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.

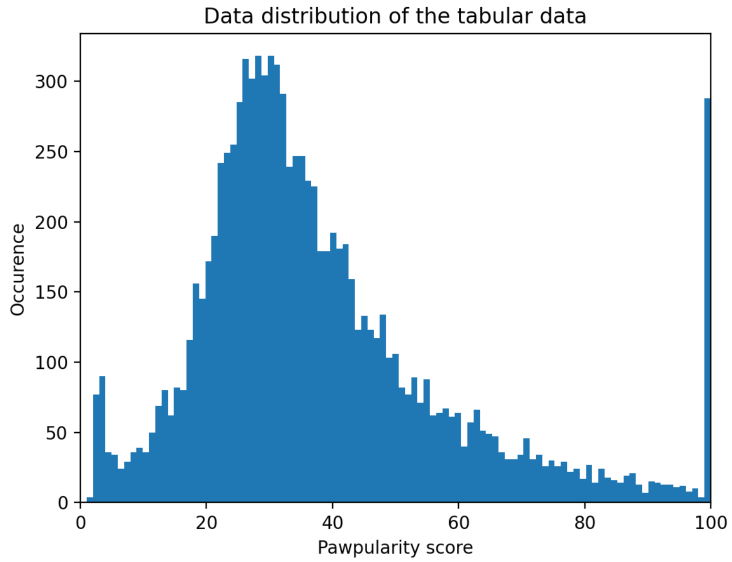


Figure 1: Histogram of the data distribution of the tabular data. On the x-axis is the Pawpularity score presented and on the y-axis the occurrence of each score. The graph is slightly right-skewed with an outlier at the score of 100.

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.

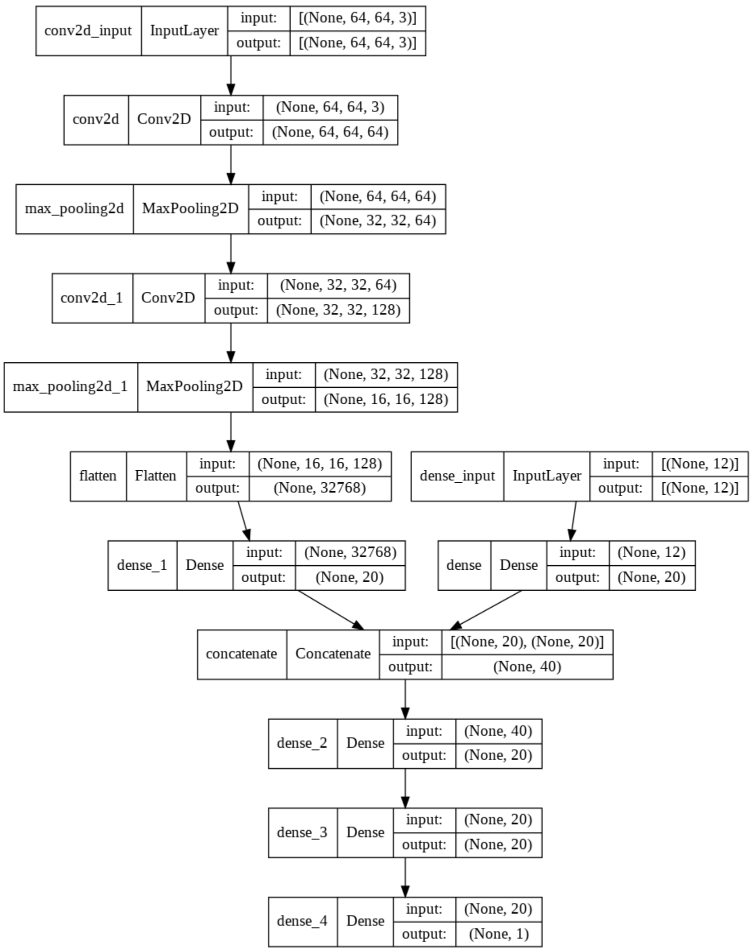


Figure 2: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below).

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 345.20 and the final validation loss is 482.29. Figure 4 shows the training metric in blue and the validation metric in orange. The final training metric is 18.57 and the final validation metric is 21.96.

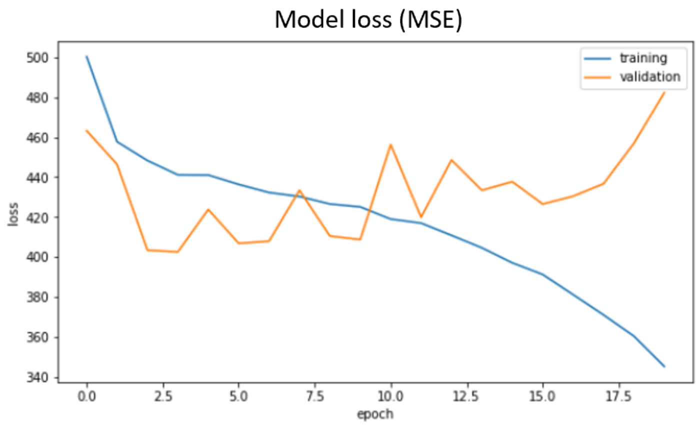


Figure 3: Graph showing the training (blue) and validation (orange) loss (mean squared error) of the concatenated model on each epoch during training the model.

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. We applied this function to the train and validation data, by which the training - and validation data are still comparable. We only applied this preprocessing to the image data, since the tabular data is binary.

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

Model pipeline and training

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

Evaluation and conclusions

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.

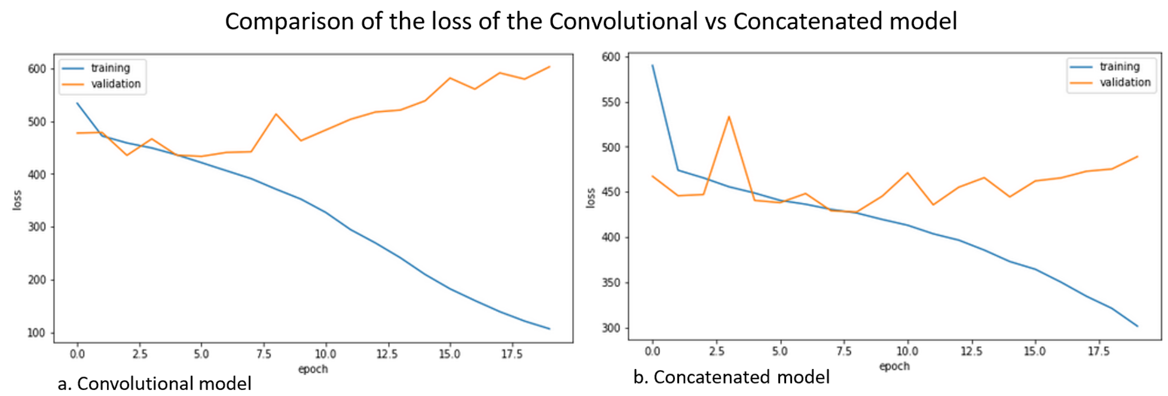


Figure 5: Graph showing comparison of the loss (mean squared error) of the training (blue) and validation (orange) of the convolutional (a) - and concatenated model (b) on each epoch during training the model.

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, whereas the validation loss stayed the same.

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. However, the use of preprocessing methods did not solve the problem of overfitting.

