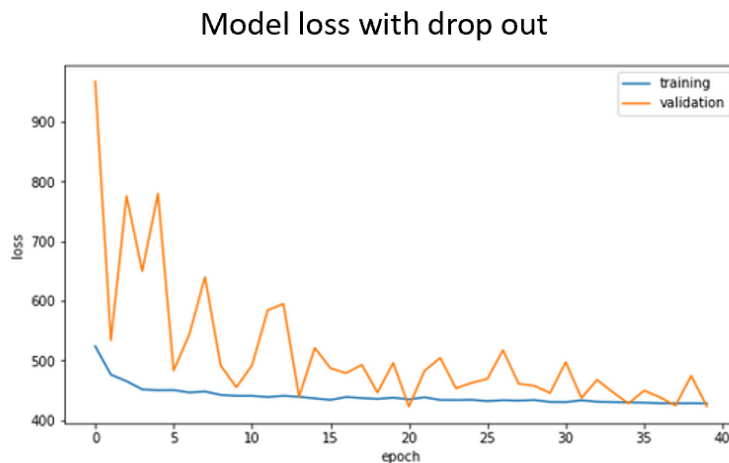


Figure 7: The model with a dropout rate of 0.8 for the the tabular, convolutional- and concatenated network shows less overfitting. The training loss is 427.9708 and the validation loss is 423.1221 with 40 epochs.

The model with the dropout layers has a training loss of 427.97 and a validation loss of 423.12 (figure 7). The validation loss is therefore lower than the training loss, which means that the model does not overfit anymore. The current training loss is 427.97, the previous training loss was 301.49. This means that the training loss increased using dropouts. This means that we lost a lot of information due to dropout, so the model's learning performance worsens.

For our experiment, we will first try to make the model learn better, by adding extra layers. If that succeeds, we want to try to finetune the dropout rate in order to retain more information. We will try to lower the dropout rate, while still preventing the model from overfitting.

Chapter 4: Adding more 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 loss is still fluctuating a lot. Therefore, we will try to add several layers to create a deeper network that is able to learn better, so the training and validation loss will decrease. We will still use a dropout rate of 0.8, to isolate only the effect of adding extra layers.

Data analysis and preprocessing

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

Model pipeline and training

We tried two new versions of the model. In the first version, we added one extra layer in the tabular neural network, with a dropout rate of 0.8 and 20 hidden nodes and relu activation. In the convolutional neural network we added one more convolutional layer and one extra dense layer, with 20 hidden nodes. In the concatenated model we added one dense layer as well. An overview of this neural network is shown in figure 8. The number of epochs is again set to 40. In the second version, we added another convolutional layer.

