Report of Iterative Development

Machine Learning Project

Brian Jakobs, Vera Eising, Joram Brokkelkamp, Damla Baspinar

Minor AI

Teachers: Tim Doolan, Wouter Vrielink
Supervisor: Kelly Spaans

Date: the 3rd of February, 2022

Contents

1. Milestone I: First Model

1.1 Introduction

Sheltered cats and dogs display specific characteristics which can be considered determinants of adoption[1]. Since animals that are not adopted can be subject to suffering in different forms[2], it seems logical for shelters to attempt to focus their resources on determinants with an increasing effect on the likelihood that a given animal will be adopted[1]. Petfinder.my, a Malaysian platform regarding animal welfare, operates using a similar concept[3]. With a cuteness meter the organization assigned a *'Pawpularity'* score to submitted pictures of cats and dogs, which is calculated using the statistics of animals' profile pages on the website . A higher score correlates to a more attractive picture. This score can be used to improve photo-quality in order to optimize the chances of adoption. The current algorithm is still under development and could be improved[4]. The model described in this report seeks to provide an improved version of this algorithm. The aim of this report is to elucidate the process of formation of said model.

The first version of the model in question is a convolutional neural network which takes images of animals and designated attractiveness scores as inputs and performs a number of computations to estimate the score of attractiveness for another given image.The rationale behind this approach is that deep learning is known to be successful in tasks that require image recognition[5]. The goal regarding the performance of this model is to achieve a model that shows a decrease in its loss over a number of epochs. This shows the designed model has the capacity to be trained.

1.2 Data Analysis and Preprocessing

The data provided for every sample is composed of the images of a pet and additional tabular data regarding every given image. The tabular data states whether the following composition parameters were present or absent in a given image:

- Focus (Pet stands out against 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 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)".[6]

Additionally, the tabular data also includes the "*Pawpularity*", which reflects each pet profile's page view statistics[6]. Furthermore, this data set only contains 9912 samples, which is a relatively small amount for training.

The tabular data was analyzed using the *Normal Equation*. It was established that the relation between the features and the outcome "*Pawpularity*" was not linear. Furthermore, it was established that the tabular data does not contain any missing values.

The data used for this version of the model comprises two parts: the "*Pawpularity*" within the tabular data and the images of pets. Figure 1 shows that the used data set is imbalanced, as most images have a "*Pawpularity*" score of approximately 30 with a peak around 100. The mean "*Pawpularity*" score is 38.

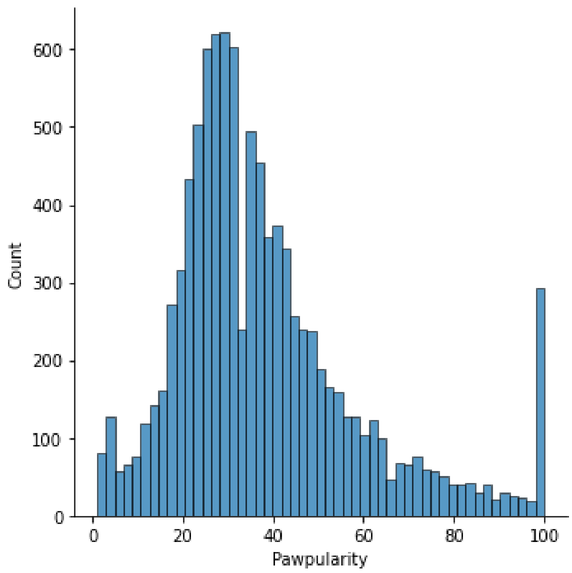


Figure 1: This figure is a histogram that presents the number of images for every "Pawpularity" score. It shows the data set is imbalanced.

Regarding the images, every image was originally resized to a shape of 128x128x3. All provided testing data had this specific shape, whereas there the shapes of the samples in the training data differed. Therefore, it was decided to resize all training data to the same shape as testing data. However, after careful consideration it was established that resizing the images to a shape of 64x64x3 yielded similar results whilst saving computational expenses. For this reason, the images were resized to a shape of 64x64x3. Furthermore, the data was split into training and testing data, corresponding with 80% and 20% of all data respectively.

No further preprocessing or augmentation was applied to the images.

1.3 Model Pipeline and Training

The current predictive model is a convolutional neural network. The input of a given sample is the image of a pet, whereas the output is the "*Pawpularity*" score.
The architecture of the convolutional neural network consists of an input layer, three hidden layers and an output layer.
With the exception of the last layer, a ReLu activation function was used in every layer. The ReLu activation function was chosen for the hidden layers of the predictive model to avoid problems such as vanishing gradient during backpropagation in training[9]. The ReLu function was chosen as it is non-linear around the coordinate (0,0) and non-linearity is a necessity for a model to learn complex decision boundaries[10].
The output layer provides one output. No activation function was applied in this layer. Given that the task at hand is a regression problem, a non-linear activation function should not be necessary in the last layer.
Further details regarding the model pipeline are presented in Figure 2.

Model: Mile stone I

Layer (type)	Output Shape	Params
Convolution (2D)	(128, 128, 3)	1792
Max pooling (2D)	(64, 64, 64)	0
Flatten	(262144)	0
Dense 1	(512)	134218240
Dropout (0.1)	(512)	0
Dense 2	(256)	131328
Dense 3	(128)	32869
Dense 4	(1)	129

Total params: 134,384,385
Trainable params: 134,384,385
Non-trainable params: 0

Figure 2: Details regarding the model pipeline

Concerning the training of the model, the Mean Squared Error was used as the loss function for optimization. Additionally, the Root Mean Squared Error was used as a metric to judge the performance of the model, since this is customary in regression problems such as the one at hand. The optimizer Adam was applied. The model was trained using 20 epochs.

1.4 Evaluation and Conclusions

Regarding the model loss and the root mean squared error, there is a clear decrease for the training data (Figure 3). Although this decrease seems to stagnate after the first epoch in Figure 3, it is actually consistent over the course of 20 epochs. The loss for the validation loss also consistently decreases through 20 epochs.
The decreases in both the model loss as well as the root mean squared error are a manifestation of the trainability of the model, thus the goal of forming a model with the capacity to be trained was obtained.

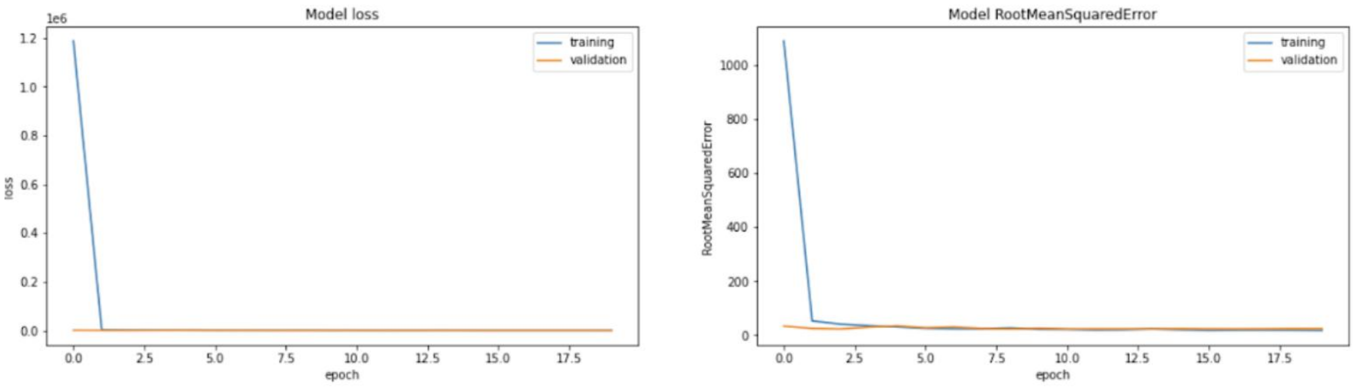


Figure 3: This figure shows the graphs of convergence of the model over 20 epochs.

Figure 3 shows that the model loss and root mean squared error were relatively high after the first epoch. Because of this, the relatively small nuances between the training and the validation values are not visible in the learning curves. In order to clarify this image, the y-axis for the model loss was limited to a value of 10000 and the y-axis for the root mean squared error was limited to 100 for the model loss and root mean squared error respectively. The outcome is illustrated in Figure 4.

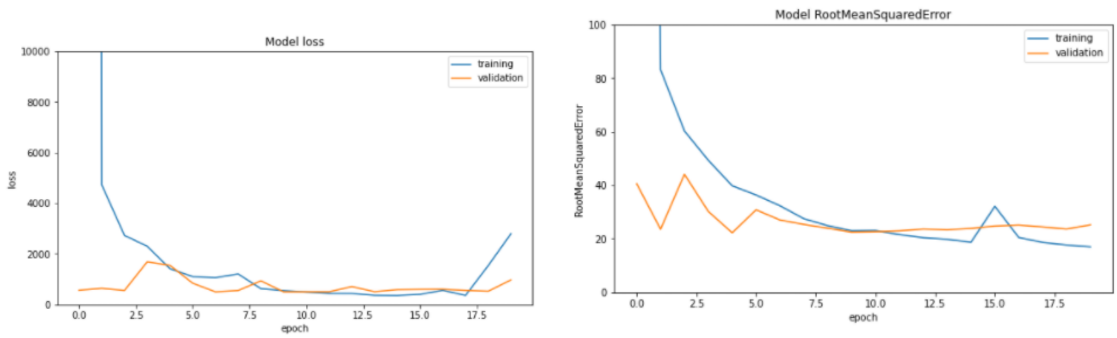


Figure 4: This figure shows the graphs of convergence of the model over 20 epochs with limited y-axes.

In Figure 4, the graph of the model loss shows that after approximately 17 epochs, the model loss and root mean square error both start to increase for the training data. This is a manifestation of divergence rather than convergence in gradient descent. This could be caused by a learning rate that is too high.

Given the small amount of available data, the application of k-fold cross validation and data augmentation could improve the training process of the model in future versions. Additionally, the use of preprocessing and augmentation methods in the images could derive better results. Furthermore, the increasing difference between the training loss and the validation loss during the last epochs seems to imply overfitting. Lastly, the problem of divergence should be solved.

2. Milestone II: Early stopping

2.1 Introduction

In the previous version of the predictive model, the learning curves appeared to be quite similar for the training and validation data up until epoch 16. After this epoch, the model loss steeply moves upwards (see Figure 4). This problem will be addressed in this version of the model by applying the method of early stopping. Early stopping is traditionally used to prevent overfitting[7]. It seems that it could be a way to improve the current learning curves.

2.2 Data Analysis and Preprocessing

No adaptations regarding data analysis or preprocessing were made in the current version of the model. See section 1.2 for a description of the current situation.

2.3 Model Pipeline and Training

In this version of the model, only 16 epochs were used to train the model. No further improvements were made with regard to the model pipeline.

2.4 Evaluation and Conclusions

After implementing early stopping, the model loss no longer shows a lift towards the last epochs. Both the loss as well as the root mean squared error now seem to be on a similar trend downwards. 16 epochs seems to be the optimal number of epochs for the model in its current state.