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

Chapter 3: Dropout

Introduction

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

Data analysis and preprocessing

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

Model pipeline and training

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

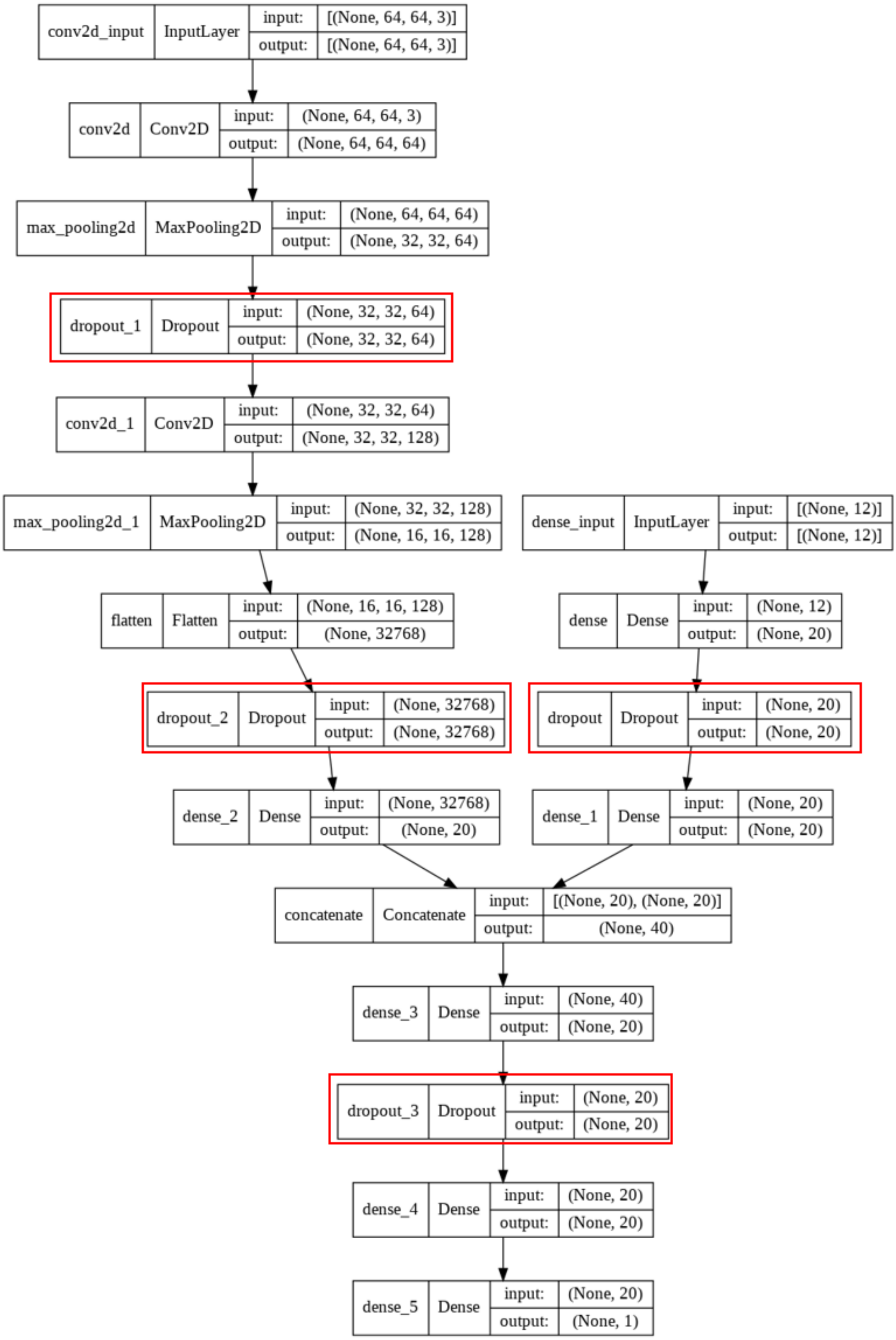
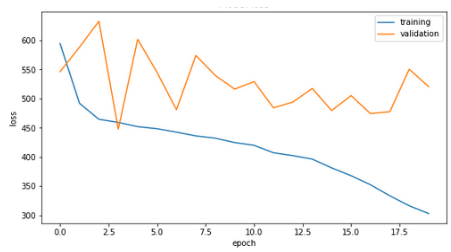
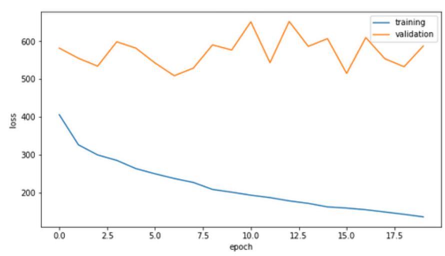
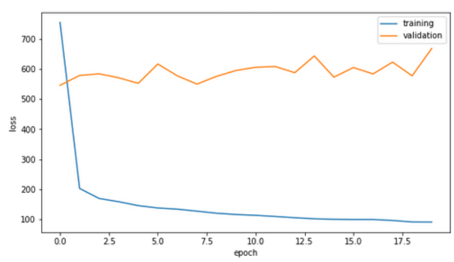
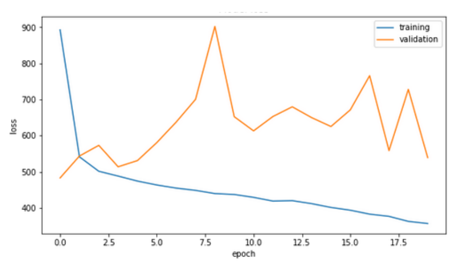


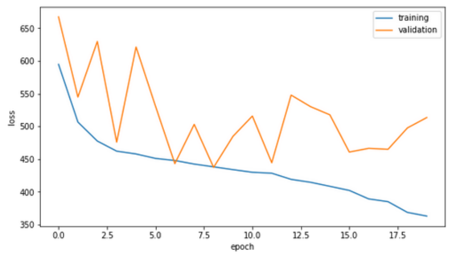
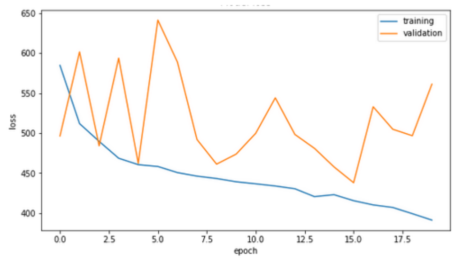
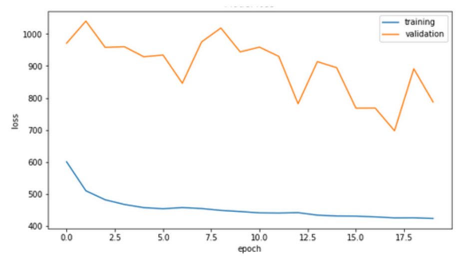
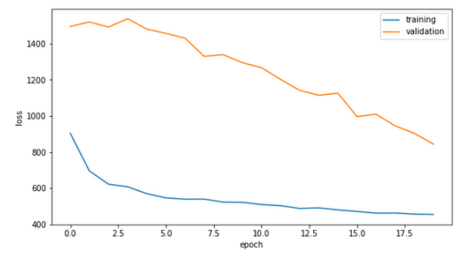
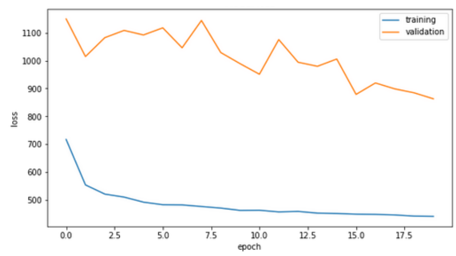
Figure 6: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below) with dropout (red).

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

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

The graph of the dropout rate of 0.8 for the tabular, convolutional and concatenated network showed the best trend downwards for the validation 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.

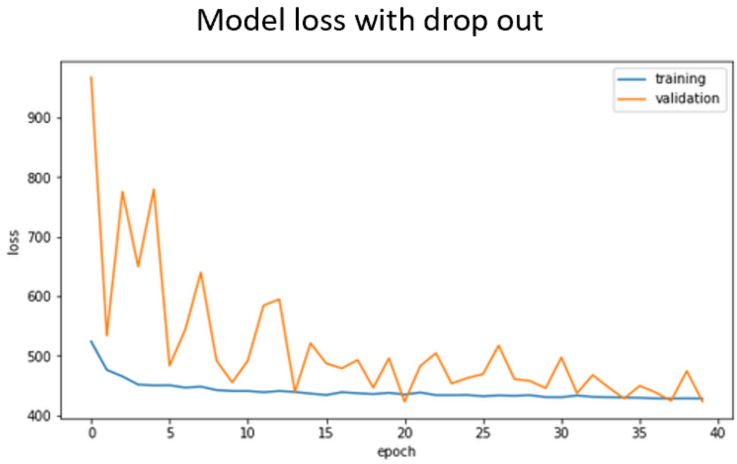


Figure 7: The model with a dropout rate of 0.8 for the the tabular, convolutional- and concatenated network shows less overfitting. The training loss is 427.97 and the validation loss is 423.12 with 40 epochs.

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. 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 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.

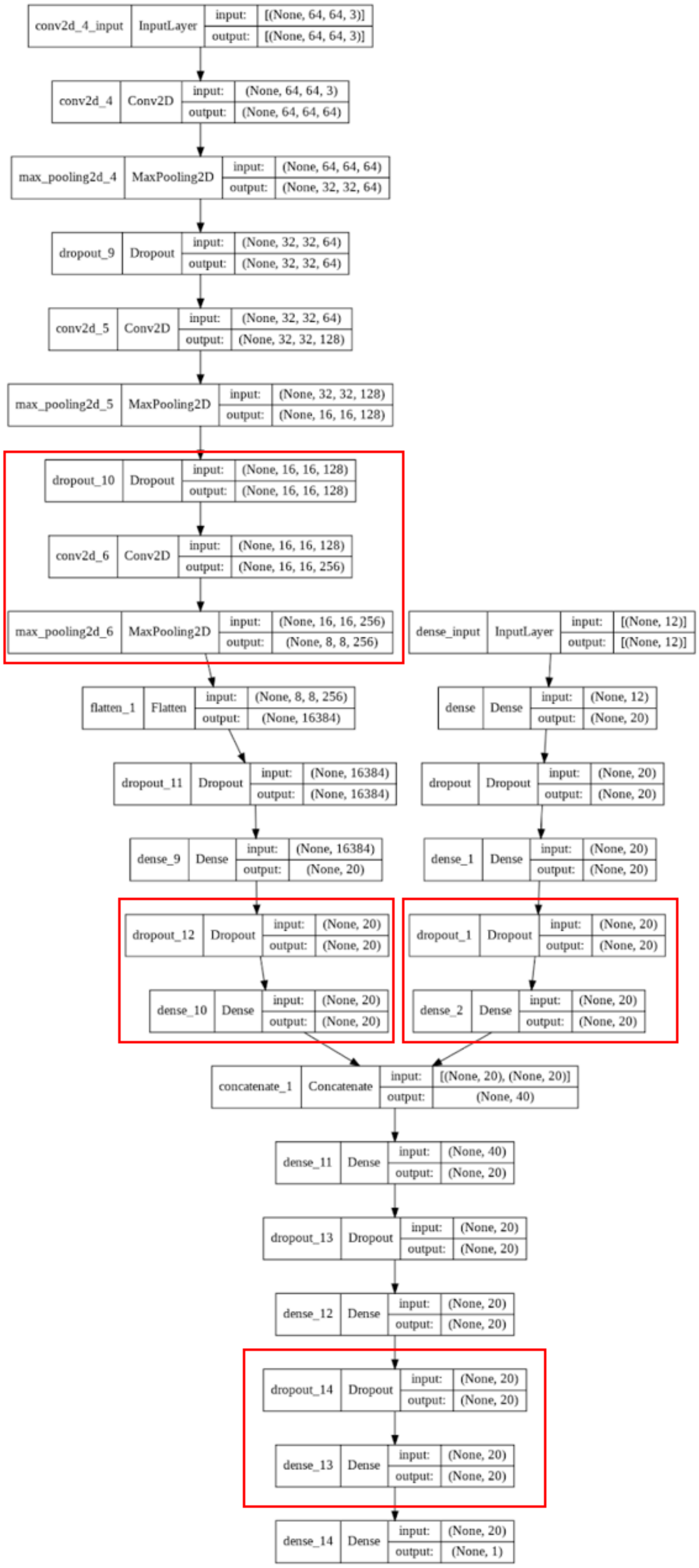
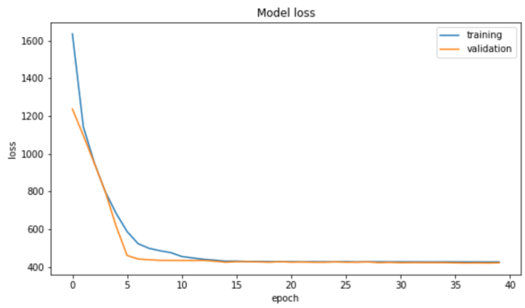
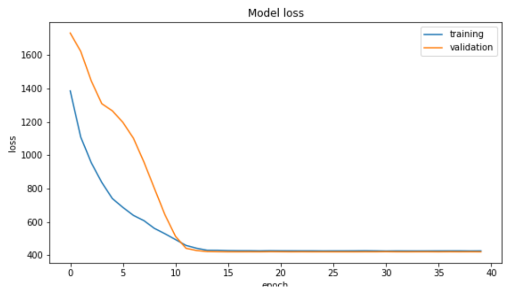


