Visual explanation of features that influence popularity of pet images

Isha Shah
Georgia Tech
ishah40@gatech.edu

Abstract

In this paper, we explored the use of existing pre-trained deep neural network architectures (ResNet, VGG, DenseNet, and EfficientNet) for transfer learning and build custom layers on top to include the metadata features in determining the popularity of pets (cats & dogs). We further use three different saliency map techniques: Vanilla Gradient, GradCam and RISE, to visually explain the important features of the pet's image. We came up with several hypotheses to interpret popularity among different images and using saliency maps, we presented evidence to confirm or reject the hypothesis.

1. Introduction

Despite the numerous applications of AI aiding all forms of human intelligence, one domain that has not yet been much explored is Naturalistic Intelligence. In this project, we set out to challenge ourselves in exploring the applications of Deep Learning to emulate human's perception or emotional stimuli towards subjects captured in images, specifically of pets by detecting a "popularity" score, a metric derived to measure a pet's cuteness or attractiveness and was termed *pawpularity*.

By understanding what features make an image "popular," we can leverage those features to take photos that highlight the cuteness of the pet. This can be useful for pet adoption services wherein most people shortlist pets that they want to consider adopting prior to visiting the shelter based on how these pets look in profile images.

While we focused on pets (dogs and cats) due to the nature of our data, this study can be easily extended to identify photographic styles and designs that make pictures more captivating (e.g., for brand advertising, photography techniques and other similar applications).

2. Data

We used the 'PetFinder.my - Pawpularity Contest' dataset from Kaggle. The data consists of three major items:

i. Pet Images of cats and dogs
ii. Metadata of images includes binary features describing the subject, like focus, eyes of the pet facing front, pet having a clear face, single pet, action at the time of snap, presence of accessory, group photo, collage, presence of human, occlusion, and blur.
iii. *Pawpularity* (popularity) score for each image on a scale of 0-100. This number is derived by taking into account page view statistics, normalising for traffic data and excluding duplicate clicks, bot accesses, and sponsored profiles.

3. Approach

In this work, we attempted to predict a pet's popularity score based on an image of the pet and some metadata features of this image, using several convolutional neural network (CNN) architectures known to work well on image classification. We used these neural networks to:

1. Perform regression on the images and metadata to predict the popularity score, and
2. Generate saliency maps to highlight which parts of the images appear to be most significant resulting in high or low popularity scores. We describe our framework in the subsections that follow.

3.1. Model

3.1.1 EDA - Data Augmentation

Our training data had a highly imbalanced distribution of scores (Figure 1), with the majority of the images having scores within the 20-40 range. This proved to be a challenge to our models as most of the predictions will tend towards this range as well. While it is likely that the neural networks can learn these nuances if we train them longer, this will require a substantial amount of time and computational resources for hyperparameter tuning and architecture search, which we do not have the luxury of for this project.

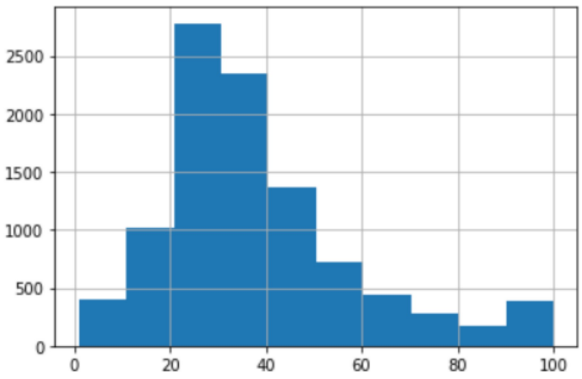
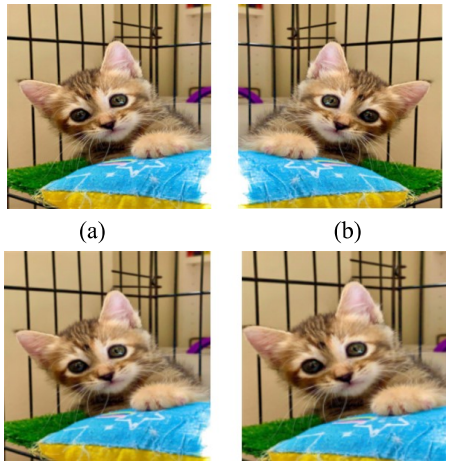