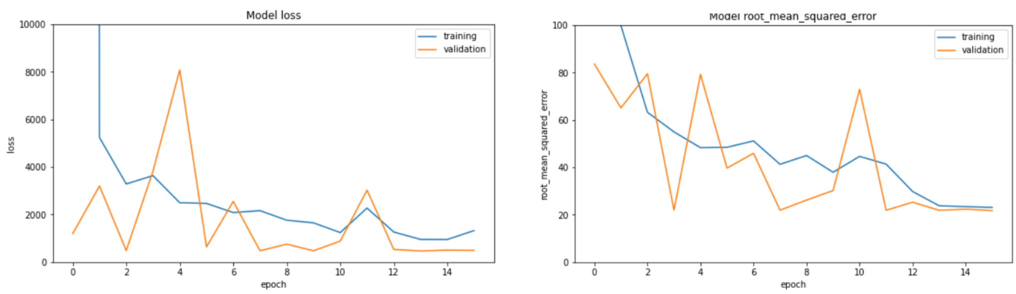


Figure 5: This figure shows the graphs of convergence of the model over 16 epochs with limited y-axes.

The learning curves both seem to show a stagnation at the last epochs rather than a decrease in the model loss or the model root mean squared error. It seems as if the model lacks complexity to fit the data better.

3. Milestone II: A Deeper Convolutional Neural Network

3.1 Introduction

The previous version of the convolutional neural network contained a single Conv2D layer. Adding more of such layers should correspond to adding complexity to the predictive model. This could enable the predictive model to learn more high level features, which illustrates the rationale for making a deeper neural network for the current version of the model. This might enhance the performance of the model, which should manifest itself in a decreased loss and mean squared error in the learning curves at the last epochs, compared to the previous version of the model.

3.2 Data Analysis and Preprocessing

In this version of the model, no modifications were made concerning data analysis or preprocessing. See section 1.2 for the most recent updates regarding this matter.

3.3 Model Pipeline and Training

The convolutional model was expanded with three additional Conv2D layers. Numbers of four, five and six additional Conv2D layers were tested as well. However, as the model seemed to behave very similarly after adding these layers, a number of three additional Conv2D layers was chosen for the optimal model at this stage of development. Figure 6 provides additional details regarding this optimal model pipeline.

Model: Mile stone II

Layer (type)	Output Shape	Params
Convolution 1 (2D)	(128, 128, 3)	1792
Max pooling 1 (2D)	(64, 64, 64)	0
Convolution 2 (2D)	(64, 64, 128)	73856
Max pooling 2 (2D)	(32, 32, 128)	0
Convolution 3 (2D)	(32, 32, 256)	295168
Max pooling 3 (2D)	(16, 16, 256)	0
Convolution 4 (2D)	(16, 16, 512)	1180160
Max pooling 4 (2D)	(8, 8, 512)	0
Flatten	(32768)	0
Dense 1	(512)	16777728
Dropout (0.1)	(512)	0
Dense 2	(256)	131328
Dense 3	(128)	32869
Dense 4	(1)	129

Total params: 18,493,057
Trainable params: 18,493,057
Non-trainable params: 0

Figure 6: Details regarding the model pipeline

3.4 Evaluation and Conclusions

In this version of the model, the training loss around the last epoch is approximately 200. In the earlier version this value used to be approximately 2000. There is a clear decrease in model loss between the respective models, thus the model seems to be trained better in its more complex form. The trade-off is that the model now appears to be overfitting, given that the training loss still seems to follow a decreasing trend. On the other hand, the validation loss has stagnated, starting at approximately 6 epochs.

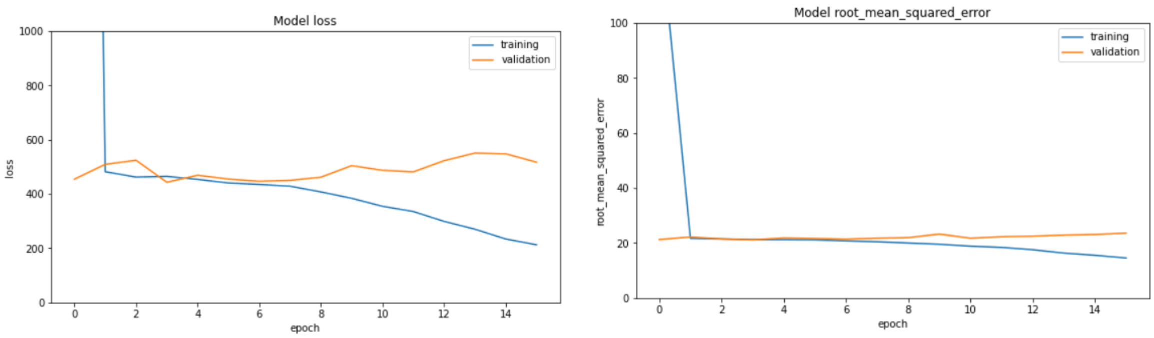


Figure 7: This figure shows the graphs of convergence of the model over 16 epochs with limited y-axes.

4. Milestone II: Normalization

4.1 Introduction

Given that the ReLu activation function was used, the input of the model should be normalized to have a mean of zero and standard deviation of one{10]. In this version of the model, normalization will be applied to the input samples. More efficient training should expectedly manifest itself in earlier conversion to a local optimum during gradient descent, by showing that learning curves stagnate earlier compared to previous versions of the model.

4.2 Data Analysis and Preprocessing

Referring back to section 1.2, the only update with regard to the data is the application of normalization to the image features. The pixels were normalized to have a mean value of zero and a standard deviation of one. In this way, the pixels of every sample (i.e. image) were normalized towards zero.

4.3 Model Pipeline and Training

In this version, the model pipeline remains unchanged to the pipeline described in section 3.3.

4.4 Evaluation and Conclusions

Normalization was applied under the assumption that it would make training more efficient. In the prior version of the model, the learning curves stagnated after approximately 8 epochs. In this version, this event occurs after approximately 1 epoch (Figure 8). It seems that the training process of the model has indeed become more efficient.

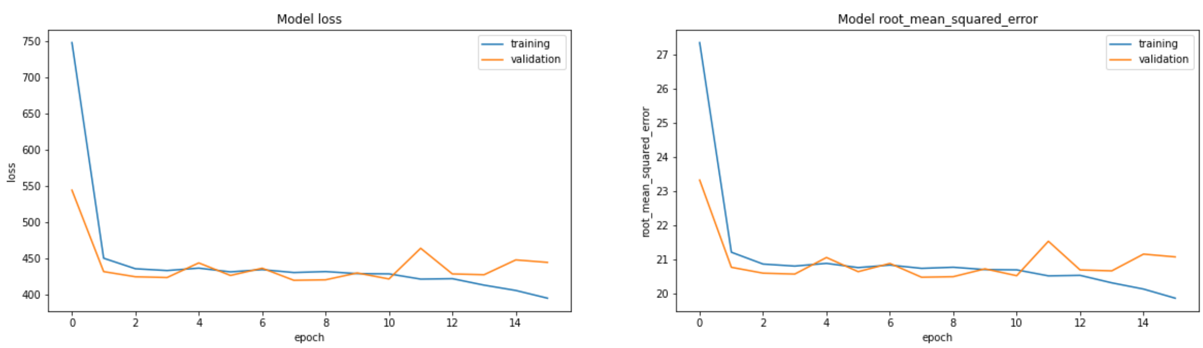


Figure 8: This figure illustrates the model loss and model root mean squared error after the input of the model was normalized.

5. Milestone II: Batch Normalization

5.1 Introduction

In the previous versions of the model the input nodes of the model were normalized to have a mean and standard deviation of zero and one respectively. Considering the depth of the convolutional neural network. it seems appropriate to additionally normalize the values of all hidden layers. This will be done by applying batch normalization[10]. Batch normalization reduces the problem of input values changing in between layers and makes the training process of a neural network more efficient[10].

5.2 Data Analysis and Preprocessing

No further improvements regarding data analysis or preprocessing were made in this version of the model.

5.3 Model Pipeline and Training

In the current version of the model, a batch normalization layer was added after every maxpool layer consecutively. This resulted in three different models which were compared to select the optimal model regarding batch normalization. For details regarding the model pipeline see Figure 9.

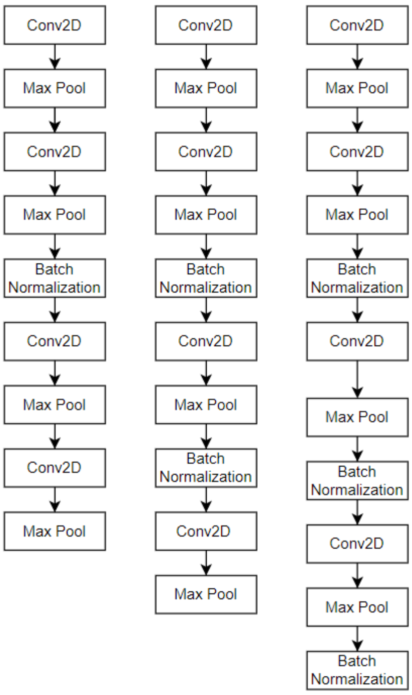
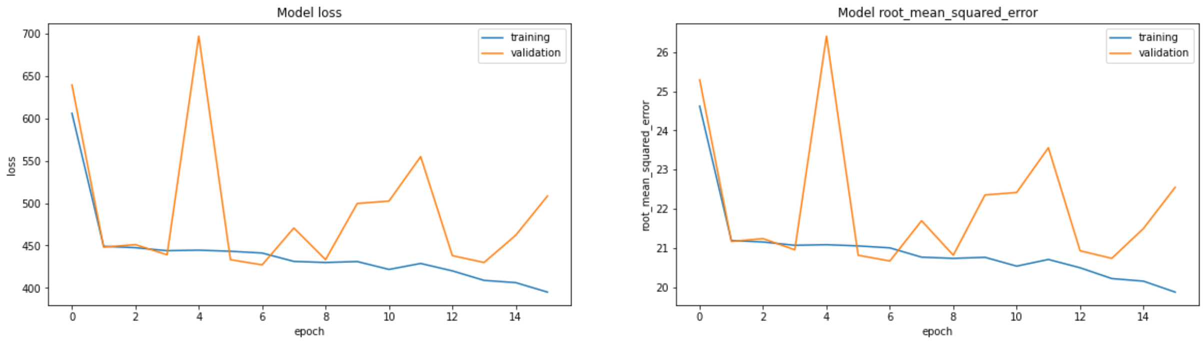


Figure 9:This figure shows the part of the model pipeline that was used for experiments in this version of the model, which is everything up until the flatten layer. A batch normalization layer was added after every experiment.