Figure 8: Overview of the layers in the convolutional network for images (left), neural network for tabular data (right) and concatenated network (below). The new layers are highlighted in red. This is the network we will use in later versions.

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 training and validation loss when adding more layers. Adding just a few layers to the existing models resulted in a similar result.

Layers	**Model loss (MSE)**	**Model loss (MSE) graph**
Tabular 1 extra layer Convolutional 1 extra convolutional layer 1 extra dense layer Concatenated 1 extra dense layer	Training loss 425.85 Validation loss 421.31	
Tabular 1 extra layer Convolutional 2 extra convolutional layers 1 extra dense layer Concatenated 1 extra dense layer	Training loss 425.84 Validation loss 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 training loss of the model with only 1 extra layer was not lower than in the previous model from chapter 3 (425 vs 428, respectively). However, the shape of the validation loss now shows less variability, which is therefore easier to reproduce. 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.

Chapter 5: Reducing the dropout rate

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

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

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

Evaluation and conclusions
When using a dropout rate of 0.2 for one or more layers, the model started to overfit again. Eventually, the best option was a model where all layers had a dropout of 0.4, except for the dense layer of the CNN, which had a dropout of 0.3. The best attempt with decreasing the dropout rate is displayed in figure 9. The table with all the attempts can be found in appendix A. Looking at the learning curves and the final training and validation loss, we choose to continue with the model that has each dropout rate set to 0.4, except for the dropout rate in the two dense layers in the CNN, which are set to 0.3. We choose this model, because the validation loss (420.33) is lower than the training loss (425.17) and the learning curves show that the model is not overfitting. Decreasing the dropout rates more leads to the model overfitting again. We also choose to continue using 60 epochs, since it seems like the model is still learning in the period of epoch 40 to 60.

Slowly decreasing the dropout rate led to more fluctuating learning curves, especially for the validation loss. This makes sense, because the model now can learn features more specifically, so it is more sensitive to noise in the data. The final validation loss increases because the model is overfitting the training data more.

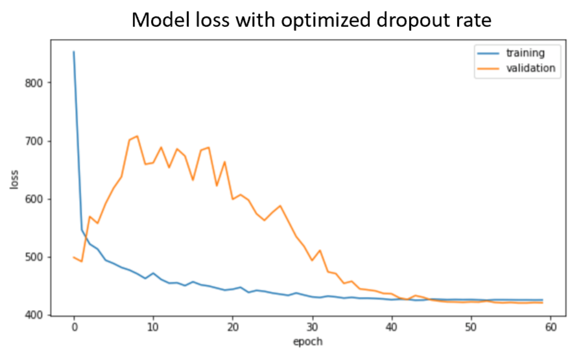


Figure 9: The model with a reduced dropout rate still performs well and does not overfit

Chapter 6: Batch normalization

Introduction

After adjusting the dropout rate the model seemed to converge after 40 epochs. To increase the learning speed of the network we will apply batch normalization. When performing batch gradient descent, the input range varies greatly between batches. Batch normalization will reduce this variance by applying a scaling factor to them. After this, all weight vectors will be closer to each other which will lead to more stable gradients and thus faster convergence. The network is also able to learn better because the individual neurons will not react strongly to small changes. It might eventually lead to a reduction of the number of training epochs required to train the network.

Data analysis and preprocessing

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

Model pipeline and training

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

Evaluation and conclusions

The outcome of adding batch normalization is depicted in figure 10. After applying batch normalization, the training loss has decreased from 425.17 after reducing the dropout rate to 148.97. This is a significant decrease of more than 60%. While the training loss turned out to be relatively low, the validation loss increased with batch normalization. It increased from 420.33 to 498.32. The cause for this might be that our model does not generalize well enough to new samples and overfits. In order to solve this problem we will add a regularization term in chapter 8.

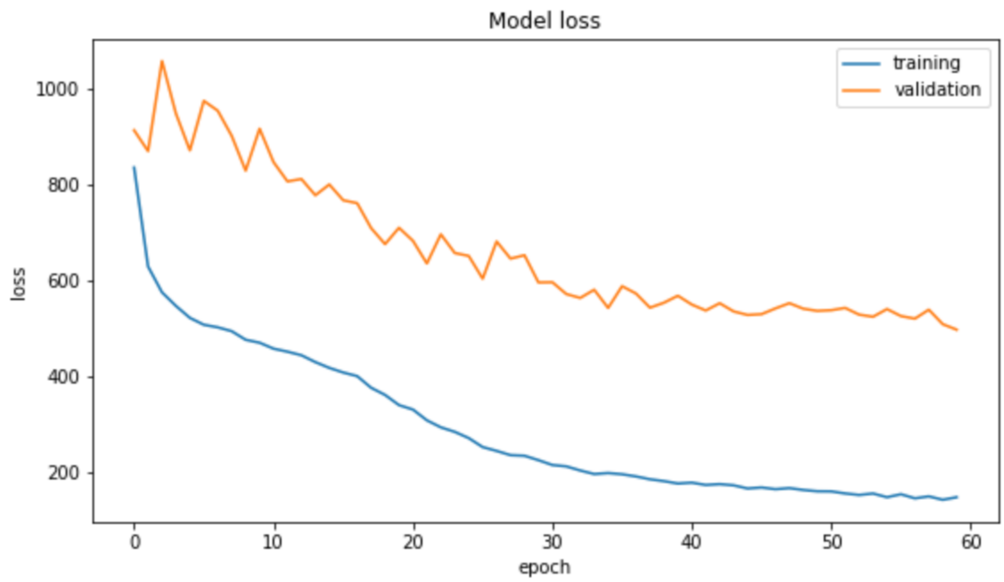


Figure 10: The model with several batch normalization layers seems to overfit again on the training data.

Chapter 7: Linear output

Introduction

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

Data analysis and preprocessing

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

Model pipeline and training

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

Evaluation and conclusions

The model using the clipped linear output still gives a fairly good result for the model loss compared to our previous version. The new training loss was 425.39 compared to 427.99 and the new validation loss was 429.38 compared to 420.66. The results are displayed in figure 11. In conclusion, the model did not improve and had similar results compared to our previous model.

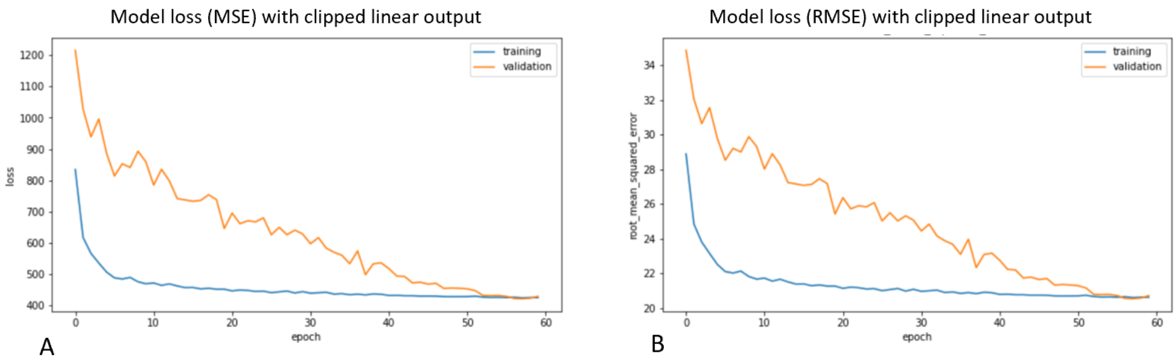


Figure 11: The model using a clipped linear output still gives a comparable model loss (MSE) (A)for the training loss (blue) of 425.39 and the validation loss (orange) of 429.38. (B) the training RMSE (blue) was 20.62 and the validation RMSE (orange) was 20.72.

Chapter 8: Regularization term

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

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

Model pipeline and training
Different types of regularization will be tried. In each layer, it is possible to add kernel, bias and activity regularization. The options of regularization are L1, L2 or L1 and L2 combined
From table 1, it seems like overfitting arises in the concatenated model, so regularization terms will be added in these layers as a starting point. Regularization terms will also be tried in the dense layers of the CNN. Different values for the regularization parameter lambda will be used. We will use L2 regularization, since this is the most common type.

