Algorithm Exploration for Ranking Animal Cuteness

Rebecca DeSipio
Virginia Tech
rdesipio26@vt.edu

Fan Yang
Virginia Tech
fanyang@vt.edu

Abstract

Each year, thousands of unadopted shelter animals are euthanized. Often, shelters will post pictures online for those interested in adopting. These photos can be crucial to saving an animal. With advancements in Machine Learning (ML) and Artificial Intelligence (AI), these websites build AI-driven models to provide rankings for these images. This study will explore various ML regression and classification methods including Convolutional Neural Networks, Support Vector Machines, and k-Nearest Neighbors in an attempt to accurately provide a score for given images. This study follows from a Kaggle competition entitled the "Pawpularity Conteset" where users were challenged to predict pet popularity rankings using ML methods. The best models from the competiton were able to reach $\sim$17.5 RMSE, while we were able to achieve $\sim$19.5.

1 Introduction

Earlier this year, Kaggle (an online dataset database) held a competition to predict the popularity score of animal pictures using AI/ML methods. This competition used pictures from PetFinder.my, a Malaysian animal welfare platform, along with the associated ranked scores. Many shelters will use similar websites to advertise their animals such as PetFinder.com based in the United States.

Our project explored regression and classification using ML and deep learning (DL) to predict the popularity scores of animals. Regression methods included a Convolutional Neural Net (CNN) and Support Vector Machines (SVMs), and classification used K-nearest neighbors (KNN) and a CNN. The following sections will define the data, elaborate on these methods, and discuss findings.

The Python code and models can be found on GitHub: *https://github.com/MaxLHood/AML.*

2 Problem and Dataset

The competition dataset contains a total of 9,912 images of cats and dogs up for adoption. The pictures might contain other objects such as people, additional animals, and can be a single image or collage. A CSV file was also provided with the image title as {ID}.jpg and the Pawpularity score (ranked 0-100%), which is a proxy for the popularity of an animal. Using an AI driven "Cuteness Meter", the Pawpularity scores are determined by user traffic (i.e. the more viewings a photo receives, the higher the score) and based on unique views of the pets' profile pages, adjusted for duplicate clicks, crawler bots, and sponsored profiles. For this competi-

Figure 1: Examples of the raw images

tion, the Root Mean Squared Error (RMSE) is the evaluation metric, defined as: RMSE $= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$ where $\hat{y}_i$ is the predicted value and y_i is the original value for each instance (i).

The provided pet images served as input feature vectors and the corresponding Pawpularity scores the labels. Before building any models, we plotted the data distribution histogram (Figure 2). The majority of scores range between 20-50% and visible spikes are at the tail ends, both indicating a skewed distribution.

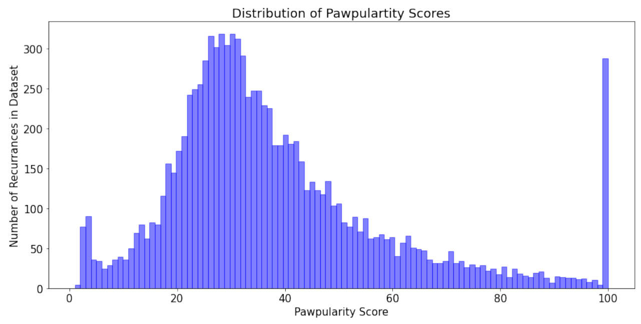

Figure 2: Distribution of “Pawpularity” scores data

This competition has since concluded, and the top RMSE of the public leaderboard was ~ 17.5. This was used as a benchmark for the work computed in this project. The competition details and dataset can be found at [4]. In the next section, we will discuss the implementation of each regression and classification method.

3 Implementation

Convolutional Neural Network (Regression)

CNNs are commonly used methods for image based regression problems. Through convolutional layers, backpropagation automatically learns image filters (i.e. local patterns). Stacking convolutional layers can also help the model find large complex features that are composite of smaller ones, such as features associated with “cuteness” of pet images potentially correlated with Pawpularity. The initial model architecture was based on [2]) and consisted of four 2D-convolution pooling layers followed by a feed forward network shown in Table 1.