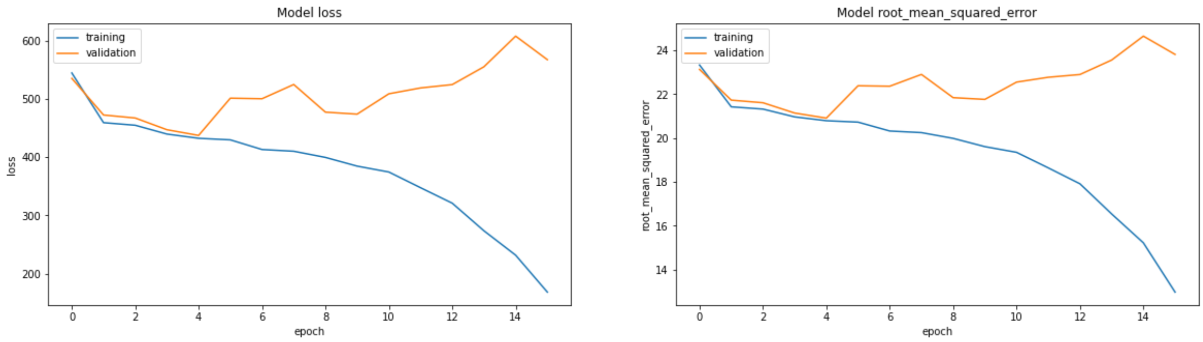
5.4 Evaluation and Conclusions

Figure 10 shows that the course of the validation cost is most smooth after the application of two batch normalization layers, compared to both prior versions as well as versions with additional batch normalization. Additionally, overfitting has become more clear in this model, as the model loss continues to decrease for the training data, whereas it stagnates for the validation data, starting after approximately 4 epochs. The model with two batch normalization layers will be maintained for the development of future versions of the model.

A single layer of batch normalization



Two layers of batch normalization



Three layers of batch normalization

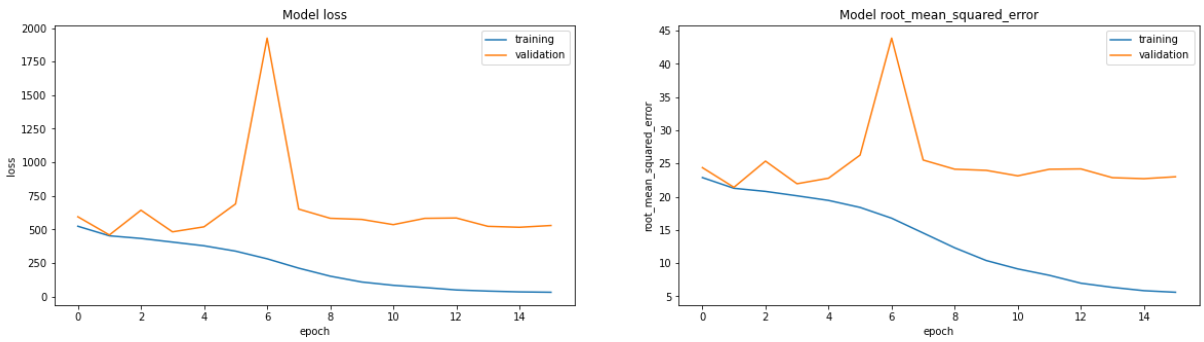


Figure 10: This figure illustrates the model loss and model root mean squared error after the consecutive application of batch normalization layers.

6. Milestone II: Dropout

6.1 Introduction

A problem regarding the earlier versions of the model is overfitting. In order to address this problem, dropout will be applied to the model. Dropout is a regularization method that prevents a neural network from relying too heavily on specific nodes by randomly selecting nodes and setting them to zero during training. Because of this, the neural network becomes less likely to rely on specific nodes or weights in training[11].

6.2 Data Analysis and Preprocessing

In this version of the model, no adjustments were applied in data analysis or preprocessing. See section 2.2 for the most recent modifications regarding this matter.

6.3 Model Pipeline and Training

Several experiments regarding dropout were arbitrarily conducted for this version of the model. The best version is presented here.
Firstly, before every dense layer, a dropout layer was added with a dropout rate of 0.4, which means approximately 40% of the inputs in these layers was randomly set to zero.
Secondly, dropout was added before the two last Conv2D layers, with a dropout rate of 0.2. This dropout layer has a relatively low dropout rate compared to the dropout layers before the dense layers in order to preserve the information regarding high-level image features earlier in the convolutional neural network.
Figure 11 presents a summary of the pipeline concerning the current model.

Model: Mile stone II, dropout

Layer (type)	Output Shape	Params
Convolution 1 (2D)	(128, 128, 3)	1792
Max pooling 1 (2D)	(64, 64, 64)	0
Convolution 2 (2D)	(64, 64, 128)	73856
Max pooling 2 (2D)	(32, 32, 128)	0
Convolution 3 (2D)	(32, 32, 256)	295168
Max pooling 3 (2D)	(16, 16, 256)	0
Dropout 1 (0.1)	(16, 16, 256)	0
Convolution 4 (2D)	(16, 16, 512)	1180160
Max pooling 4 (2D)	(8, 8, 512)	0
Flatten	(32768)	0
Dropout 2 (0.4)	(512)	0
Dense 1	(512)	16777728
Dropout 3 (0.4)	(512)	0
Dense 2	(256)	131328
Dropout 4 (0.4)	(512)	0
Dense 3	(128)	32869
Dropout 5 (0.4)	(512)	0
Dense 4	(1)	129

Total params: 18,493,057
Trainable params: 18,493,057
Non-trainable params: 0

Figure 11: Details regarding the model pipeline

6.4 Evaluation and Conclusions

Compared to prior versions of this convolutional neural network, the model shows a smaller difference between the training and validation in the loss as well as root mean squared error, as seen in Figure 12. However the model is still overfitting, even though the loss is no longer decreasing. The next step should be to explore alternative methods against overfitting.

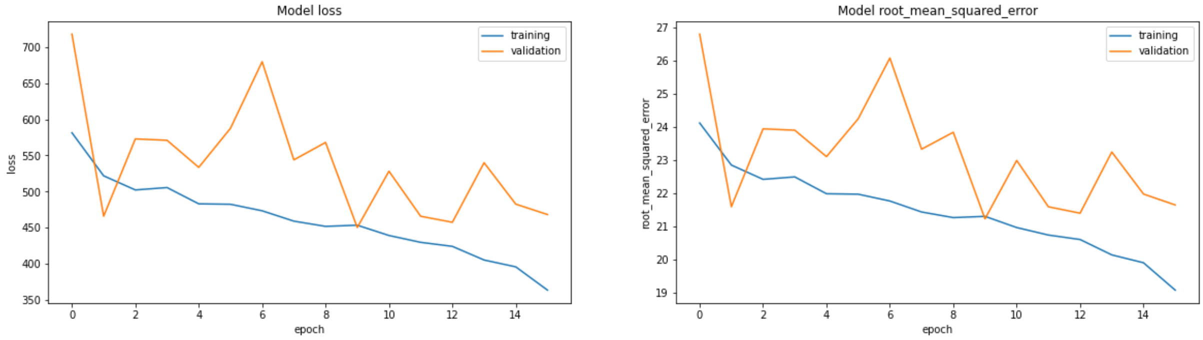


Figure 12: This figure illustrates the model loss and model root mean squared error after the application of dropout.

7. Milestone III: Additional Regularization techniques

7.1 Introduction

Given that the previous version of the model seemed to suffer from an overfitting problem that cannot be solved with dropout only, it makes sense to apply additional regularization methods at this stage of the model. For this reason, L2 regularization will be applied next[12]. In this chapter, the interaction between dropout and L2 regularization will be explored. L2 works by forcing weights to be smaller, but not 0, which should help against overfitting. It is common in machine learning to prefer L2 regularization over L1 regularization. Furthermore we also chose to use an activity regularization in favor of a bias or kernel regularization, since this regularization should work best against overfitting by reducing the layer's output[13].

7.2 Data Analysis and Preprocessing

No additional improvements regarding data analysis or processing were made in this version of the model.

7.3 Model Pipeline and Training

In this version, L2 and regularization were systematically applied at various places in the model. See Figure 13 for a schematic overview of the model that was eventually selected.

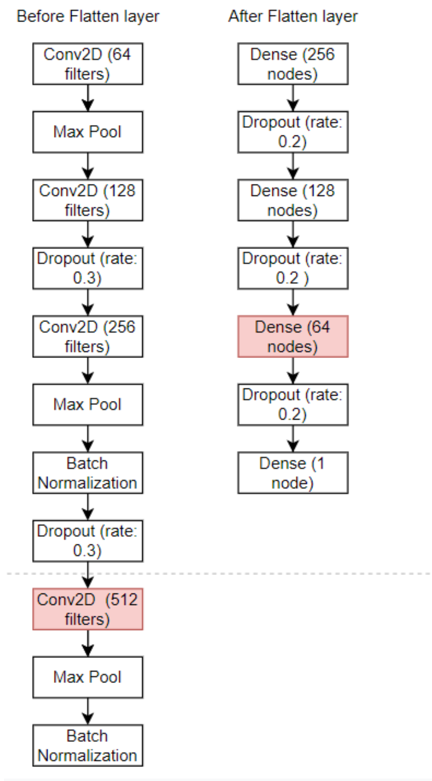


Figure 13: Details regarding the model pipeline. L2 regularization was applied in the layers highlighted in red.

7.4 Evaluation and Conclusions

Figure 14 shows that the validation root mean squared error does not improve beyond a value of approximately 20. The model seems to underfit the data as the validation loss becomes lower than the training loss. It seems that the performance of the model in its current form cannot be enhanced through regularization techniques. Perhaps artificially augmenting the data might improve the validation performance.

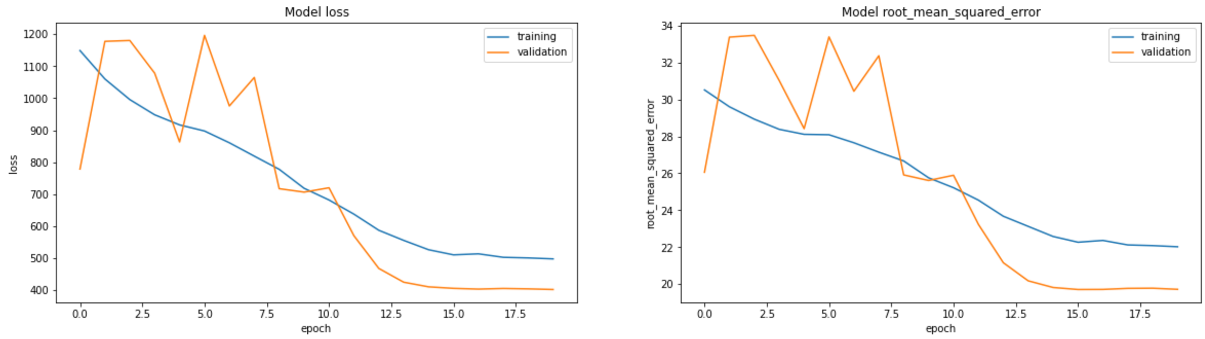


Figure 14: This figure illustrates the model loss and model root mean squared error after applying additional regularization.