Evaluation and conclusions
After several tries of many different options we figured that it was not possible to get our model to the point where it would perform better on the validation data than before but still remain a low training loss. All tries are visible in Appendix B. We eventually chose a regularization term where the training loss was still relatively low and where the validation loss was not too high. This model is still overfitting but less than the other options. It also seems to perform well before 30 epochs. We used an L2 kernel, bias and activity regularizer of 1e-3 on the dense layers of our CNN and changed lambda to 1e-1 for the dense layers of the concatenated model. This led to a training loss of 230.49 and a validation loss of 514.46. In further versions we will try to train the model better on the small amount of provided data by using k-fold cross-validation and data augmentation in order to prevent overfitting.

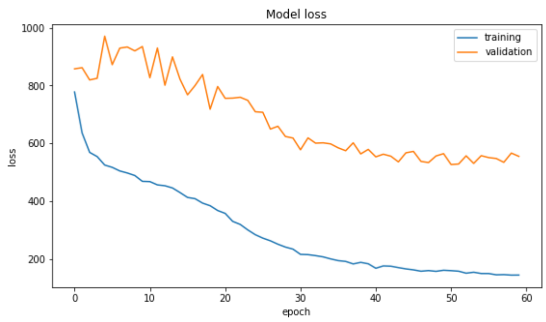


Figure 12: Learning curve of the model with a clipped linear output and batch normalization, before adding regularization terms.

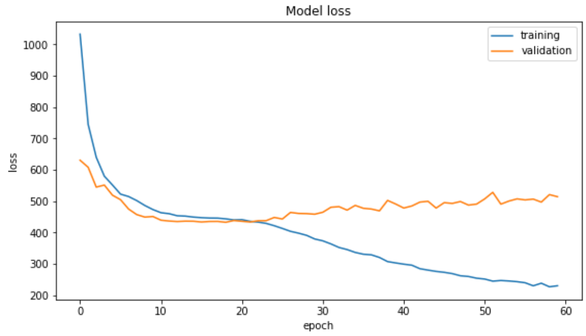


Figure 13: Learning curve of the model after adding a L2 regularization term of 1e-3 on the 2 dense layers in the CNN and a term of 1e-1 on the dense layers of the concatenated model.

Chapter 9: K-fold

Introduction

The dataset that is available for this model is relatively small. That is why the model would benefit from applying k-fold cross-validation. It is a method that trains and validates the model k times with each time a different training and validation set. This way, all data can be used in training the model and the validation loss of the model will be determined more accurately. Using k-fold cross-validation will thus give us a better idea of how the model works.

Data analysis and preprocessing

In the versions before, the tabular and image data were split in a training (80%) and validation (20%) set. However, this is not needed when using k-fold cross-validation, so these steps are removed from the data preprocessing. The non-split tabular and image data are used for k-fold. The data is split in x- and y-values, with the x-values being the features in the tabular data and the images in the image data, and the y-values being the 'Pawpularity' scores. The k-fold function shuffles the data and splits the data in k equal parts.

Model pipeline and training

The number of folds (k) is first set to 5, so the data is split in 5 parts. Each time the model is being trained, another part is used as the validation set and the remaining 4 parts are used as the training set. Thus, the model is being trained 5 times, with 5 different learning curves. The training and validation loss are stored and at the end to compute the average training and validation loss. Besides 5-fold, we also try 10-fold cross-validation.

Evaluation and conclusions

The 5 learning curves during 5-fold cross-validation, as shown in figure 14-18, show that the model is overfitting. The training loss is decreasing during training, whereas the validation loss increases after a decrease. The average training loss is 114.83 and the average training RMSE is 9.71. The average validation loss is 379.55 and the average validation RMSE is 18.76. These numbers are lower than in the previous version. It also seems like the model performance improves over the k-fold splits.
Figures 19-23 show the learning curves of the sixth to tenth split in 10-fold cross-validation. It seems like the model is overfitting less in these splits. The average training loss is 90.10 and the average training RMSE is 8.26. The average validation loss is 242.83 and the average validation RMSE is 14.39. These losses are much lower than in previous versions, so it appears the model is learning better.
In future versions, we can try data augmentation to further artificially increase the dataset. Furthermore, we can try to remove outliers at the 'Pawpularity' score of 100, so the scores are normally distributed. This may also lead to a better learning model.

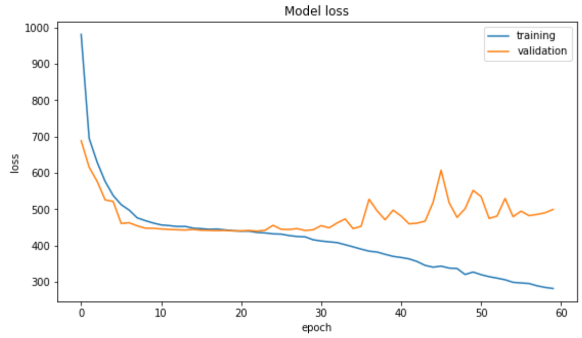


Figure 14: Learning curve of training and validation loss during first k-fold split

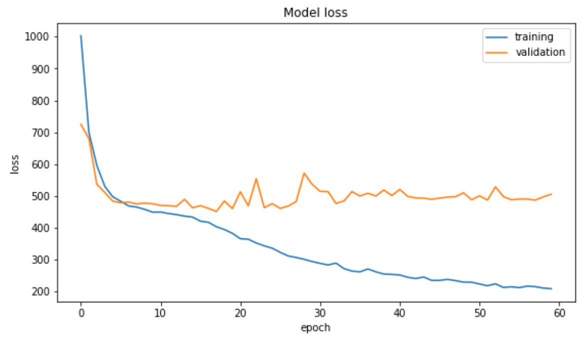


Figure 15: Learning curve of training and validation loss during second k-fold split

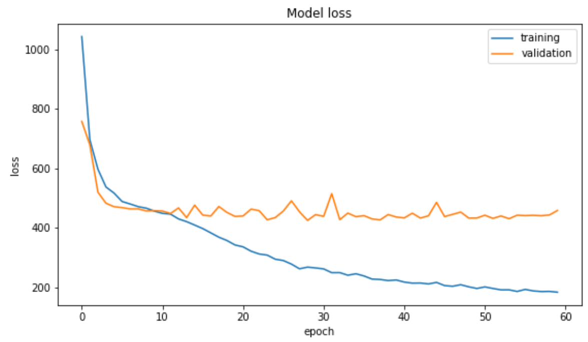


Figure 16: Learning curve of training and validation loss during third k-fold split

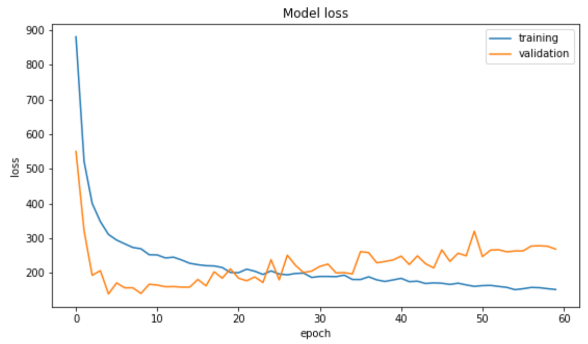


Figure 17: Learning curve of training and validation loss during fourth k-fold split

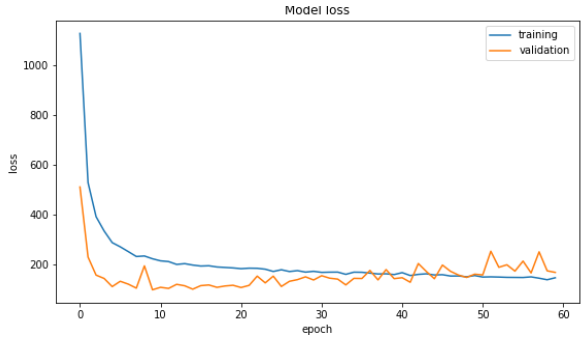


Figure 18: Learning curve of training and validation loss during fifth k-fold split

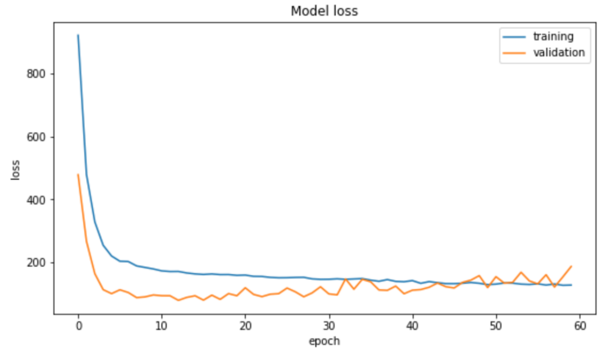


Figure 19: Learning curve of training and validation loss during sixth k-fold split

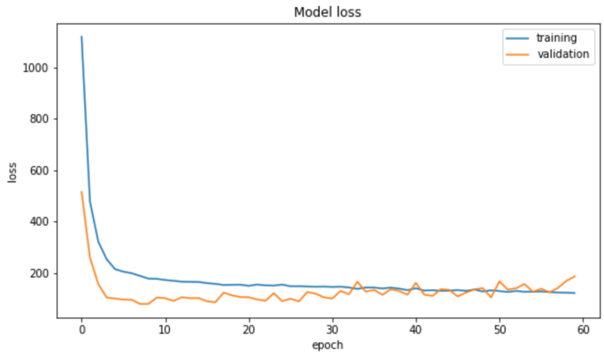


Figure 20: Learning curve of training and validation loss during seventh k-fold split