Figure 1: Distribution of Popularity Scores of Training Images

A more practical alternative is to consider oversampling by adding image transformations to augment our training data. After resizing the images such that they are all 224x224 pixels, we applied any one of the following transformations to images with scores within the 0-20 and 50-60 ranges: (1) Horizontal Flip, (2) Center Crop (to 90% of image size then resized to 224x224), or (3) Gaussian Blur ($\sigma = 0.3$). Figure 2 shows an illustration of these transformations. We believe that these transformations should be imperceptible enough

to not significantly affect how a human would rate the images. We concur, however, that more experimentation can be performed to find which other transformations work well for our data.



(a) (b)

(c) (d)

Figure 2: Image Transformations – (a) Resize, (b) Horizontal Flip, (c) Gaussian Blur, (d) Center Crop

Another important note to mention is that these transformations do not completely remove the score imbalance (otherwise, we risk the model focusing on learning the images themselves rather than the general properties of these images). Nonetheless, this should be able to help the model to better learn the features that result in the scores within these minority ranges as we show in our results.

3.1.2 Custom Deep Neural Networks for Regression

We leveraged select famous CNN architectures pre-trained on the ImageNet database and used transfer learning to help us save time on training, tuning which parameter layers/blocks to freeze and which to further train on our data. However, these models were trained for image classification tasks, not regression, and as such we modified the final layer of these models and concatenated these with feature embeddings from our metadata generated from feed-forward linear layers. We passed this concatenated set of feature embeddings to a few more feed-forward layers then to a final layer with a single neuron that predicts the popularity score. Being a regression model, we used MSE Loss as our metric for assessing quality of performance. A generalized diagram of our neural network architecture is shown in Figure 3.

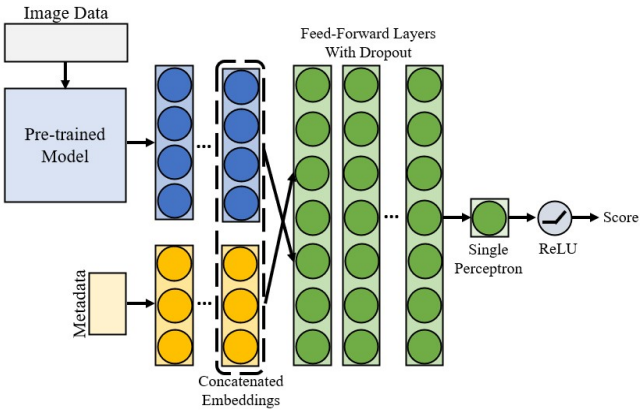


Figure 3: Our Deep Neural Network Architecture

3.2 Neural Network Visualization

Since we are using a neural network to imitate the behaviour of perceived popularity, it is important to understand what aspects of the image are looked at while deciding the popularity score. Additionally, our best-performing model does comparably well to a human and it would help build trust in the system to bring insights missed by humans or to validate initial perceptions. Visualisation is a strong tool to aid us in understanding the system as well as the data.

We specifically focus on saliency maps in this project. Saliency maps are of two types:

Gradient based: Gradients calculated on the pixel values of the image by back propagating the loss holding the model constant gives us information on how the score would change on varying the pixel slightly. This measures the sensitivity of the pixel to the final decision. Higher the sensitivity, the more crucial the pixel has been for the prediction. It should be noted that this works only in the case of whitebox models where we know the architecture and hence have access to gradients. We use simple gradients and GradCAM.

Perturbation based: In this method, we block a few pixels or patches of pixels to see how much the final prediction changes. Based on the change in score, the importance score is calculated for pixel or block of pixels. This works with black box models where we don't have access to model weights and architecture because all we need is the output for a given input. We use a method called RISE (Randomized Input Sampling for Explanation of Black-box Models) in this category.