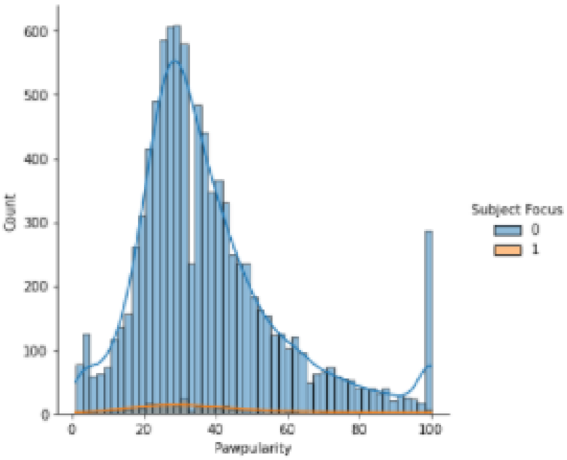
8. Milestone III: Tabular Data

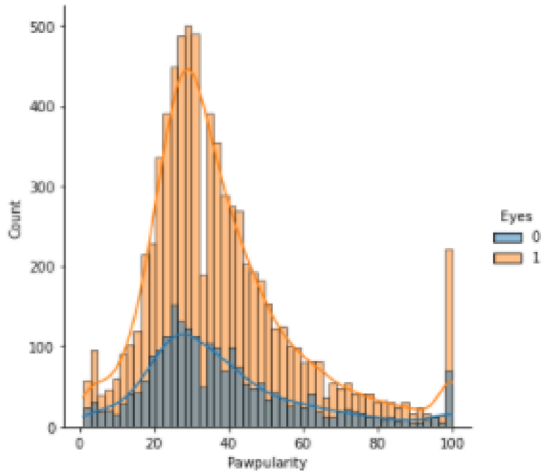
8.1 Introduction

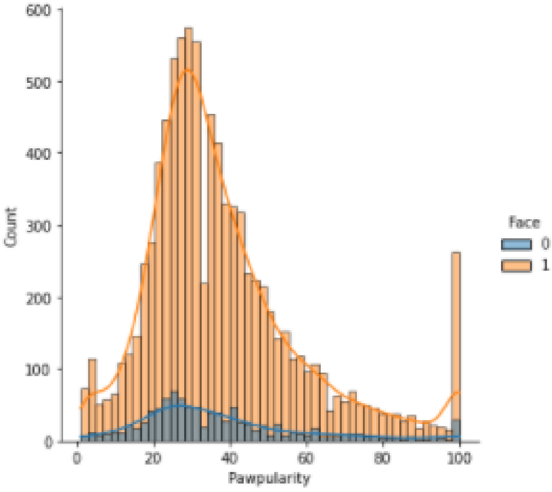
Though it has not been used in any version of the model thus far, with the exception of the "*Pawpularity*" score, a set of tabular data was provided in addition to the image data. Adding this data to the model is a potential way of enhancing the performance of the model. For this reason, chapter 8 describes the development of the prior model into a convolutional neural network with added tabular data.

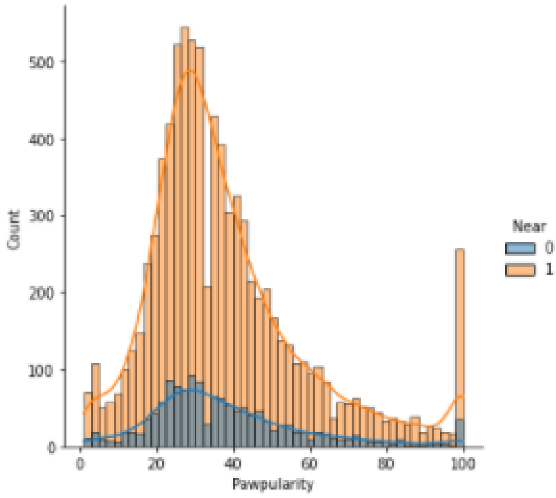
8.2 Data Analysis and Preprocessing

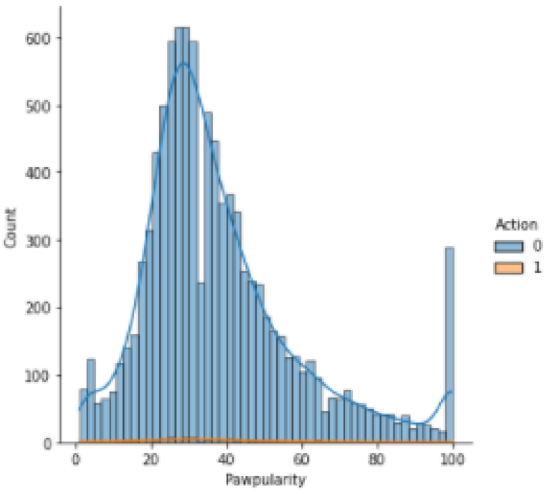
For this version of the model, all tabular data was taken into consideration, in addition to the image data which was included in previous versions of the model. An additional data analysis was conducted first to estimate the performance of the model using tabular data. The graphs in Figure 15 show a combined histogram and distribution plot for every feature regarding the images in the tabular data. The distribution plot specifically, shows that distribution of the presence or absence of features does not differ with respect to the '*Pawpularity*' score. This implies the absence or presence of a specific feature in itself is not sufficient to indicate a higher or lower score. For this reason, the addition of tabular data to the image data is unlikely to change the performance of the model with regard to its error.

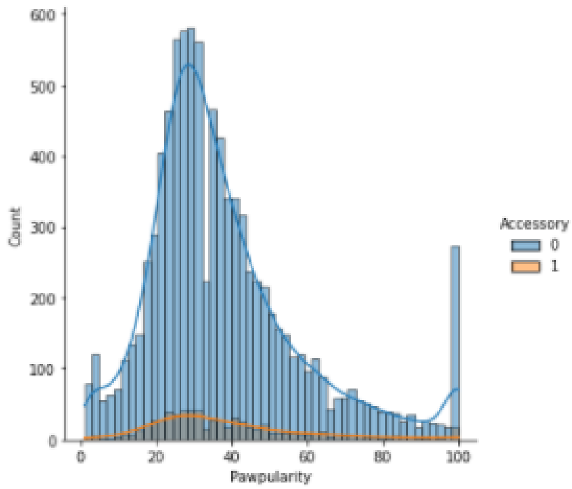

Count
Pawpularity
Subject Focus
0
1


Count
Pawpularity
Eyes
0
1


Count
Pawpularity
Face
0
1


Count
Pawpularity
Near
0
1


Count
Pawpularity
Action
0
1


Count
Pawpularity
Accessory
0
1

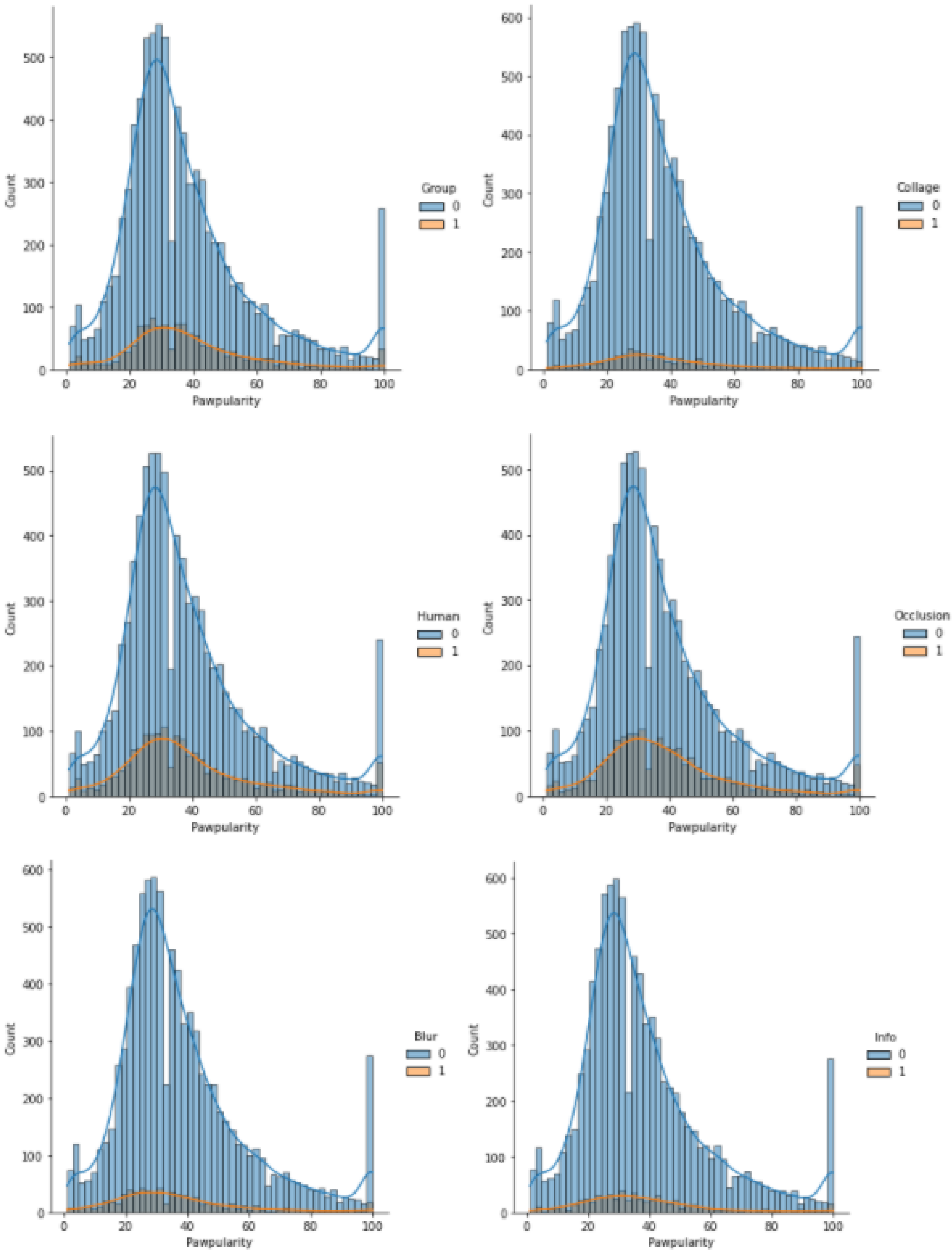


Figure 15: The combined histograms and distribution plots for each of the features in the tabular data.

8.3 Model Pipeline and Training

To add the tabular data, a second model was created and concatenated with the original model. We experimented with different amounts of layers to find the optimal model for the tabular data. First, we tried a single layer to just use the tabular data as it is. We also tried to make it more complex by adding more layers to see whether the performance of the model would improve.

8.3.1 Tabular Model 1

For tabular model 1, only the raw tabular data was used. We did not make the model more complex.

8.3.2 Tabular Model 2

For tabular model 2 we added 4 extra layers before we concatenated it to the main model. This will enable the model to learn more complex ideas for combinations with the tabular data. The nodes in the layers for this model evolve as follows (12, 30, 50, 30, 10). After, the models combine.

8.3.3 Tabular Model 3

For tabular model 3 the impact of more layers on the loss was explored. This means the data will be more complex. We implemented the following layers (12, 30, 90, 270, 810, 270, 90, 30, 12).

8.4 Evaluation and Conclusions

8.4.1 No tabular data

We can see that without tabular data the training curves of the model start stagnating around 15 epochs for both the training and validation.