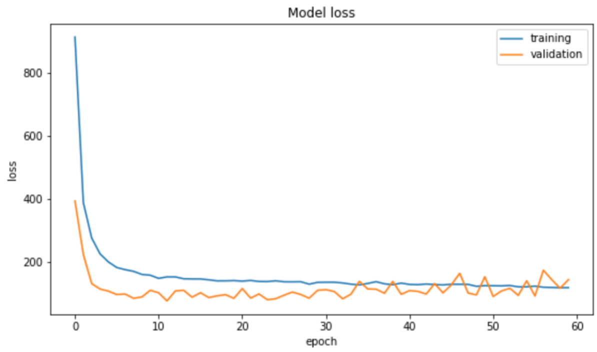


Figure 21: Learning curve of training and validation loss during eight k-fold split

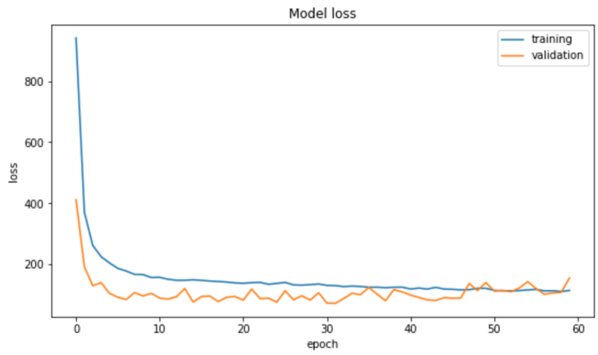


Figure 22: Learning curve of training and validation loss during ninth k-fold split

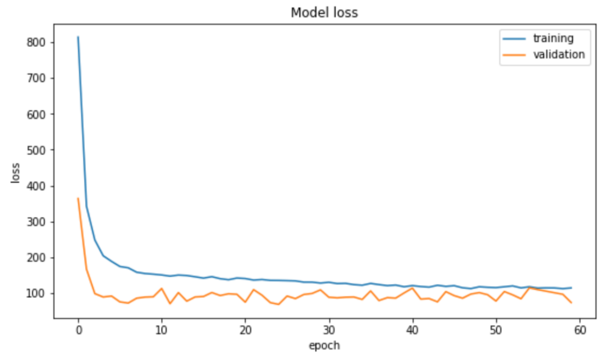
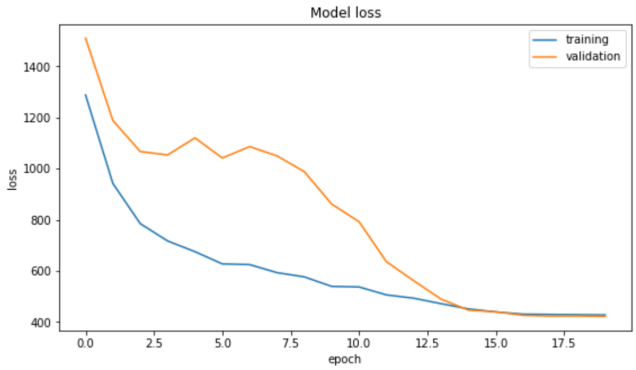
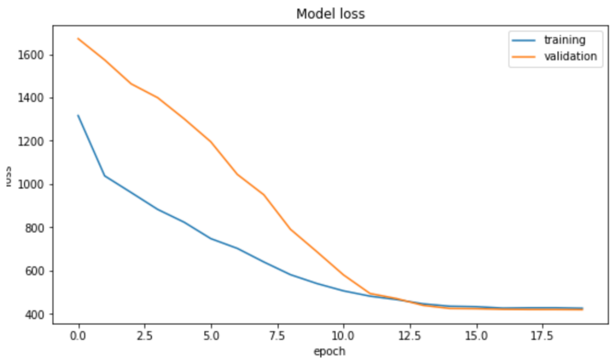
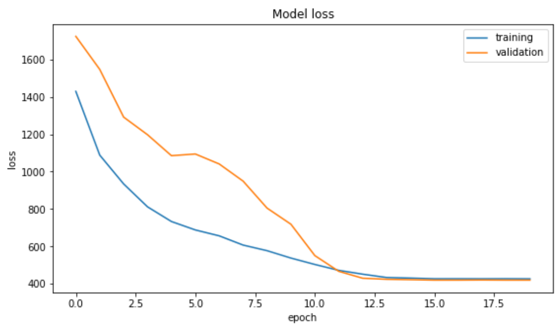


Figure 23: Learning curve of training and validation loss during tenth k-fold split

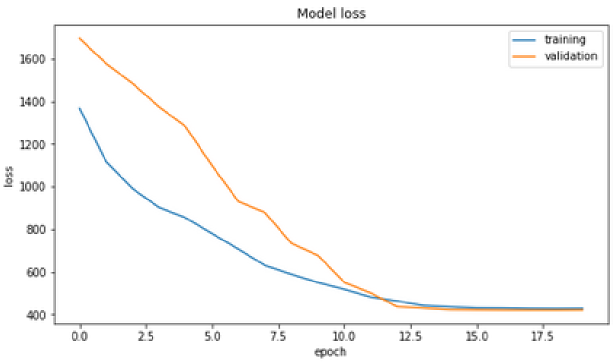
Appendix A. Table with all the combinations tried to decrease the dropout

Layers	**Model loss (MSE)**	**Model loss graph**
<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: 0.8 Dense layer 2: 0.8 <u>Concatenate</u> Hidden layer 1: 0.8 Hidden layer 2: 0.7	<u>Training loss</u> 427.39 <u>Validation loss</u> 421.44	
<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: 0.8 Dense layer 2: 0.8 <u>Concatenate</u> Hidden layer 1: 0.7 Hidden layer 2: 0.8	<u>Training loss</u> 426.84 <u>Validation loss</u> 420.23	
<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: 0.8 Dense layer 2: 0.7 <u>Concatenate</u> Hidden layer 1: 0.8 Hidden layer 2: 0.8	<u>Training loss</u> 427.08 <u>Validation loss</u> 420.13	

<table>
<tr>
<td>Tabular
Dense layer 1: 0.8
Dense layer 2: 0.8

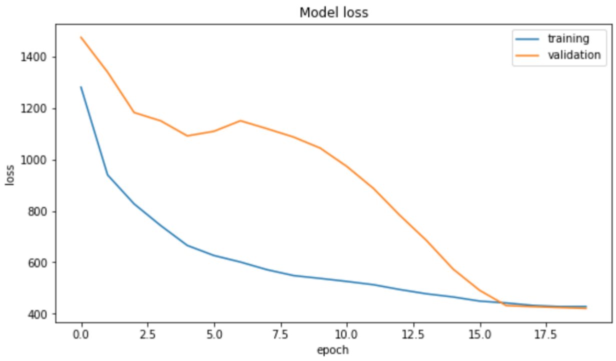
Convolutional
Conv2D layer 1: 0.8
Conv2D layer 2: 0.8
Dense layer 1: 0.7
Dense layer 2: 0.8

Concatenate
Hidden layer 1: 0.8
Hidden layer 2: 0.8</td>
<td>Train loss
427.81

Validation loss
420.66</td>
<td>
</td>
</tr>
<tr>
<td>Tabular
Dense layer 1: 0.8
Dense layer 2: 0.8

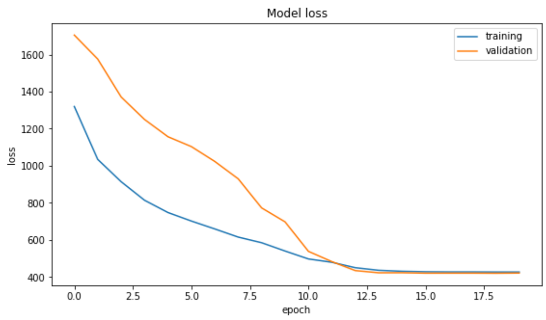
Convolutional
Conv2D layer 1: 0.8
Conv2D layer 2: 0.8
Dense layer 1: 0.8
Dense layer 2: 0.8

Concatenate
Hidden layer 1: 0.6
Hidden layer 2: 0.8</td>
<td>Training loss
427.96

Validation loss
420.84</td>
<td>
</td>
</tr>
<tr>
<td>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

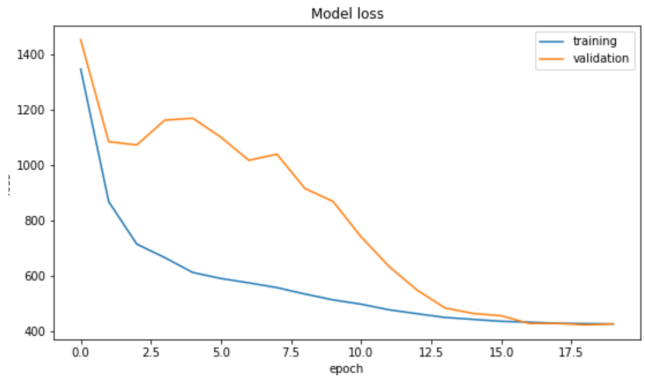
Concatenate
Hidden layer 1: 0.8
Hidden layer 2: 0.8</td>
<td>Training loss
426.75

Validation loss
421.12</td>
<td>
</td>
</tr>
</table>

<table>
<tr>
<td><u>Tabular</u>
Dense layer 1: 0.8
Dense layer 2: 0.8

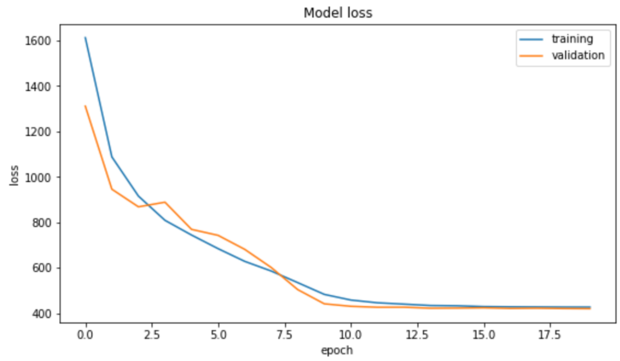
<u>Convolutional</u>
Conv2D layer 1: 0.8
Conv2D layer 2: 0.8
Dense layer 1: 0.8
Dense layer 2: 0.6