Layer Type	Parameters	Activation
Convolution (filters, kernel)	128, 12	ReLU
Max Pool (pool size)	3	
Convolution (filters, kernel)	256, 8	ReLU
Max Pool (pool size)	3	
Convolution (filters, kernel)	512, 5	ReLU
Max Pool (pool size)	3	
Convolution (filters, kernel)	1024, 2	ReLU
Max Pool (pool size)	3	
Linear (hidden, dropout)	4096, 0.5	ReLU
Linear (hidden, dropout)	2048, 0.5	ReLU
Linear (hidden, dropout)	1024, 0.5	Tanh
Linear (hidden, dropout)	1	Linear

Table 1: CNN Model

In total, this model has 22,216,577 trainable parameters. Each pooling layer had a gradually decreasing filter and window size for the model to search for compositions of small patterns. The kernel sizes, number of filters, hidden dimension sizes, etc., were chosen so the model would be as over-parameterized as possible. Various other configurations were also tried, but the architecture remained largely unchanged.

Next, we tried a transfer learning approach using the ResNet50[3] model from the Keras Python library. This residual CNN was trained on over a million images from the ImageNet database, with undoubtedly many images of cats and dogs. The model has been better trained to recognize animal features and differentiate cats ver-

Layer Type	Parameters	Activation
ResNet50		
Linear (hidden, dropout)	2048, 0.5	ReLU
Linear (hidden, dropout)	1048, 0.5	ReLU
Linear (hidden, dropout)	512, 0.5	ReLU
Linear (hidden, dropout)	256, 0.5	ReLU
Linear (hidden, dropout)	1	Linear

Table 2: ResNet50 Model

sus dogs. By adding a fully connected regression head to this model, we hoped to leverage this pre-training and fine-tune the weights to be responsive to Pawpularity. Since the internal architecture of ResNet50 is beyond the scope of our project, Table 2 reports simply on the high-level model structure.

In total, this model has 30,485,505 trainable parameters. As before, linear layers were stacked with progressively halving dimensions, terminating in the uni-dimensional regression output.

Support Vector Machine Regression (SVR)

SVMs were chosen for a similar rationale as CNNs. Features that can predict higher/lower Pawpularity scores can be used to estimate a function that models this relationship through the Epsilon Support Vector Regression. A considerable amount of effort was spent fitting models using linear, polynomial, sigmoid, and RBF kernels. Having had previous exposure training SVMs by tuning the optimizer and adjusting the C parameter, a similar technique was employed. The hypothesis space was: {linear, poly-3, poly-5, poly-10, sigmoid, RBF} X {C: C in [0, 2]} x {Epsilon: Epsilon in [0, 2]} x {Tolerance: Tolerance in [0, 0.1]} x {Gamma: 'auto', 'scale'}. With minimum prior experience and an enormous hypothesis space, a brute-force grid search approach was implemented in these ranges.

Classification: KNN & CNN

Classification was explored to determine if an image could be accurately classified with a preassigned label. The raw images were preprocessed two ways: the input image converted to the raw RGB pixel intensities, and a color histogram based on hue, saturation, and value to characterize the color distribution. This preprocessing follows the example found in [5]. The label scores were split six different ways: raw scores (100 bins: 0-100%), rounded scores (11 bins: by 10%), twenty split (5 bins: by 20%), tri-split (3 bins: $\leq$10%, =10%, otherwise), half scores (2 bins: $\leq$ 50% and $>$ 50%), and binary split (2 bins: 100% and otherwise). For example, the tri-split images were assigned a 0 (if $\leq$10%), 10 (if =10%), and 100 otherwise. Based on a test image, we hoped to build a model that could accurately predict which bin (0, 10, or 100) the Pawpularity score would fall into.

In addition to KNN, CNNs with classification heads were explored. Numerous models were built, varying the number of layers and parameters. The ResNet50 model was also implemented. The results are discussed in the next section.

4 Results

CNN (Regression)

Given the small image dimensions (128x128x3) it was relatively easy to train an over-parameterized CNN and let the learning process select the relevant features. As per the competition's specification, Mean Squared Error was selected as the loss function, while an Adam Optimizer with a learning rate of 0.001 was used for training.