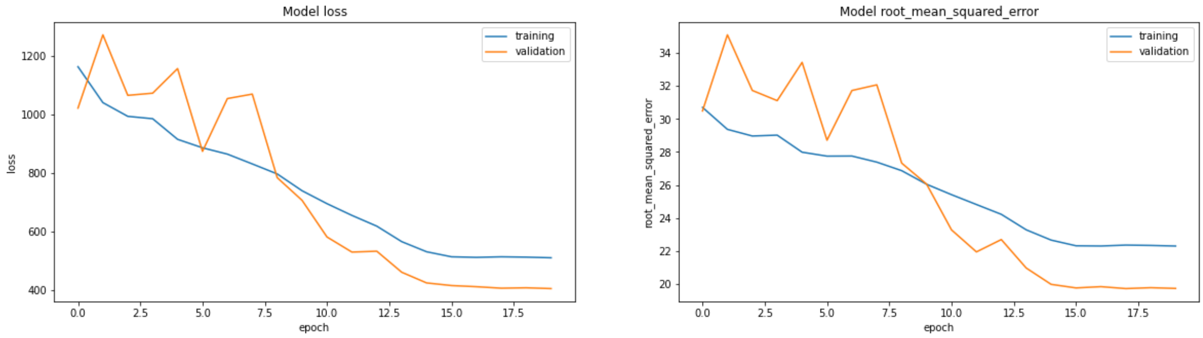


Figure 16: The model without tabular data

8.4.2 Tabular Model 1

When we added a single layer without any complexity, we observed that adding tabular data made the model less accurate compared to the model without any tabular data, though only by a small margin. It does seem like the data starts stagnating around its minimum faster.

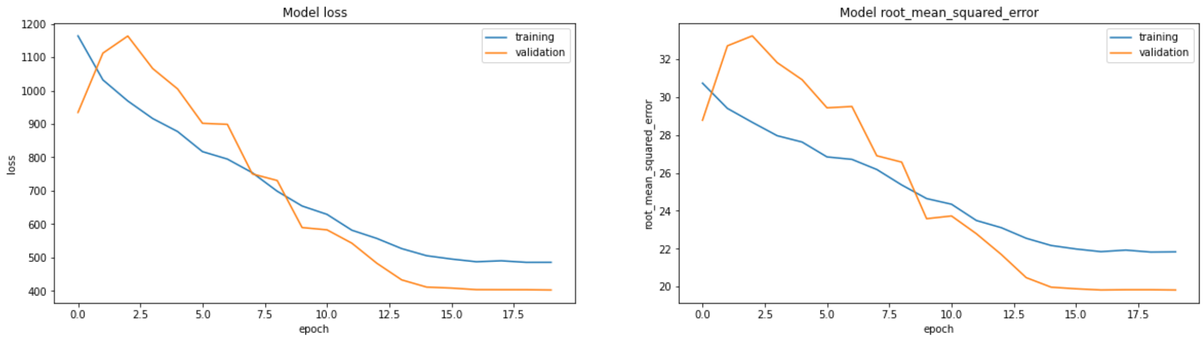


Figure 17 : The model with tabular data without added complexity

8.4.3 Tabular Model 2

After using 4 layers, the model learned a lot faster, as it already starts stagnating around its minimum after only 3 epochs. The validation costs are also lower than without using tabular data at all.

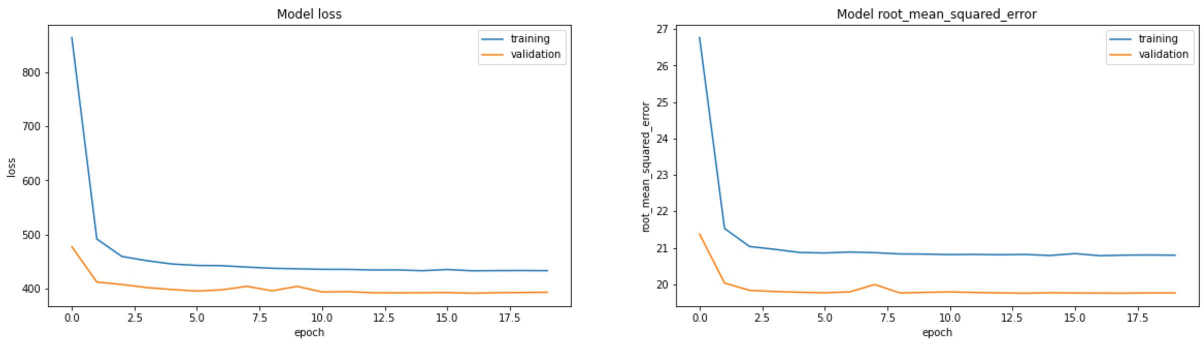


Figure 18: The model with tabular data with added complexity

8.4.4 Tabular Model 3

For model 3 we added more complexity to the model. Figure 18 shows that this increased the average root mean squared error made the model less consistent. The model has a higher root mean squared error and does not seem to be the best fit. The training curve does start stagnating at its minimum even faster as it is already close to its minimum after only 2 epochs.

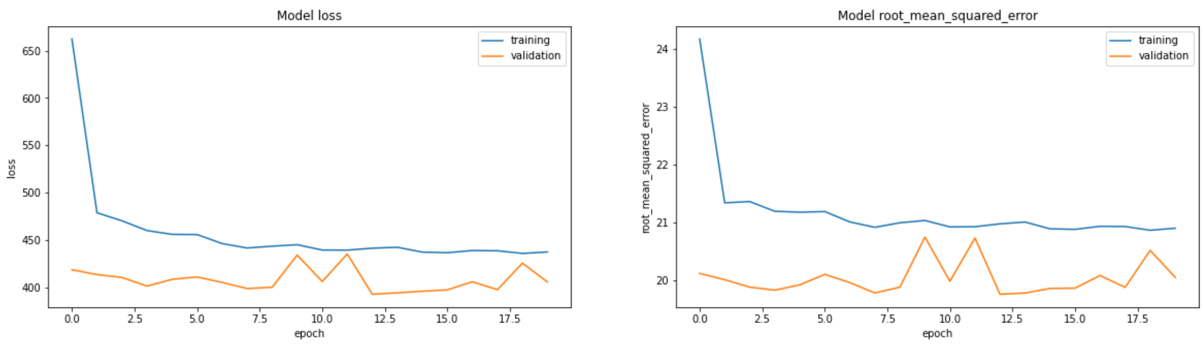


Figure 19: The model with tabular data with additional added complexity

From the figures above we can conclude that Model 2 seems to be the best model to continue using. It reaches the lowest root mean squared error value. Additionally, the course of the root mean squared error is the most consistent compared to not using the tabular data and using more and less complexity for the tabular data.

9. Milestone IV: Optimizers

9.1 Introduction

In all the previous models Adam was used as the optimizer. We were most familiar with the optimizer Adam and after researching online, it became evident Adam is seen as one of the best optimizers for convolutional neural networks[13]. Yet, we wanted to find out whether that was the case for this specific network too, as the network is quite small and also has tabular data.

9.2 Data Analysis and Preprocessing

No additional improvements regarding data analysis or processing were made in this version of the model.

9.3 Model Pipeline and Training

9.3.1 The SGD Optimizer

In all earlier versions of the model, the minimum root mean squared error was approximately 20 for both the training and the validation data. One possible explanation why the error may not have decreased further, is that the model may have reached a local optimum in gradient descent, rather than the global optimum. The SGD optimizer corrects for momentum[14]. This could prevent reaching the local optimum.

9.3.2 The FTRL Optimizer

Follow The Regularized Leader (FTRL) is an algorithm developed by Google to counter sparsity in models [15]. For example, certain features in the images could only exist 1 or 2 times in our data set of over 9000 samples. This optimizer would pick up on that and take these sparse features into account. We wanted to see if this could be the case in our model.

9.4 Evaluation and Conclusions

9.4.1 The SGD Optimizer

The SGD optimizer did not work with its default learning rate of 0.01. This is likely to be caused by the exploding gradient problem[16]. When the learning rate was set at a lower value of 0.0001, the graph started at a value of approximately 29, which is a relatively high value compared to the starting point with use of different optimizers. Simultaneously, the graph showed a decreasing trend after 20 epochs. In order to investigate whether this decreasing trend was continuous after 20 epochs, the optimizer was run over a course of 100 epochs. It has become evident that the model loss stagnates after approximately 30 epochs, at a value of approximately 20. Thus, the SGD optimizer does not allow for enhancements on the model performance.The graphs are illustrated in Figure 20.

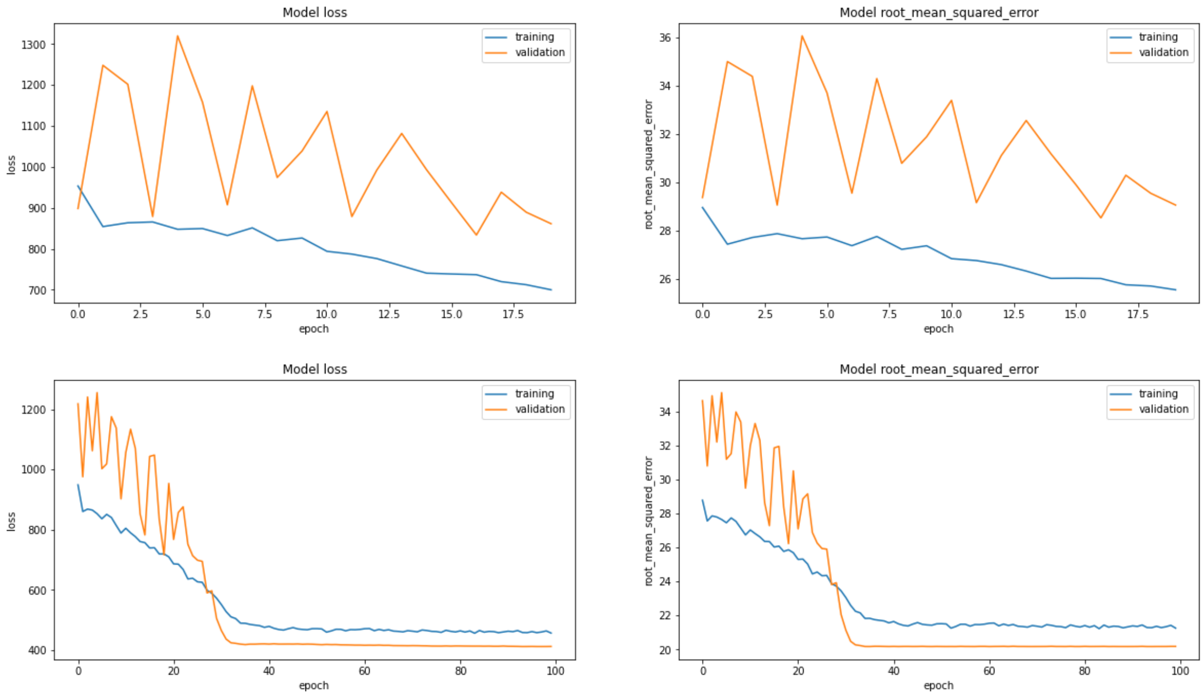


Figure 20: The learning curves of the model using the SGD optimizer after 20 epochs (top) and 100 epochs (bottom).

9.4.2 The FTRL Optimizer

As seen in the image below (Figure 21) this optimizer did not yield better results than our previous model. It even started overfitting after about 10 epochs. Therefore, we decided not to continue with this optimizer.