<u>Concatenate</u>
Hidden layer 1: 0.8
Hidden layer 2: 0.6</td>
<td><u>Training loss</u>
427.87

<u>Validation loss</u>
427.61</td>
<td>
</td>
</tr>
<tr>
<td><u>Tabular</u>
Dense layer 1: 0.8
Dense layer 2: 0.8

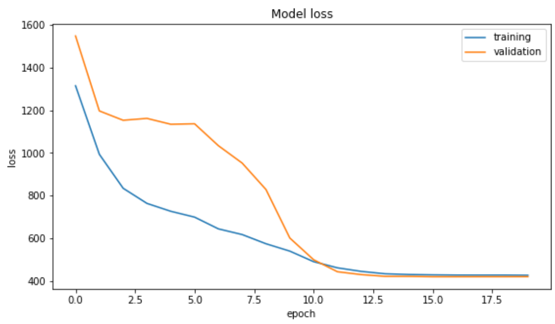
<u>Convolutional</u>
Conv2D layer 1: 0.6
Conv2D layer 2: 0.8
Dense layer 1: 0.8
Dense layer 2: 0.8

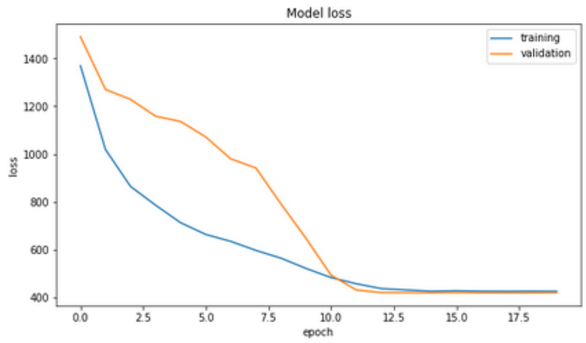
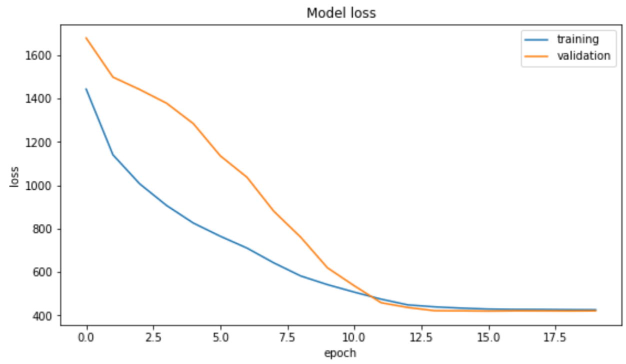
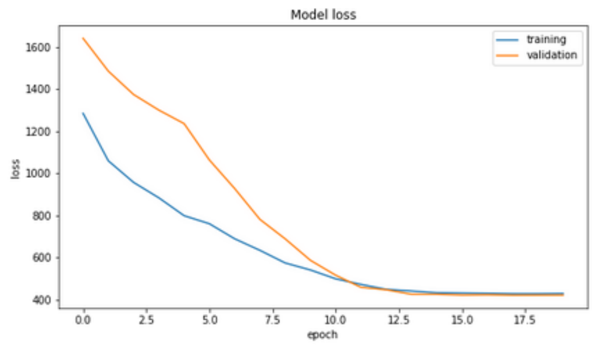
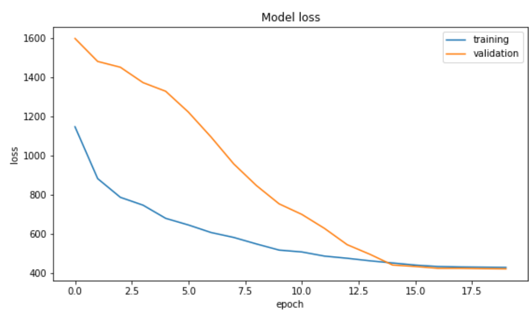
<u>Concatenate</u>
Hidden layer 1: 0.8
Hidden layer 2: 0.8</td>
<td><u>Training loss</u>
427.14

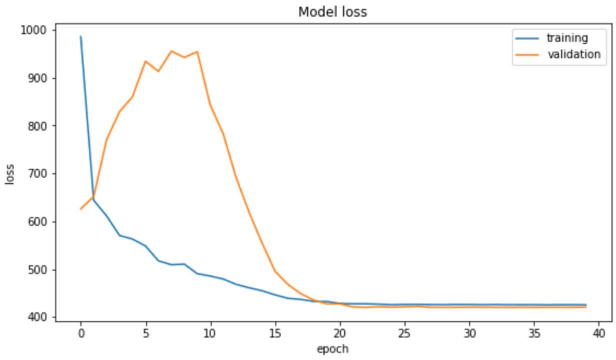
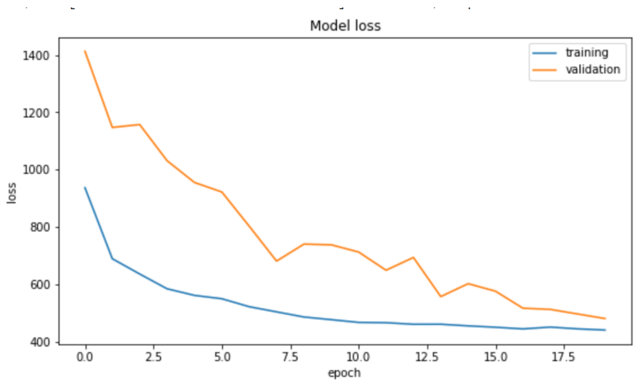
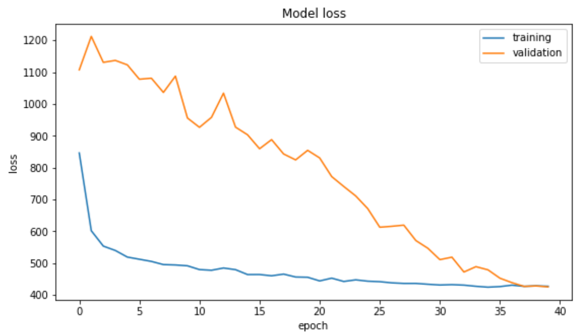
<u>Validation loss</u>
420.22</td>
<td>
</td>
</tr>
<tr>
<td><u>Tabular</u>
Dense layer 1: 0.8
Dense layer 2: 0.8

<u>Convolutional</u>
Conv2D layer 1: 0.8
Conv2D layer 2: 0.6
Dense layer 1: 0.8
Dense layer 2: 0.8

<u>Concatenate</u>
Hidden layer 1: 0.8
Hidden layer 2: 0.8</td>
<td><u>Training loss</u>
426.98

<u>Validation loss</u>
420.41</td>
<td>
</td>
</tr>
</table>

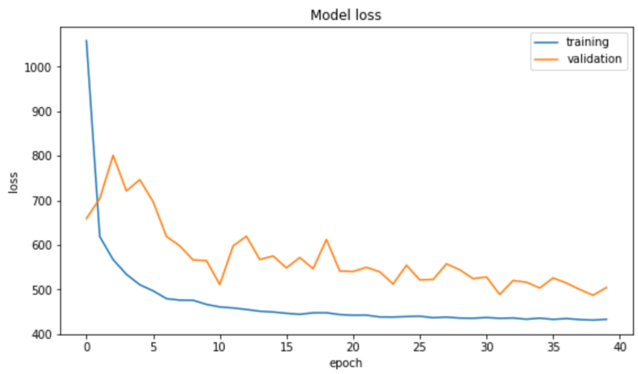
Tabular Dense layer 1: 0.8 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.6 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training loss 426.49 Validation loss 420.75	
Tabular Dense layer 1: 0.6 Dense layer 2: 0.8 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training loss 426.33 Validation loss 421.38	
Tabular Dense layer 1: 0.8 Dense layer 2: 0.6 Convolutional Conv2D layer 1: 0.8 Conv2D layer 2: 0.8 Dense layer 1: 0.8 Dense layer 2: 0.8 Concatenate Hidden layer 1: 0.8 Hidden layer 2: 0.8	Training loss 428.17 Validation loss 420.38	
Tabular Dense layer 1: 0.7 Dense layer 2: 0.7 Convolutional Conv2D layer 1: 0.7 Conv2D layer 2: 0.7 Dense layer 1: 0.7 Dense layer 2: 0.7	Training loss 427.55 Validation loss 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 loss</u> 425.39 <u>Validation loss</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 loss</u> 440.42 <u>Validation loss</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 loss</u> 427.62 <u>Validation loss</u> 425.83	

<table>
<tr>
<td><u>Tabular</u>
Dense layer 1: 0.5
Dense layer 2: 0.4

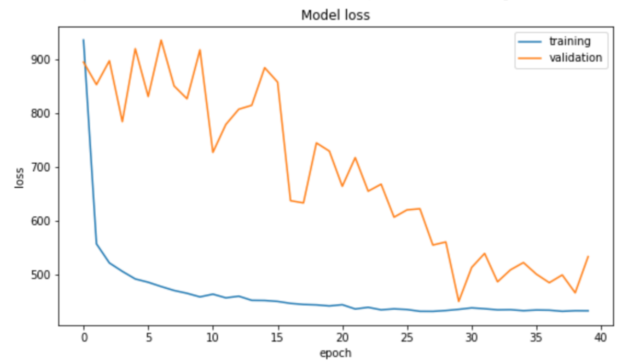
<u>Convolutional</u>
Conv2D layer 1: 0.5
Conv2D layer 2: 0.5
Dense layer 1: 0.4
Dense layer 2: 0.4