3.2.1 Gradient-based Saliency maps

Vanilla Gradient descent is the simplest gradient-based Saliency map technique which primarily uses backpropagation for visualization [1]. After training our model on the images, we calculated the gradient of the mean squared error loss function with respect to the input image pixels. This results in a map of different features in the image with positive or negative values. For pixels with a high positive value, a small variation of these pixels would result

in a large variation in the resulting score of the input images. The resulting heatmaps were used to interpret features in the image that contributed to its popularity score and support our hypothesis on how popularity is calculated by the deep neural network. Figure 4(a) shows a pipeline of the Vanilla gradient descent Saliency map technique and Figure 4(b) shows a sample output of an image with its respective Saliency Map.

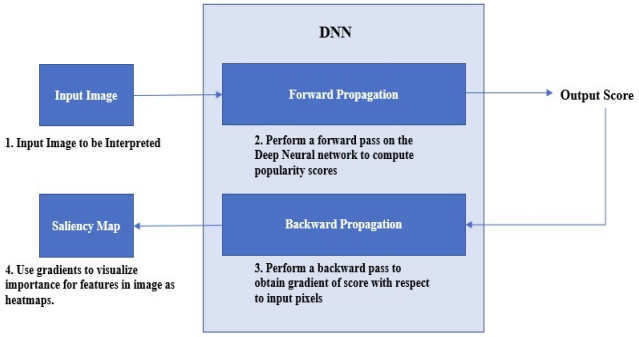


Figure 4(a): Vanilla Gradient for Saliency Map

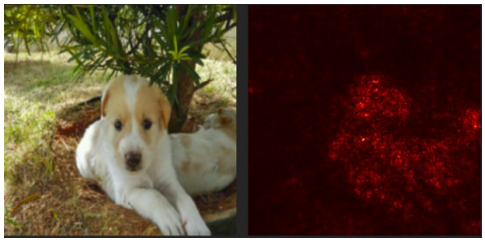


Figure 4(b): Image with its Saliency Map

3.2.2 Gradient-weighted Class Activation Mapping (GradCAM)

We used this gradient based technique by Selvarajan et al. [2] to visualise the higher order constructs captured in the convolutional neural layer of the deep neural network that predicts the popularity score of pets. Specifically, we computed attribution from the last CNN layer. As detailed in this paper, this method involved computing the gradients of the output popularity score with respect to the chosen convolution layer and averaged them over each of the 48 output channels, in this case. The average gradient for each channel is then multiplied by the layer activations and the results are summed over all channels. This final output can be viewed as a mask to the input, which is finally upsampled to the dimension of the input image. This upsampled GradCAM highlights the areas in the image that the model is sensitive to. In other words, changing the pixels in the bright regions of GradCAM will influence the output prediction strongly. Below is the pipeline for generating GradCAM representation.

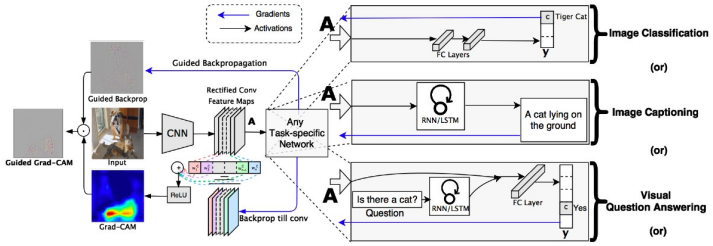


Figure 5: Pipeline of gradcam extending to specific tasks

Suiting the requirement, we used Mean Squared Error (MSE) of the predicted vs actual output, as the loss function in backpropagating for GradCAM. MSE captures the deviation of predicted pet's popularity score from actual, which is minimized to improve the accuracy of prediction.

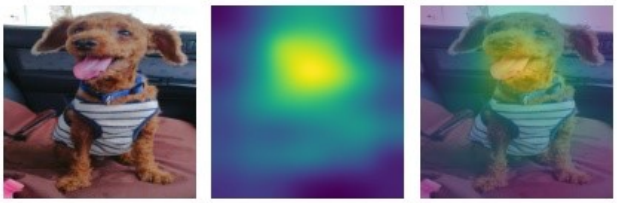


Figure 6: (a) Original image from dataset (b) saliency map generated by GradCAM highlights the important pixel cluster in neon green (c) GradCAM overlaid on the original image.