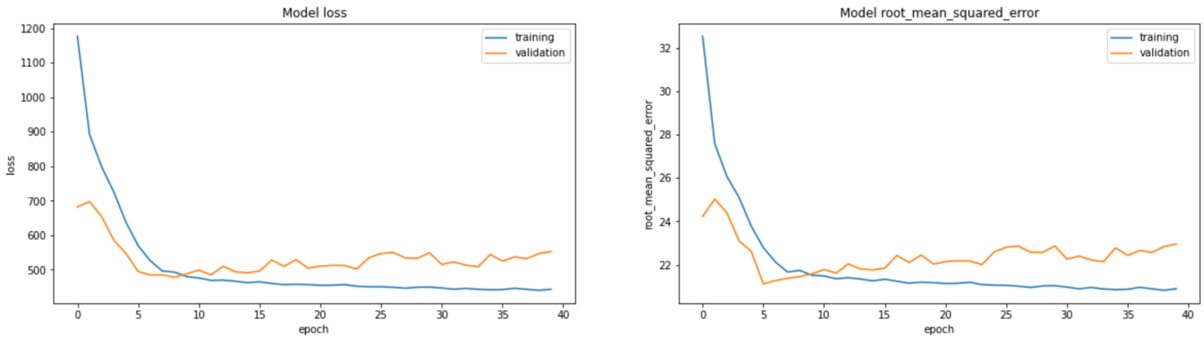


Figure 21: The learning curves of the model using the FTRL optimizer after 40 epochs.

In the end we have come to the conclusion that the optimizer Adam is a better fit for our model than the other optimizers we have found.

10. Milestone IV: Altering The Addition of Tabular Data

10.1 Introduction

In accordance with Chapter 8, the best manner to add tabular data to the model was with some more complexity, yielding the graphs shown in Figure 17. The acquired model now appears to learn quickly and the course of the learning curves is smooth. However, the minimum cost seems to be reached after 3 epochs, which highlights the informative nature of the tabular data for the predictive model. A possible side effect to this could be that the model now relies heavily on the tabular data to make predictions and trains its tabular branch much sooner than its convolutional branch. This could imply the model is not able to reach its potential; the model might preoccupy itself with training the tabular branch completely before it starts to train the convolutional branch. This in turn might require too many epochs to be viable. In order to prevent this occurrence, this chapter explores the option of altering the addition of the tabular data by its location in the network. We have also decided to add two columns to the tabular data. These columns contain the original size of the images before they were resized. We believe this could be valuable information as the quality of the picture could be an important asset to find the right "*Pawpularity*" score.

10.2 Data Analysis and Preprocessing

In accordance with section 1.2, all images were resized to a shape of 64x64x3. This resulted in a loss of information regarding certain aspects of the original images. The original size of the image is an example of a feature that was no longer taken into consideration after resizing the images. In order to counter this, we added the original length and width of every image to the tabular data. As seen in the image below it does not seem like the width and height of an image has any correlation with the *Pawpularity* score.

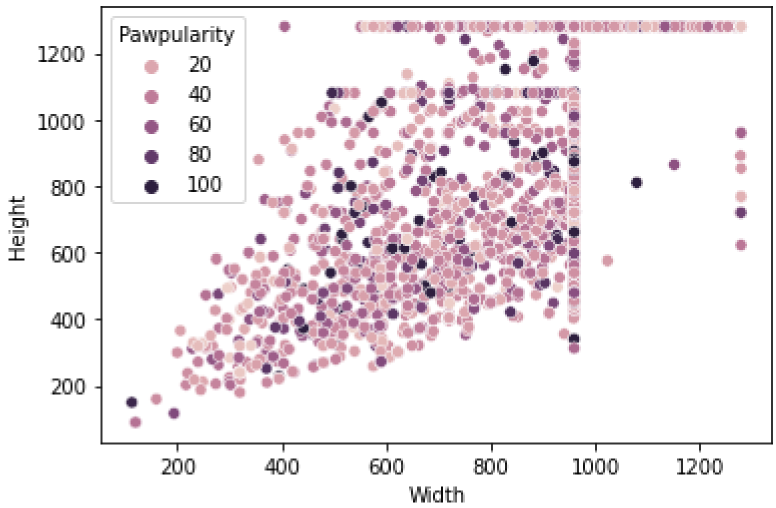


Figure 22: A scatterplot of the width and height of the original images with color coding of the Pawpularity. There is no apparent correlation between the size of an image and its Pawpularity score.

10.3 Model Pipeline and Training

In the previous version of the model with tabular data, the data was added at the end of the network, which made the curves decrease more quickly, though it stagnated at this minimum. An explanation for this is that the model sees the tabular data as more important and focuses mostly on this part of the model. Because of this, the model can apparently learn quickly until a certain value but it is unable to learn more valuable data from the convolution layers as it deems them as less important. In our new model we add the tabular data just after we have flattened the convolution data. This should counter this effect as the tabular data and the convolution data will be more intertwined because of the dense layers. It should result in a more even distribution of what data the model deems important.

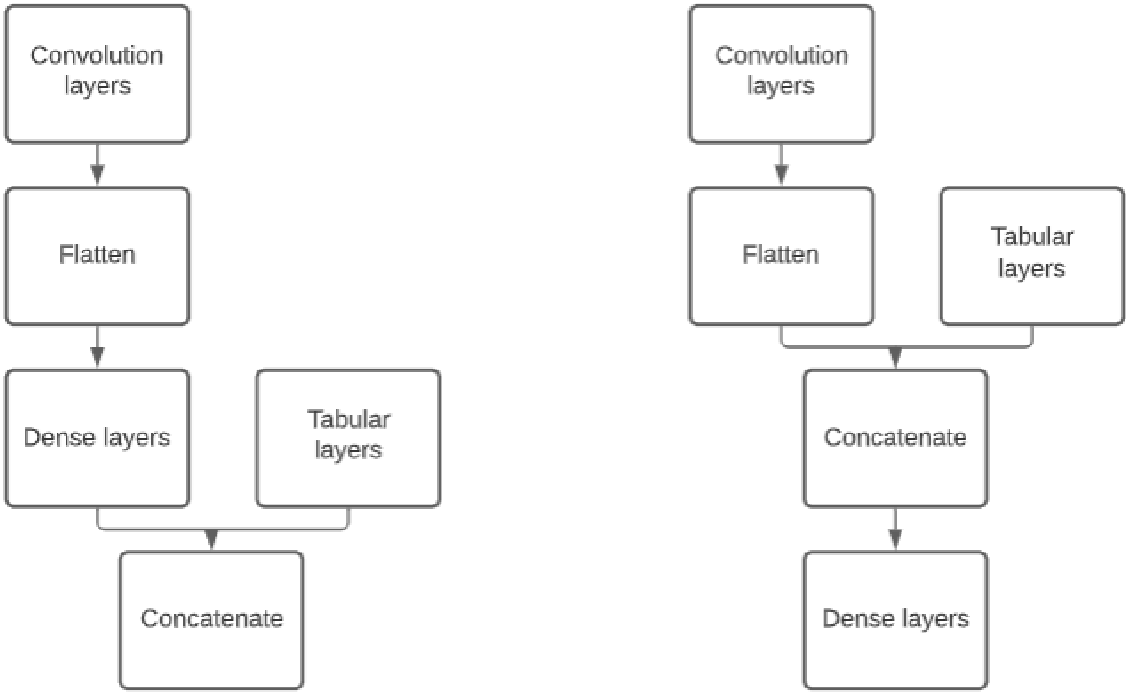


Figure 23. On the left we can see the old model of how tabular data was concatenated into the network. On the right we can see the new model.

10.4 Evaluation and Conclusions

10.4.1 The extra tabular data

The extra tabular data did not seem to impact the training RMSE at all. The results showed almost no changes in comparison to Figure 16.

10.4.2 The concatenated model

As we expected, the model indeed didn’t learn as fast as it did in the previous model. It now takes a lot longer for the model to reach a minimum. And as shown in Figure 24 even after 40 epochs the model seems to still be learning. Although the model below doesn’t look perfect, we do believe we can make tweaks to the model to make it better.

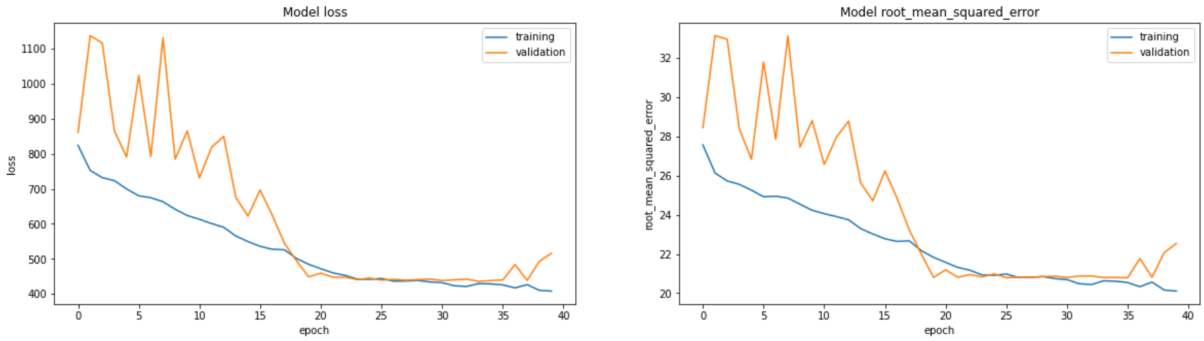


Figure 24: The learning curves of the new concatenated model.

10.4.3 The farce of the tabular data

After some careful analysis of our data, we observed that the prediction of the tabular data model is the same for almost all the images the algorithm receives. The algorithm doesn't consider the images when the tabular data is included. Instead, it finds a single value it can use in order to have a relatively low mean squared error. This is shown in Figure 25. The actual "*Pawpularity*" scores are distributed just as the entire data set as seen in chapter 1.2, Figure 1. But, as we can see more clearly in Figure 26, the predictions of the model are all centered around the "*Pawpularity*" score of 35.5. Zooming in (Figure 25), it clearly shows that all the predicted popularity scores do not become smaller than 34 or bigger than 37.