<u>Concatenate</u>
Hidden layer 1: 0.3
Hidden layer 2: 0.3</td>
<td><u>Training loss</u>
433.03

<u>Validation loss</u>
504.42</td>
<td>
</td>
</tr>
<tr>
<td><u>Tabular</u>
Dense layer 1: 0.4
Dense layer 2: 0.4

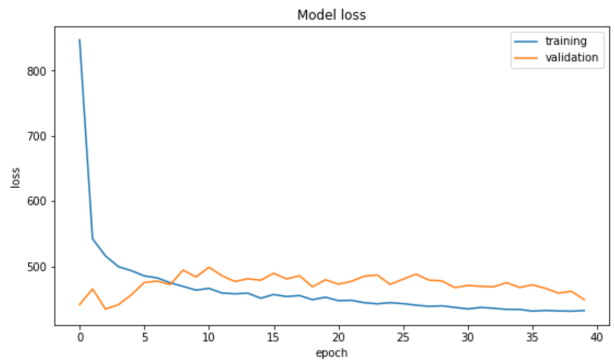
<u>Convolutional</u>
Conv2D layer 1: 0.4
Conv2D layer 2: 0.4
Dense layer 1: 0.4
Dense layer 2: 0.4

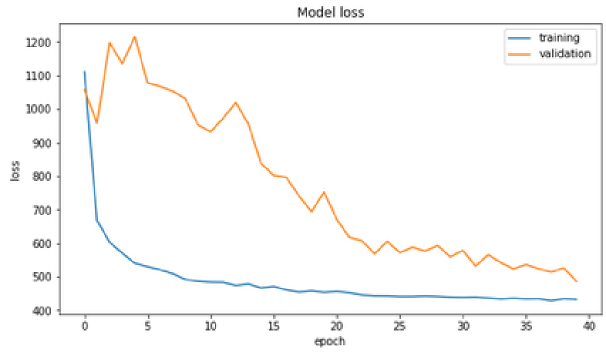
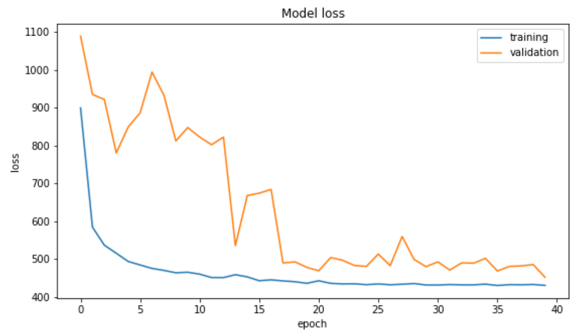
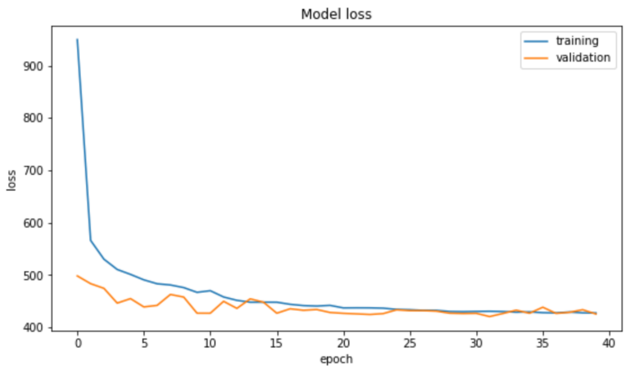
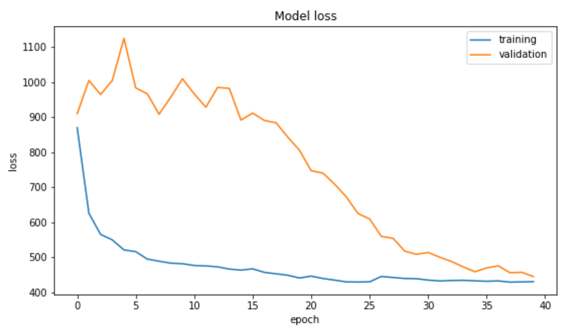
<u>Concatenate</u>
Hidden layer 1: 0.2
Hidden layer 2: 0.2</td>
<td><u>Training loss</u>
433.20

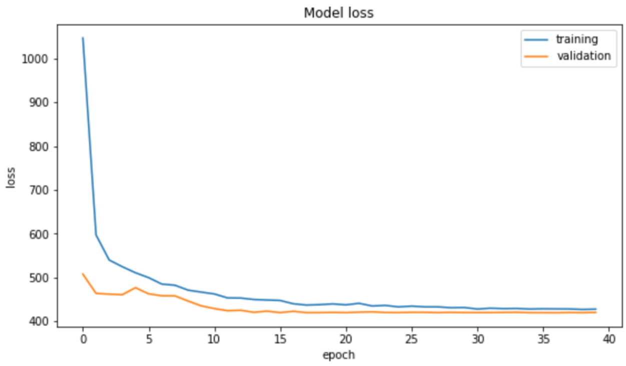
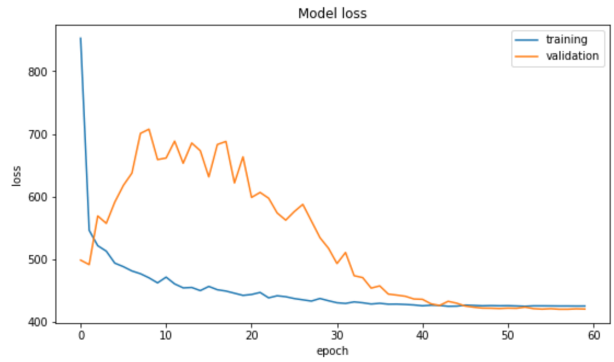
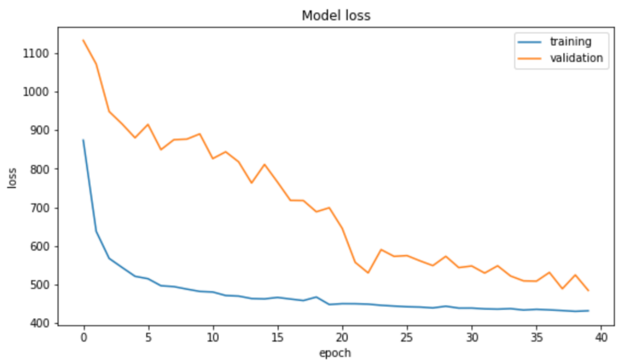
<u>Validation loss</u>
533.93</td>
<td>
</td>
</tr>
<tr>
<td><u>Tabular</u>
Dense layer 1: 0.4
Dense layer 2: 0.4

<u>Convolutional</u>
Conv2D layer 1: 0.4
Conv2D layer 2: 0.4
Dense layer 1: 0.4
Dense layer 2: 0.4

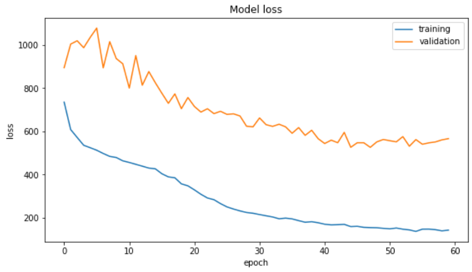
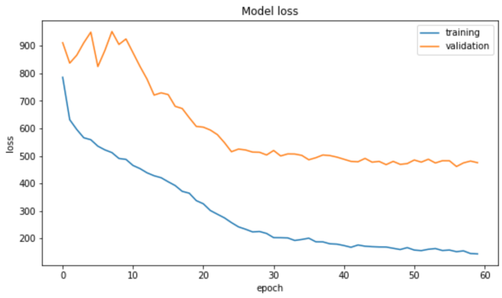
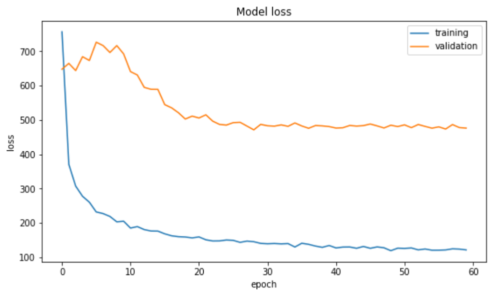
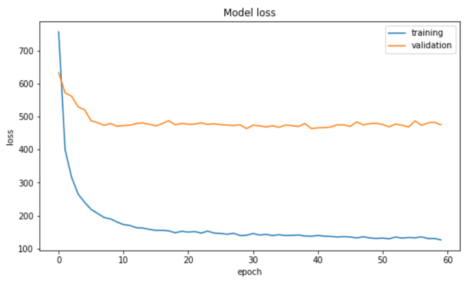
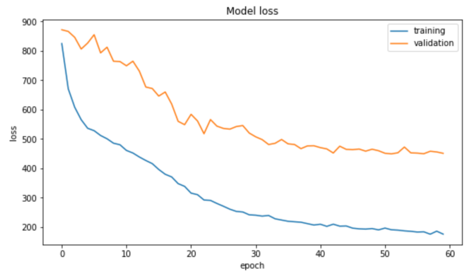
<u>Concatenate</u>
Hidden layer 1: 0.3
Hidden layer 2: 0.3</td>
<td><u>Training loss</u>
432.48