3.2.3 RISE

The method of RISE by Petsiuk et al. [3] describes how to get the saliency map by passing several masked images to the classifier. The masks are generated by stretching and interpolating a square grid where each block is visible with a probability of visibility p to create patches of blocks. The results of passing the masked images are further aggregated using a weighted sum as shown in the figure. The weights are well defined for a classification scenario in the paper as the output probability of the correct class.

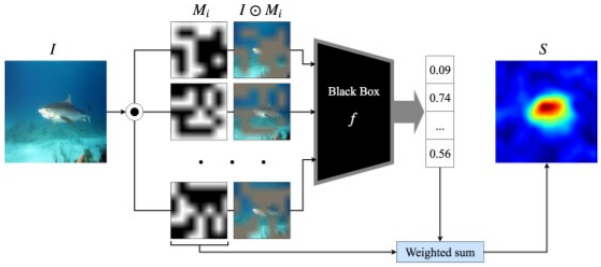


Figure 7: Overview of RISE method

The scenario at hand is however that of a continuous valued output and thus the measure that can be used as a weight is not so obvious. Directly using the final prediction score of popularity out of 100 could serve as a valid weight if the aim of the saliency map was to look at features causing high popularity. However, here we want to get an idea of which features are attributed to the score predicted by the model (and by extension people) as cute. Squared error indicating how much it deviates from the predicted score is a metric

that could be used. However, it is a measure of error and not a measure of contribution owing to its nature of being inversely related As shown in Figure 4(b) So we flip the obtained saliency map to account for this.

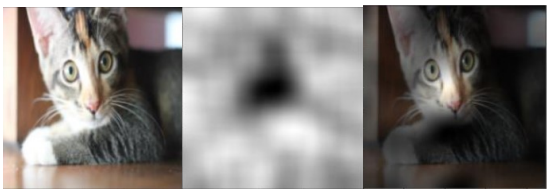


Figure 8: (a) Original image from dataset (b) saliency map generated by RISE highlighting the important pixels in black (c) The saliency map after flipping overlaid on the original image.

4. Experiments & Results

We have experimented on different architectures, algorithms, methods, and hyperparameters to obtain the most accurate score predictions given our project timeline. The best-performing model was used to generate the visualizations that highlight the most significant features in the image for the given scores. These experiments are discussed in depth in the following subsections.

4.1. Model Selection and Score Predictions

Given the complexity of our neural network architecture, there are many hyperparameter combinations that we tuned during our training procedure but focused on the following categories:

4.1.1. Pre-trained Model Architectures

We considered several well-known CNN architectures based on their size/complexity and their published performance on ImageNet classification. For our final experimentation, we opted for the following: ResNet [4], VGG [5], DenseNet [6], and EfficientNet [7]. These are considerably deep architectures; thus, we have decided to fix the pre-trained weights and biases for the first parameter layers/blocks to speed up training. We tested retraining the last 50-150 parameter layer weights and biases on our data, along our added layers. We found that training ~100 layers worked best – fewer layers and the model lacked the capacity to generate the abstractions to accurately predict the scores, while more layers made the model overfit easily to the training data.

4.1.2. Feed-Forward Layer Depth and Width

Since the image data has already been passed through the pre-trained architectures, we only modified the width (i.e., the number of neurons) of the final layer. On the other hand, the metadata only contains 12 features so we passed this to several feed-forward layers with an increasing number of neurons. We found that a balanced number of neurons or feature embeddings between image data and metadata worked best in terms of predictive performance rather than just putting more neurons on the image data, which shows that the addition of the metadata is important in predicting the scores. We passed these concatenated embeddings into another set of feed-forward layers with dropout layers (p = 0.2-0.5) in between. The final output which is a single perceptron is passed through a ReLU activation to clip negative predictions since the scores are bounded between 0-100.