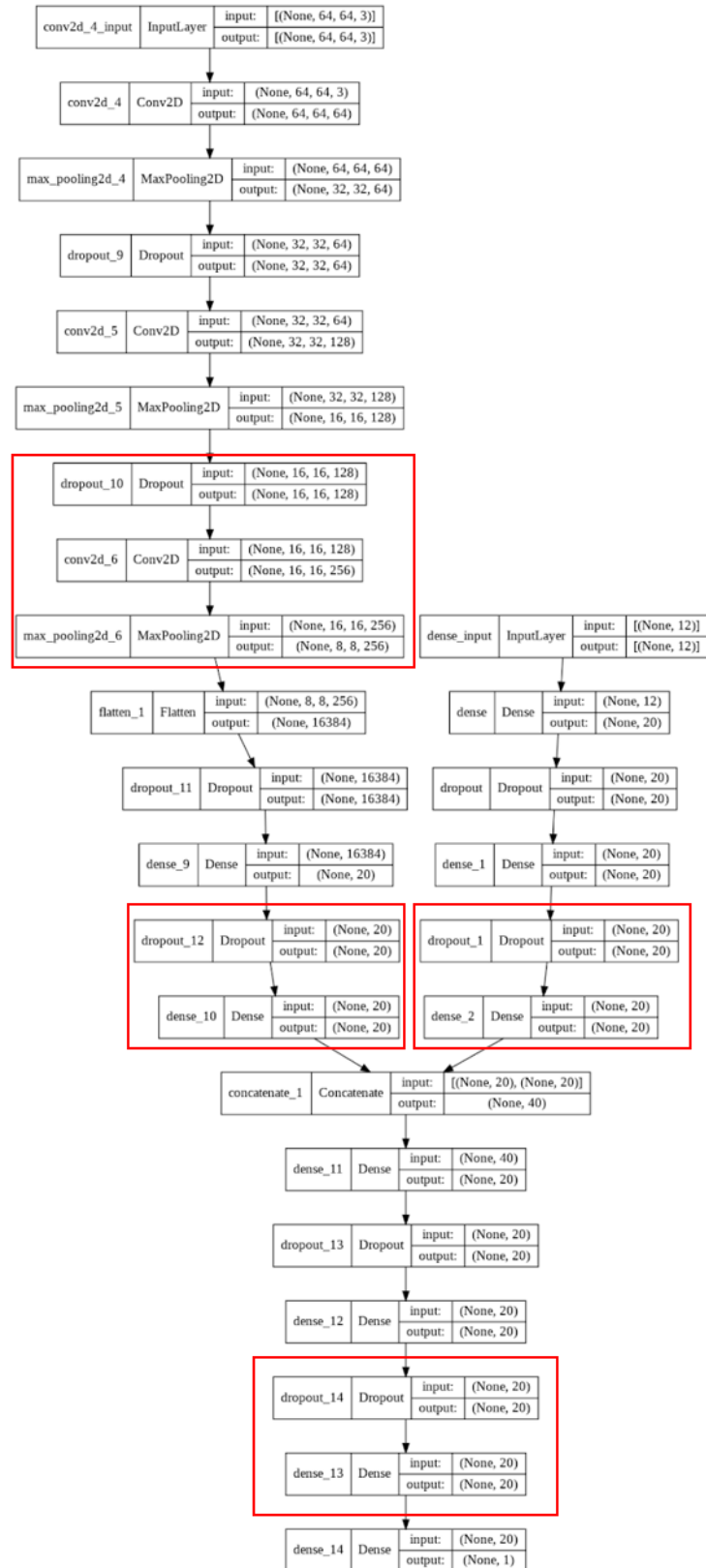
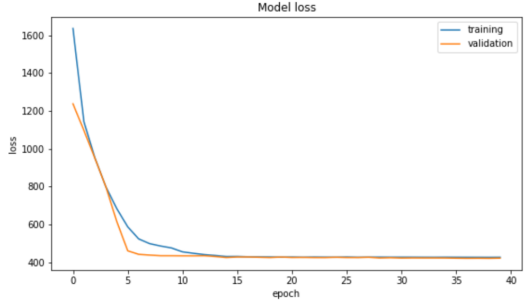
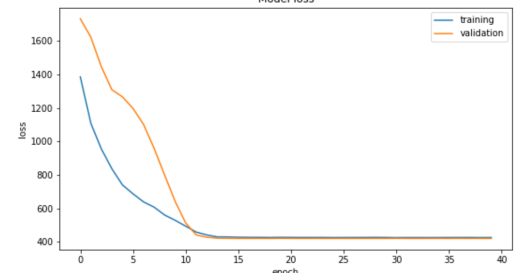


Figure 8: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below). The new layers are highlighted in red. This is the network we will use in later versions.

Evaluation and conclusions

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

Table 2. Model and RMSE loss when adding more layers. Adding just a few layers to the existing models resulted in a fairly good result.

Layers	Model loss (MSE)	Model loss (MSE) graph
<u>Tabular</u> 1 extra layer <u>Convolutional</u> 1 extra convolutional layer 1 extra dense layer <u>Concatenated</u> 1 extra dense layer	<u>Training loss</u> 425.85 <u>Validation loss</u> 421.31	
<u>Tabular</u> 1 extra layer <u>Convolutional</u> 2 extra convolutional layers 1 extra dense layer <u>Concatenated</u> 1 extra dense layer	<u>Training loss</u> 425.84 <u>Validation loss</u> 420.21	

For the model with one extra layer in each submodel, the training and validation loss decrease as the model is being trained. The validation loss is lower than the training loss, meaning that the model does not overfit. The same is true for the model with 2 extra convolutional layers. However, in this model the validation loss had a little plateau after 5 epochs. Because the shape of the validation loss seemed better in the model with only 1 extra layer we will continue with this model.

Unfortunately, the loss of the model with only 1 extra layer was not lower than in the previous model (chapter 3). However, the shape of the validation loss now shows less variability. Hence, we will continue with this model. We will try to decrease the dropout per layer to retain more information about the data. We will do this by slowly decreasing the dropout probability per layer. We expect that the neural network can learn specific features better and that the loss will decrease. We have to be cautious that the model does not start to overfit again with a lower dropout rate.