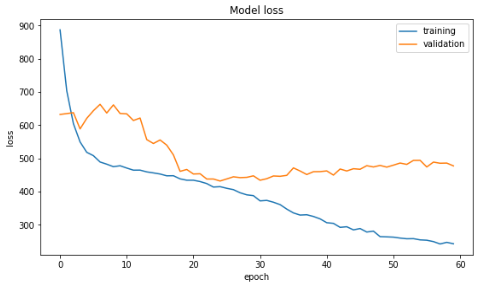
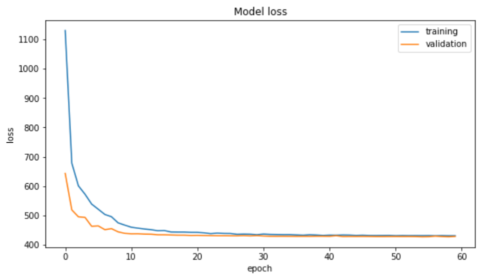
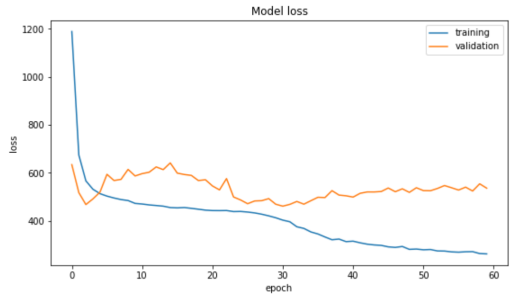
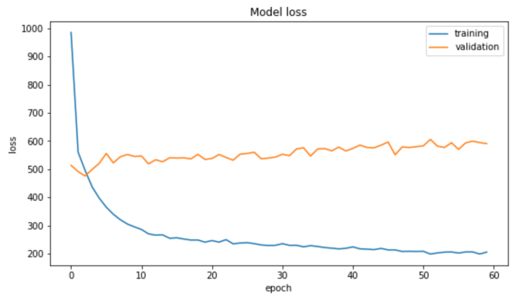
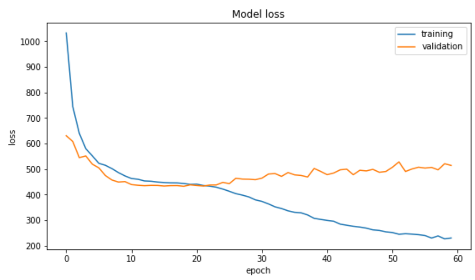
<u>Validation loss</u>
449.33</td>
<td>
</td>
</tr>
</table>

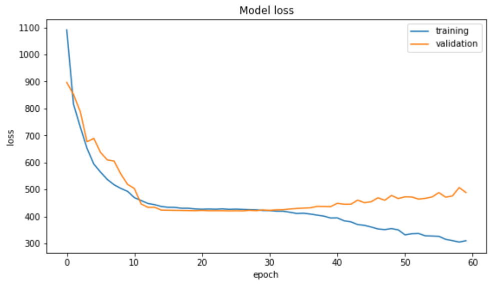
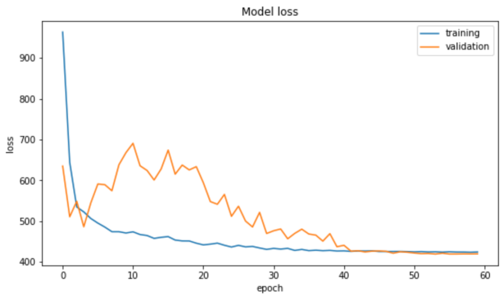
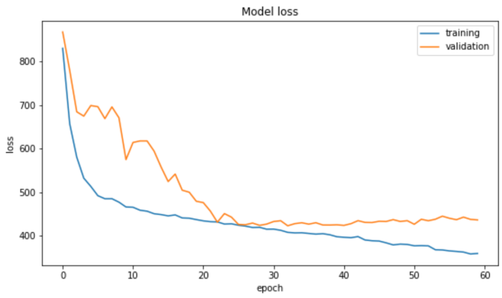
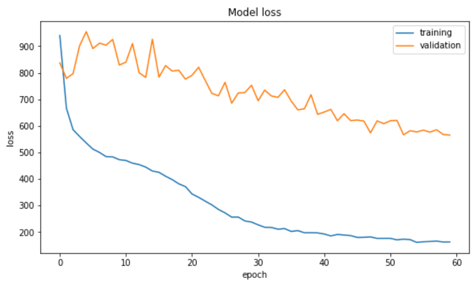
<u>Tabular</u> Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Convolutional</u> Conv2D layer 1: 0.5 Conv2D layer 2: 0.5 Dense layer 1: 0.5 Dense layer 2: 0.5 <u>Concatenate</u> Hidden layer 1: 0.4 Hidden layer 2: 0.4	<u>Training loss</u> 432.71 <u>Validation loss</u> 486.46	
<u>Tabular</u> Dense layer 1: 0.5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

Concatenate Hidden layer 1: 0.4 Hidden layer 2: 0.4		
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: 0.3 Dense layer 2: 0.3 Concatenate Hidden layer 1: 0.4 Hidden layer 2: 0.4	Training loss 427.99 Validation loss 420.66	
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: 0.3 Dense layer 2: 0.3 Concatenate Hidden layer 1: 0.4 Hidden layer 2: 0.4 Epochs 60	Training loss 425.17 Validation loss 420.33	
Tabular Dense layer 1: 0.3 Dense layer 2: 0.3 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: 0.4 Hidden layer 2: 0.4	Training loss 432.06 Validation loss 484.75	

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

Regularization	**Model loss (MSE)**	**Model loss graph**
<u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-4) Bias regularizer: L2 (ƛ=1e-4) Activity regularizer: L2 (ƛ=1e-5)	<u>Training loss</u> 142.23 <u>Validation loss</u> 566.46	
<u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-3) Bias regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3)	<u>Training loss</u> 143.76 <u>Validation loss</u> 475.27	
<u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-2) Bias regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2)	<u>Training loss</u> 121.60 <u>Validation loss</u> 476.53	
<u>Concatenated layers</u> Kernel regularizer: L2 (ƛ=1e-1) Bias regularizer: L2 (ƛ=1e-1) Activity regularizer: L2 (ƛ=1e-1)	<u>Training loss</u> 126.76 <u>Validation loss</u> 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)	<u>Training loss</u> 175.96 <u>Validation loss</u> 451.23	

Activity regularizer: L2 (λ=1e-3)		
<u>Dense layers CNN</u> Kernel regularizer: L2 (λ=1e-2) Bias regularizer: L2 (λ=1e-2) Activity regularizer: L2 (λ=1e-2) <u>Concatenated layers</u> Kernel regularizer: L2 (λ=1e-2) Bias regularizer: L2 (λ=1e-2) Activity regularizer: L2 (λ=1e-2)	<u>Training loss</u> 242.83 <u>Validation loss</u> 477.41	
<u>Dense layers CNN</u> Kernel regularizer: L2 (λ=1e-1) Bias regularizer: L2 (λ=1e-1) Activity regularizer: L2 (λ=1e-1) <u>Concatenated layers</u> Kernel regularizer: L2 (λ=1e-1) Bias regularizer: L2 (λ=1e-1) Activity regularizer: L2 (λ=1e-1)	<u>Training loss</u> 430.36 <u>Validation loss</u> 427.87	
<u>Dense layers CNN</u> Kernel regularizer: L2 (λ=1e-1) Bias regularizer: L2 (λ=1e-1) Activity regularizer: L2 (λ=1e-1) <u>Concatenated layers</u> Kernel regularizer: L2 (λ=1e-3) Bias regularizer: L2 (λ=1e-3) Activity regularizer: L2 (λ=1e-3)	<u>Training loss</u> 262.07 <u>Validation loss</u> 536.07	
<u>Dense layers CNN</u> Kernel regularizer: L2 (λ=1e-1) Bias regularizer: L2 (λ=1e-1) Activity regularizer: L2 (λ=1e-1) <u>Concatenated layers</u> Kernel regularizer: L2 (λ=1e-2) Bias regularizer: L2 (λ=1e-2) Activity regularizer: L2 (λ=1e-2)	<u>Training loss</u> 205.78 <u>Validation loss</u> 590.64	
<u>Dense layers CNN</u> Kernel regularizer: L2 (λ=1e-3) Bias regularizer: L2 (λ=1e-3) Activity regularizer: L2 (λ=1e-3) <u>Concatenated layers</u> Kernel regularizer: L2 (λ=1e-1) Bias regularizer: L2 (λ=1e-1) Activity regularizer: L2 (λ=1e-1)	<u>Training loss</u> 230.49 <u>Validation loss</u> 514.46	

Dense layers CNN Kernel regularizer: L2 (ƛ=1e-2) Bias regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2) Concatenated layers Kernel regularizer: L2 (ƛ=1e-2) Bias regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2) Dropout Concatenated layers: 0.6	Training loss 310.57 Validation loss 488.90	
Dense layers CNN Kernel regularizer: L2 (ƛ=1e-1) Activity regularizer: L2 (ƛ=1e-1) Concatenated layers Kernel regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3)	Training loss 424.51 Validation loss 419.86	
Dense layers CNN Kernel regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2) Concatenated layers Kernel regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2)	Training loss 359.52 Validation loss 436.45	
Dense layers CNN Kernel regularizer: L2 (ƛ=1e-2) Concatenated layers Kernel regularizer: L2 (ƛ=1e-2)	Training loss 162.95 Validation loss 565.34	
Dense layers CNN Kernel regularizer: L2 (ƛ=1e-2) Activity regularizer: L2 (ƛ=1e-2) Concatenated layers Kernel regularizer: L2 (ƛ=1e-3) Activity regularizer: L2 (ƛ=1e-3)	Training loss 220.07 Validation loss 479.80