4.1.3. Optimizer, Learning Rate, and Training Duration

Dealing with a regression task, we found that adaptive gradient algorithms such as Adam worked better than the standard stochastic gradient descent (SGD). To further reduce the risk of overfitting, we implemented the AdamW version of the algorithm, which includes a weight decay regularization factor and this has been shown by Loshchilov and Hutter to significantly improve Adam's generalization performance [8]. Furthermore, we used a learning rate scheduler that decreases the learning rate by a factor of 10 (starting from 1e-4) when the validation loss does not improve anymore. We found that the optimal state given our hyperparameters occurs shortly before the 10th epoch.

4.1.4. Quality of Score Predictions

Table 1 presents a comparison of our models' predictive performance (expressed as RMSE) based on the different pre-trained base CNN architectures we selected and whether data augmentation was applied.

Table 1: RMSE of Different Architectures on Score Prediction

	RMSE	
Pre-trained CNN	**w/ Augmentation**	**w/o Augmentation**
ResNet-152	11.3	19.0
VGG-19 (w/ Batch Norm)	12.7	19.5
DenseNet-161	**9.3**	**18.7**
EfficientNet-B7	10.4	20.0

The application of data augmentation proved to be effective, reducing our RMSE by half. This was mainly due to providing better predictions on our minority score ranges (e.g., prior to augmentation, our model predicted a score of 57 for a 97-scoring image; post-augmentation, the prediction became much closer at 91). Our modified neural net architecture using the DenseNet as the base pre-trained model performed best out of all the models that we tested, with an RMSE of 9.3. We are confident that this performance can be further improved through further experimentation and hyperparameter tuning, but we will leave this topic for further studies. For now, we believe that our model already produces more-than-sufficiently good results for the different visualization techniques that we selected to be applied – in our example earlier, a predicted score of 91 and a true score of 97 will both already be treated as very high scores.

4.2. Visualization

For our experiments we consider images from throughout the spectrum of high, average and low scores to visualise saliency maps.

Vanilla Gradient and GradCAM: Input images were resized to 224x224 and applied to the best performing model.

RISE: For our experiments, we use 500 masks, probability of visibility as 0.5 in each mask and the mask is obtained by stretching a grid of 8*8 to 224*224 to coincide with the image dimensions.

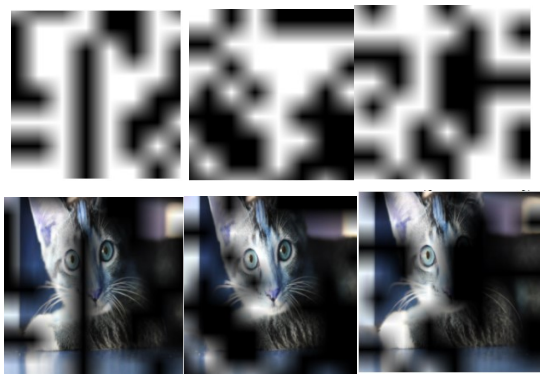


Figure 9: Randomly generated masks for RISE

We came up with several interesting hypotheses which could give us actionable insights into helping promote pets' popularity and adoption rate.

Hypothesis 1: Pictures with clear face and tongue sticking out have highest popularity scores

Below are the images of the pet followed by their RISE, GradCAM, and Vanilla Gradient activation maps of the most popular dogs.

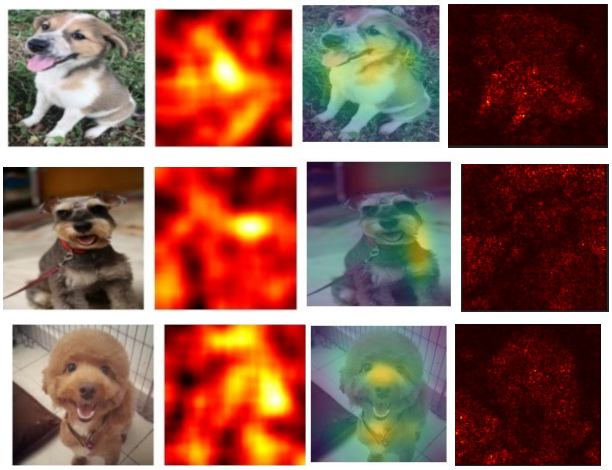


Figure 10: RISE, GradCAM & Vanilla Gradient saliency maps(columnwise) of popular pets for hypothesis 1

Examining a lot of pet images, we identified that the common features that contributed for the pet's image to be popular are:

a. Clear face with both eyes visible
b. Showing the tongue as a sign of playful energy
c. Posture of sitting on hind legs and
d. Moustache or whiskers in some cases

We also identified a few images as shown below, which satisfy the above features but still have low popularity scores. This infers that there are more aspects in a pet image with possibly strong interaction effects. We explored a few more hypotheses in this context further below.