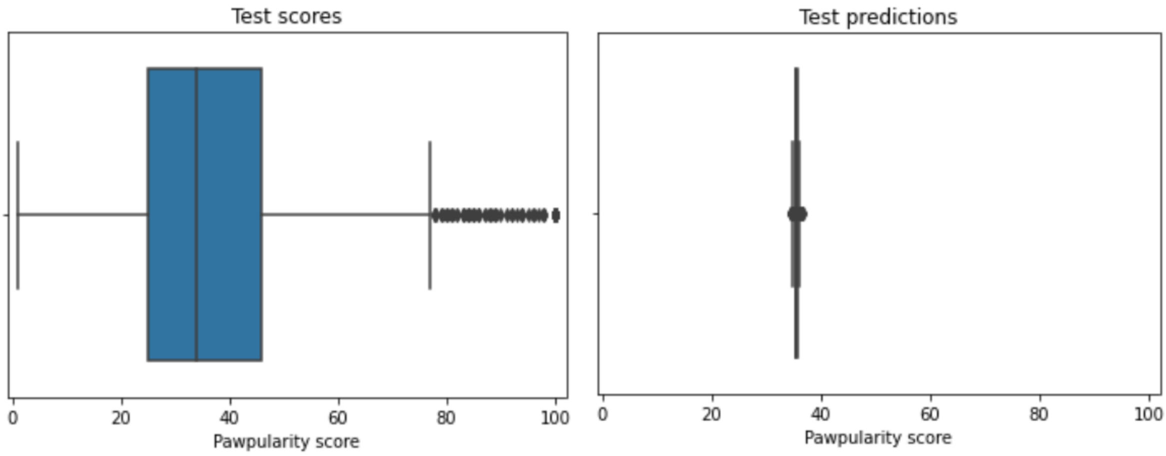


Figure 25: On the left a boxplot with the actual "Pawpularity" scores of the test data and on the right the predicted "Pawpularity" score of the test data by our model

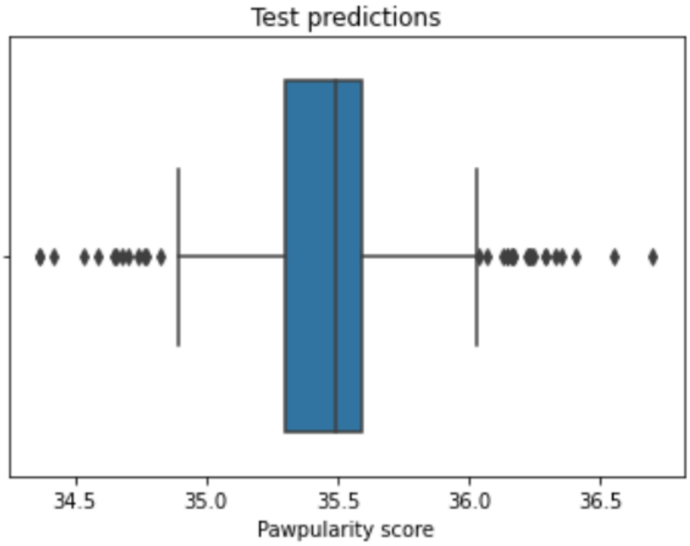


Figure 26: A zoomed in version of the boxplot of the predicted "Pawpularity" score

Knowing this, the behavior of the old model makes a lot more sense as well. In the old model, the tabular data and the image data were concatenated at the end of both models. It is likely that the old model saw this as a good way to lower the mean squared error quickly. This meant that after 2 to 3 epochs the old model was already giving all images the same prediction.

We decided that not using the tabular data is the best way to proceed.

11. Milestone IV: Network Architecture

11.1 Introduction

In the version of chapter 7, the problem of overfitting seemed to have been replaced with the problem of underfitting, as the validation loss is lower compared to the training loss (Figure 17). This illustrates the need for further tuning of the complexity of the model. In this chapter, we will explore modifications to the architecture of the network to optimize complexity.

11.2 Data Analysis and Preprocessing

No further modifications regarding data analysis or preprocessing were made.

11.3 Model Pipeline and Training

The model obtained after adding dropout and L2 regularization (chapter 6 and 7 respectively) contained an elaborate amount of regularization and started to underfit on the training data. To reduce the need for a heavy amount of dropout, the ReLu activation function was replaced. Instead, the Leaky ReLu activation function was used in all layers. This can solve a possible Dying ReLu problem and makes less normalization necessary.[20]

Various different architectures were considered for the next version of the model, in order to investigate the tuning of the complexity appropriately. Each version was trained over the course of 25 epochs.

11.3.1 Inspiration from VGG16

First, the network structure was altered in accordance with the structure from the VGG16 Convolutional Neural Network[17]. This implies adding fully connected convolutional layers to the network in order to allow the model to extract more complex features from the data set. In order to investigate the effects of this concept, all regularization methods were removed from the network. A schematic overview of the convolutional part of the network is provided in Figure 27.

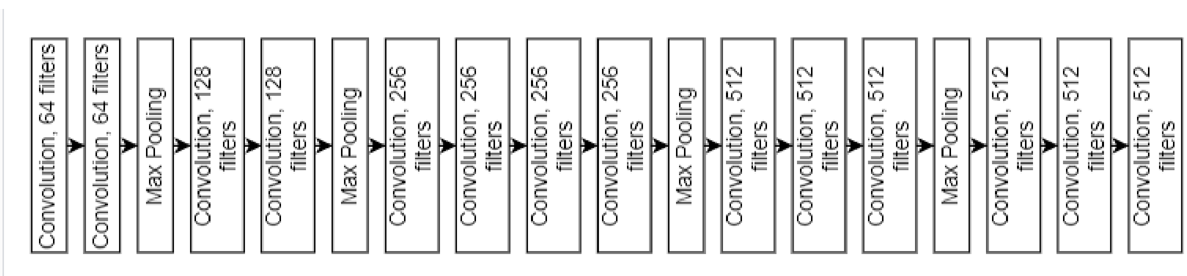


Figure 27: The VGG16-inspired structure of the network before the flatten layer.

11.3.2 Adding Dropout

The model obtained in section 11.3.1 is quite complex and definitely requires some regularization as it is overfitting. Adding dropout is the first step in making the model less prone to overfitting. A dropout layer with a dropout rate of 0.2 was added after every max pooling layer and a dropout layer with a rate of 0.5 was added after every hidden dense layer. The values of the dropout rate are relatively low, compared to the ones we used in earlier versions of the model. Through unsystematic trial and error, it was established these dropout rates were sufficient when combined with the Leaky Relu activation function.

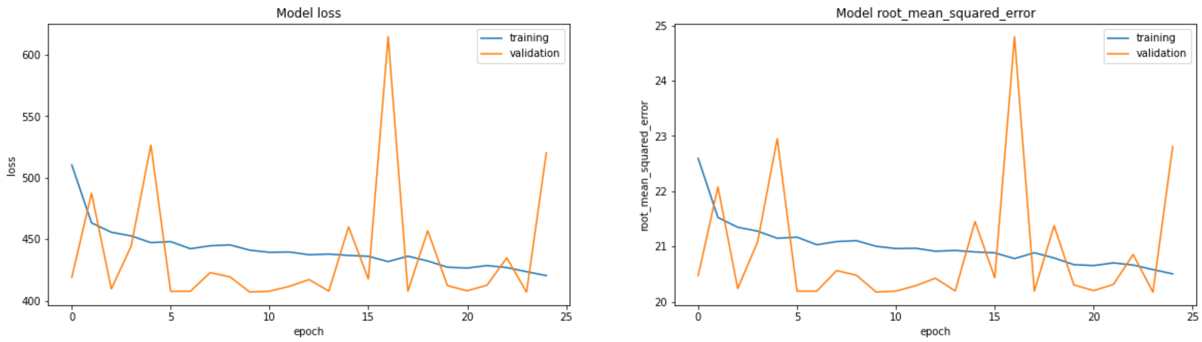
11.3.3 L2 Regularization

L2 regularization was only applied in the last convolutional layer and the first dense layer.

11.4 Evaluation and Conclusions

11.4.1 Inspiration from VGG16

After adding complexity, the training loss shows a continuous decrease over the course of 25 epochs. However, the model does appear to be overfitting as small changes in the training data cause large maxima and minima in the validation data.



. Figure 28: The learning curves of VGG16-inspired model

11.4.2 Adding Dropout

Figure 29 shows that adding dropout did not sufficiently decrease the tendency of the model to overfit on the training data. Thus, adding regularization seems necessary.

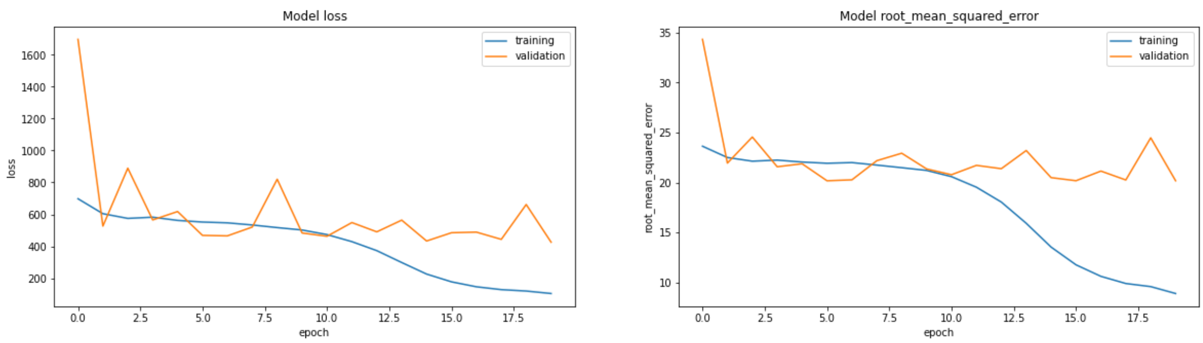


Figure 29: The learning curves of the VGG16_inspired model with dropout

11.4.3 L2 Regularization

After adding L2 regularization, the learning curves (Figure 30) seem to have stabilized towards the end of 25 epochs. Though the training loss shows a continuous decrease, the validation loss seems to have stagnated yet again at a value of approximately 20 for root mean squared error.

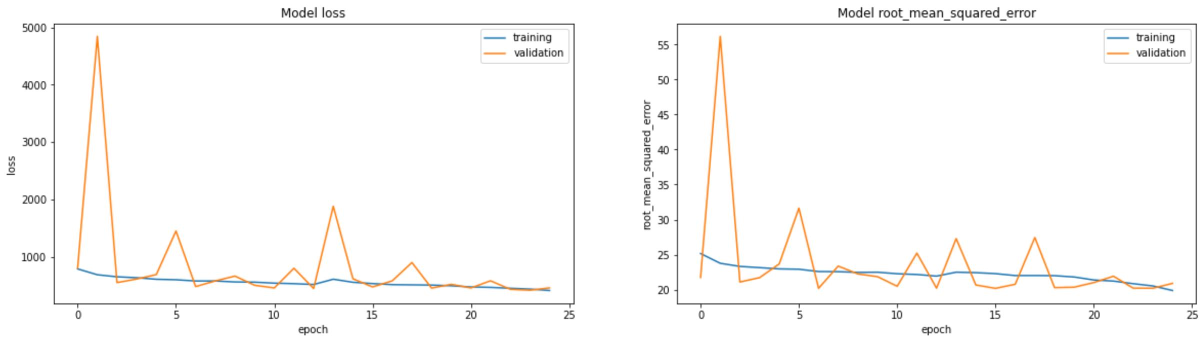


Figure 30: The learning curves of the model after adding L2 Regularization

12. Milestone IV: Data Augmentation

12.1 Introduction

For this image regression problem, the number of training samples (9912) is quite small and the performance of the model on the validation set does not increase beyond a root mean squared error of approximately 20. Given that regularization does not improve the results (as shown in chapter 11), it might be a problem related to the small data set. In this situation, data augmentation, known as a technique to increase scarce data in computer vision tasks, could be valuable[18].

12.2 Data Analysis and Preprocessing

There are several augments that can be used to preprocess images. The simple model described in section 1.3 was used to investigate what augments would be most valuable for the image data in this specific task. The results are presented in Figure 31. The five augments that resulted in the lowest root mean squared error scores on the validation data are displayed in bold letters.