We achieved good results with this naive CNN specification, with a minimum RMSE of 20 via cross validation. RMSE represents the average deviation of predicted scores from the real scores, so the model in Table 1 predicts scores within 20 points (roughly) of the actual values. In all, it took about three hours of work with no experience in DL to fit a naive model that is within 2.5 RMSE from the top Kaggle teams. This left us with a strong impression of the power and convenience of CNNs.

Model	Data Split Type	Min. Validation RMSE
CNN 1	Entire dataset	20.10 (10 epochs)
CNN 2	Pawpularity $<$ 100	20.05 (5 epochs)
CNN 3	5 $<$ Pawpularity $<$ 100	20.05 (10 epochs)
ResNet50	Entire dataset	19.50 (50 epochs)
ResNet50	Pawpularity $<$ 100	20.05 (25 epochs)

Table 3: CNN Results

After repeated runs under different parameterizations, this model was still unable break 20 RMSE. The tail ends of the data distribution could have been distorting training, so we next tried only training with images between 5 and 99 Pawpularity, causing an interesting result: the best RMSE was achieved after a single epoch reaching its minimum of 16.7, after which it started overfitting. Removing the tails considerably improved the accuracy of predicting images that did not have Pawularity scores at the tails, but unfortunately did not consistently improve prediction of overall test inputs.

Table 3 contains all CNN results. The second and third rows show results from training on a subset of the data set. Validation was done on the same 1,983 images for all experiments.

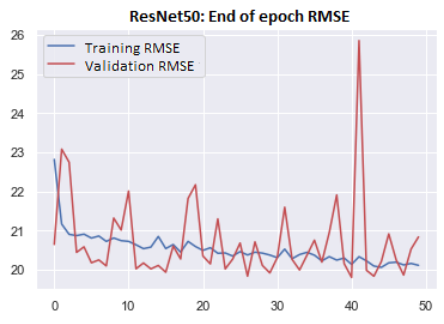

Figure 3: ResNet50 finetuning outputs

Next, we wanted to try a different approach with the ResNet50 model. The best results came from fine tuning ResNet50, which improved over a course of 50 epochs, as seen in Figure 3. Since residual CNNs are complex and ResNet50 has over ten million more parameters than our generic CNN, fine tuning took quite a bit longer.

The validation RMSE eventually stabilized between 19.5 and 20.5, reaching a minimum of 19.5. This put our model in the top 25th percentile of public leaderboard scores. Overall, we are impressed by the power of transfer learning, but remain skeptical that ResNet50 by itself would be able to break the 19.0 barrier.

SVM (Regression)

SVRs were the next method explored to see if improvements could be made over CNNs. When building an SVR model, 'C' is a user-chosen parameter that scales the penalty for slack violations. Epsilon governs the width of a band in which training loss is not penalized; larger values permit a larger margin of error [1]. We elected to use the Scikit-learn default values for *scale* and *training tolerance*, leaving C and epsilon as the only parameters to be searched for each kernel. Since the scale is most likely data dependent, these values are unknown a priori. This was solved by iterating in a manual evolutionary process: we ran the models with a random initial configuration, inspected the fitted histograms, removed the meaningless results, and gradually moved parameters toward sensible values. This was done simultaneously for all four kernel types, and on the full dataset as well as the subset sans tails. The process spanned over five days. Our best results are reported in the Table 4.

Model	Parameters	Min. RMSE
Linear ($<$ 100)	C=0.001, e=5	20.38
Linear (full)	C=0.05, e=2	22.10
Polynomial ($<$100)	C=0.2, e=0.5, d=5	23.19
Polynomial (full)	C=0.2, e=0.5, d=5	23.93
Sigmoid ($<$100)	C=0.05, e=1	20.44
Sigmoid (full)	C=0.1, e=15	20.07
RBF (full)	C=0.01, e=15	20.06

Table 4: SVM Results (C is slack penalty weight, e is insensitive loss width, d is polynomial degree)

Depending on kernel type and configurations, each SVR model took 1-2 hours to fit, which is considerably faster than it took to reach convergence on the generic CNNs ($>$7 hours) and ResNet50 ($>$35 hours). However, the advantage of quickly obtaining a global minima is overshadowed by the fact that we had to run hundreds of them in a manually curating process. As reported in Table 4, the validation RMSE obtained are similar to CNNs. All fitted models had lower variances than the empirical distribution. Linear and polynomial models produced histograms ranging from 0 to 80 that resembled the validation distribution. However, sigmoid and RBF predictions, despite producing superior RMSE, predicted nearly degenerate distributions around 38

– this result says that if we always predict an image to have a score of 38, we will on on average, be within 20 points of the actual score. Our main takeaway from doing all this is that SVM regressions are probably preferable in the hands of experts who are able to pinpoint the right parameters with just a few runs.