Model	Loss	RMSE	Val. Loss	Val. RMSE
baseline	81.22	8.73	565.62	23.78
batch norm	127,43	11,29	515,94	22,69
samplewise std norm	93,18	9,63	8327,22	91,28
featurewise std norm	57,53	7,59	3512,38	59,32
featurewise center	39,96	6,32	580,7	24,02
samplewise center	112,88	10,63	1092,93	33,07
horizontal flip	**44,41**	**6,66**	**555,84**	**23,55**
vertical flip	**47,31**	**6,88**	**559,48**	**23,66**
height shift	47,57	6,88	571,28	23,84
width shift	42,41	6,5	586,62	24,16
height & width	**49,56**	**7,04**	**534,99**	**23,17**
zoom range	49,15	7,01	641,85	25,32
shear range	152,32	11,04	581,36	24,03
rotation range	**47,88**	**6,92**	**516,05**	**22,67**
brightness range	**70,64**	**8,4**	**507,25**	**22,59**

rescale	44,85	6,7	596,94	24,36

Figure 31: This table shows the results of the experiments on data augmentation on the old model. The augments that resulted in the lowest validation RMSE values are bold.

Further analysis of these five augments was conducted with the more complex model, as it was last described in Chapter 11. These augments were separately added to the model so they could be tested individually, each augment was tested four times on 50 epochs. Additional combinations of augments were also tested. Below are the average results of the four tests.

Model	Loss	RMSE	Val. Loss	Val. RMSE
rotation (30°)	**69.69965**	**8.348025**	**464.8992**	**21.519725**
horizontal & vertical	70.75435	8.40785	717.0051	26.17835
brightness range (0.75-1.25)	70.5524	8.398625	847.8225	28.308975
width shift range (0.25)	70.45525	8.392525	860.278	27.889675
baseline	69.78875	8.3513	484.242	21.9556
horizontal flip	69.6894	8.347775	690.4875	25.81175
vertical flip	**70.20858**	**8.378375**	**515.5305**	**22.6671**
height shift range (0.25)	68.9698	8.3036	842.3513	28.32665
height & width (0.25)	**69.35865**	**8.32645**	**520.4468**	**22.761075**

Figure 32: This table shows the results of the experiments on data augmentation on the new model . The augments that resulted in the lowest validation RMSE values are bold.

12.3 Model Pipeline and Training

Analysis of the effects of the augmentation was conducted whilst training the model over 50 epochs.

12.4 Evaluation and Conclusions

As seen in the results there is only one augmentation that resulted in a validation RMSE lower than the baseline RMSE. This was the rotation range augmentation. There were a few other augmentations that were reasonably close to the baseline RMSE: the vertical flip, and the combination of height and width shift range.
Since every augmentation was only tested 4 times, outliers could have had a considerable impact on the average RMSE which in turn might have skewed results to be higher than expected. For example brightness range was the best performing augmentation on the old model but for the complex model, it was the second worst by a small amount. Rotation range, vertical flip and height combined with width shift have however performed well in both

tests, therefore these three augmentations are the best choices and will be used for the model from this point onwards.

13. Milestone IV: K-Fold Cross Validation

13.1 Introduction

It was established in section 1.2 that the data set is relatively small to train an adequate convolutional neural network. K-Fold Cross Validation is known to provide a robust estimate of the performance of a model on new data[19]. Given the small number of samples in this data set, it is a valid alternative to the manual split into training and testing data, the way it was applied in section 1.2. In addition, K-fold Cross Validation provides a way of accurately comparing the performance of the different models formed thus far. In this chapter, a total of three models will be compared to one another; a convolutional neural network (as described in Chapter 11), a convolutional neural network with applied data augmentation (in accordance with Chapter 12), and a much simpler version of the model (as described in Chapter 7). The addition of the simple model was added to make an estimation of the value of added complexity.

13.2 Data Analysis and Preprocessing

In previous versions of the model, the data was split into training and testing data in accordance with section 1.3. Now, the data will be split into 5 folds, one of which will serve as the validation data in every iteration. The number of 5 folds was chosen in order to save computational expenses needed to run more folds.

13.3 Model Pipeline and Training

Each of the models was trained using 25 epochs.

13.4 Evaluation and Conclusions

Figure 33 shows the validation RMSE for every fold and the cross-validated mean RMSE for every model. The lowest cross-validated mean RMSE score belongs to the simplest model. However, it appears that the scores of the more complex models suffer more from outliers which makes the mean RMSE higher.

Model	**Cross-validated Val RMSE scores**					**Cross-validated mean Val RMSE**
Simple Model	20.78	20.89	21.02	20.18	20.46	20.67
Convolutional neural network without data augmentation	25.31	28.78	21.09	20.21	20.40	23.16
Convolutional neural network with data augmentation	29.00	20.86	20.90	20.15	20.39	22.26

Figure 33: This table presents the 5-fold cross-validated scores of the tested models.

So to conclude, our simple model got on average the best results, while adding more complexity to the model by implementing tabular data, building the model similarly to the VGG16-inspired model or adding data augmentation only made the model fluctuate more causing outliers.

14. Milestone V: Conclusion and Discussion

As seen in chapter 13.4 our best performing model was the model described in section 7.3, making use of a deep convolutional network in combination with different regularization techniques. Making the model even more complex with use of a VGG16-inspired structure (chapter 11.3) did not result in an improved version. Similarly, adding data augmentations as seen in chapter 12 resulted in outliers, making the model inconsistent. Adding the tabular data on the other hand made the model very consistent. The reason for this was that the model did not learn anything; it just saw all images as equal and predicted similar scores for all images as seen in chapter 10.4.

Taken together, this report shows that the peak performance of a model with this structure seems to be around a root mean squared error of approximately 20. Though this seems like a rather poor performance, a myriad of different possibilities was tried with the purpose of further enhancement, through a trial-and-error-like methodology. Perhaps the overlapping nature of some of these possibilities contributed to an unclear image of the way a given change affected the performance of the model. In the future, a more systematic and goal-oriented approach might yield better results.

We believe that the data provided by Petfinder.my was not indicative enough to make good estimates. We suspect the metric used to calculate the '*Pawpularity*' was flawed in several ways. The score in itself does not necessarily correspond to a popular profile; pets can be assigned to a high or low score for a plethora of reasons. For instance, a new profile of a cute pet is unlikely to have a high '*Pawpularity*' score as the traffic of this profile needs time to increase. Similarly, an older profile is more likely to have a higher score regardless of its popularity. Another possible explanation for a high '*Pawpularity*' score is an unclear picture of a pet that could be unpopular by itself, but due to people clicking on it to see it more clearly the 'Pawpularity' will increase. Additionally, the tabular data shows that some of the pictures with different '*Pawpularity*' scores show the same features. Furthermore, the distribution of the features in the images does not seem indicative of the score, as explained in section 8.2. These considerations illustrate a selection of the reasons why the '*Pawpularity*' score might not be indicative of the popularity of a pet's profile on the Petfinder.my website. This resulted in the images getting seemingly random '*Pawpularity*' scores with no real correlation. This, in turn, made it hard for our model to learn what features were important to determine the '*Pawpularity*'.

For further research another model could be made using a different machine learning library like Pytorch or Fastai, these libraries might be more applicable on the given data in this project, and therefore could be used to create better performing models.

Another interesting future trial would be to combine the tabular data with the data from the Kaggle Petfinder.my competition of 2019. The data of the competition from 2019 looked at the speed of the adoption instead of the *'Pawpularity* score'. There is a large probability that an animal that is quickly adopted should have a high *'Pawpularity* score'. An animal that is not adopted after 100 days should have a low *'Pawpularity* score'.

We should also take into account that our data set was rather small for a convolutional neural network. The models we built were made to optimize this smaller data set and were very prone to overfitting. With a larger data set, models with less regularization where the training data trains better might be preferred as a larger data set produces a more reliable model with a more general fit for newly uploaded images.

In conclusion, we managed to make the optimal model for the data that was provided to us. For a different amount of data the optimal model might look different, as it would be likely to require less regularization. Perhaps such a model would show a better performance regarding the prediction of the '*Pawpularity'* score, although we think redefining this metric would yield the best results in the future.

References

1. Lepper M, Kass PH, Hart LA. Prediction of adoption versus euthanasia among dogs and cats in a California animal shelter. J Appl Anim Welf Sci. 2002;5(1):29-42.
2. PETA. The Deadly Consequences of 'No-Kill' Policies. [Internet]. Available from: https://www.peta.org/features/deadly-consequences-no-kill-policies/. [Accessed at 11-01-2022].
3. Petfinder.my. About Petfinder.my. [Internet]. Available from: https://www.petfinder.my/about.htm. [Accessed at 11-01-2022].
4. Petfinder.my. Cuteness Meter — How Attractive Are Your Photos?. [Internet]. Available from: https://www.petfinder.my/campaigns/cutenessmeter.htm. [Accessed at 11-01-2022].
5. Molnar C. Interpretable Machine Learning. Lulu.com; 2020.
6. Kaggle. PetFinder.my - Pawpularity Contest: Data. [Internet]. Available from: https://www.kaggle.com/c/petfinder-pawpularity-score/data. [Accessed at 12-01-2022].
7. Machine Learning Mastery. Use Early Stopping to Halt the Training of Neural Networks At the Right Time. [Internet]. Available from: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/. [Accessed at 17-01-2022].
8. towards data science. The Vanishing Gradient Problem. [Internet]. Available from: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484. [Accessed at 16-01-2022].
9. DeepLearningAI. Why non-linear activation functions. [Video]. 2017. Available from: https://youtu.be/NkOv_k7r6no. [Accessed at 16-01-2022].
10. DeepLearningAI. Normalizing activations in a network. [Video]. 2017. Available from: https://youtu.be/tNIpEZLv_eg. [Accessed at 16-01-2022].
11. DeepLearningAI. Dropout regularization. [Video]. 2017. Available from: https://youtu.be/D8PJAL-MZv8. [Accessed at 16-01-2022].
12. Neptune. Fighting overfitting with L1 or L2 regularization [Internet]. Available from: https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization [Accessed at 21-01-2022].
13. Machine Learning Mastery. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. [Internet]. Available from: [Accessed at 24-01-2022].https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
14. Towards data science. Various Optimization Algorithms For Training Neural Network. [Internet]. Available from: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6. [Accessed at 24-01-2022].
15. Google Inc. Ad Click Prediction: a View from the Trenches . Available from: http://www.eecs.tufts.edu/~dsculley/papers/ad-click-prediction.pdf [Accessed at 24-01-2022].

16. Neural Networks and Deep Learning. Why are deep neural networks hard to train? [Internet]. Available from: http://neuralnetworksanddeeplearning.com/chap5.html. [Accessed at 24-01-2022].
17. Neurohive. VGG16 – Convolutional Network for Classification and Detection. [Internet]. Available from: https://neurohive.io/en/popular-networks/vgg16. [Accessed at 26-01-2022].
18. DeepLearningAI. C4W2L10 Data Augmentation. [Video]. 2017. Available from: https://youtu.be/JI8saFjK84o. [Accessed at 17-01-2022].
19. Machine Learning Mastery. Evaluate the Performance Of Deep Learning Models in Keras. [Internet]. Available from: https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/. [Accessed at 24-01-2022].
20. Kenneth Leung, 'The Dying ReLU Problem, Clearly Explained' *Towards Data Science* (30-03-2021) https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24#0863 [Accessed at 2-02-2022].