Classification: KNN & CNN

Classification was solely an exploratory analysis. Starting with the KNN method, it was discovered early on the color histogram preprocessed data outperformed the RGB pixels by ≈1.5%, so all results use the color histogram data summarized in Table 5 for each of the six data splits. The results show an overall trend as the number of bins decreases, the accuracy increases. However, this is an expected result with KNN and while it may appear some of the results are promising, they lend to be skewed due to the data distribution.

Data Split Type	Max. Accuracy
Raw Scores	2.62%
Rounded Scores	26.93%
Twenty Split	51.23%
Tri Split	92.74%
Half Scores	78.92%
Binary Split	96.62%

Table 5: KNN Classification Accuracy Results

Next, CNNs were explored to see if any meaningful results could be achieved. The first CNN model built as a baseline was similar to that in Table 1, with the model architecture based on [2]. The same six data splits were run, and the results varied. Overall, the models provided mixed accuracies and each consecutive epoch resulted in the same accuracy, meaning the model was ultimately not learning. Various configurations were then tried over a couple of days, altering the number of layers, layer features, loss function, and preprocessed data. However, the results remained the same - the accuracy continue to go unchanged. This lead to the conclusion that the dataset cannot be learned for classification purposes. The data is quite noisy due to inconsistencies in what would be considered "cute". While it was an expected result that classification would be unsuccessful as the original problem statement is regression, it was a worthwhile exploration and a learning experience nonetheless.

5 Conclusion

Although the sole objective of the Kaggle competition was to achieve the lowest possible validation RMSE, the goal of this project was to gain experience and learn through hands-on work and experimentation. From a modeling perspective, our key takeaways are that SVR would be an incredibly powerful tool in the hands of a skilled operator, but it is considerably less convenient than CNN regression for novices, especially considering the ease of leveraging pre-trained models. After interpreting all of the model predictions, we concluded that the extreme values in the dataset are noise, and the lower prediction variance indicate that pet pictures are probably more uniformly "cute" than their Pawpularity scores would indicate.

While doing regression on the data without tails, we achieved a minimum RMSE value of 16.7. We conjectured that if we could correctly classify images as $\{100, \text{not } 100\}$ or $\{<5, 5< \text{and} <100, 100\}$, then we could possibly produce a strong ensemble model. We worked hard without success at finding such a model. This gave us the realization that human-influenced data is inherently contradictory due to different preferences, and this might be a recurring phenomenon in other datasets with subjective inputs.

Overall, this report showcased a chronological ML project workflow from minimal experience to moderate proficiency in the usage of CNNs, KNN, and SVMs for a challenging problem. We were able to demonstrate our knowledge relating to classification and regression tasks via models discussed in class as well as researched online. Although getting accurate predictions was not a primary goal, we are very proud to have come so close to achieving RMSE scores within 1.5% of the world's best Kagglers. This project was a great first step for our future Machine Learning careers.

6 Contributions

Fan Yang worked on the regression CNN and SVR methods, and Rebecca DeSipio worked on KNN and classification CNN. Both team members contributed evenly to the project on a whole.

References

[1] Chaya Bakshi. *Support Vector Regression*. URL: `https://medium.com/swlh/support-vector-regression-explained-for-beginners-2a8d14ba6e5d`.

[2] Jason Brownlee. *How to Classify Photos of Dogs and Cats (with 97% accuracy)*. URL: `https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/`.

[3] Aakash Kaushik. *Understanding ResNet50 Architecture*. URL: `https://iq.opengenus.org/resnet50-architecture/`.

[4] PetFinder. *Pawpularity Contest*. URL: `https://www.kaggle.com/c/petfinder-pawpularity-score/overview`.

[5] Adrian Rosebrock. *k-NN classifier for image classification*. URL: `https://pyimagesearch.com/2016/08/08/k-nn-classifier-for-image-classification/`.