Figure 11: GradCAM maps of unpopular pets for hypothesis 1

Conclusion: Failed to reject Hypothesis 1.

Hypothesis 2: Pictures with better color contrast of the background to the dog have higher popularity scores

Pictures with contrasting backgrounds have higher popularity. Below are a few examples with their saliency map produced by RISE, GradCAM and vanilla gradient activation maps.

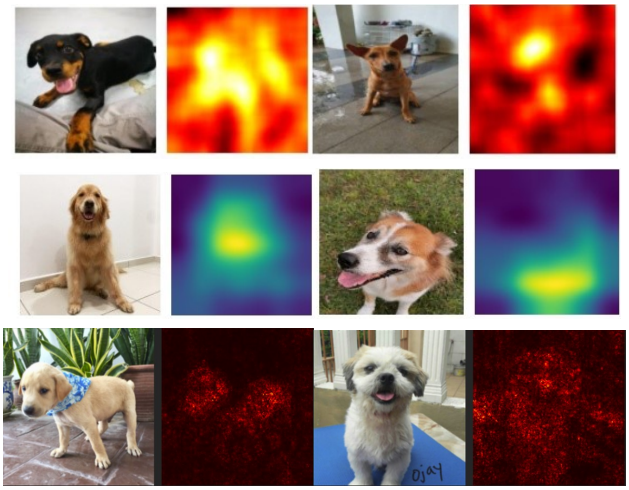
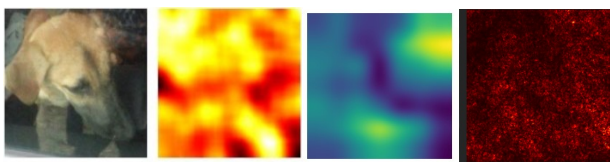


Figure 12 : RISE, GradCAM & Vanilla Gradient saliency maps (rowwise) of popular pets for hypothesis 2

Pictures with not-so-contrasting backgrounds are least popular. Their saliency maps produced by RISE, GradCAM, & Vanilla Gradient appear to be chaotic and the model struggles to even identify the pet, as shown in Figure 13.



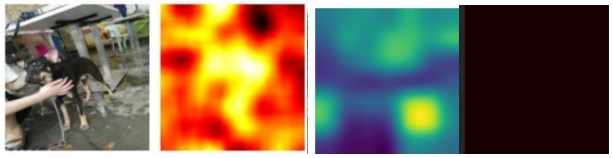


Figure 13: RISE , GradCAM & Vanilla Gradient saliency maps (rowwise) of least popular pets for hypothesis 2

We observe a more focused saliency map when the contrast is stark, and a more dispersed saliency map when the distinction of the pet and the background is not so clear. This contrast effect seems to play an important role in reducing the popularity score of the images referred in Figure 11.

Conclusion: Failed to reject Hypothesis 2.

Hypothesis 3: Pictures of pets decorated with accessories are popular

This, in addition to being a popular belief, is also something that could bring us actionable insights. If it turns out to be true, investing in such props for photos makes sense, and if not, it prevents us from the irrelevant addition of accessories.

Figure 14(a): RISE , Vanilla Gradient & GradCAM saliency maps(from top-left) of least popular pets not mapping accessories

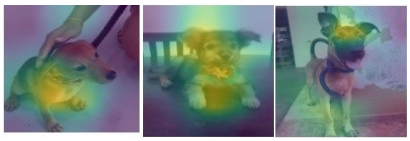


Figure 14(b): GradCAM saliency maps of least popular pets mapping accessories

From the above figures, we see that the saliency maps do not always highlight the accessories of popular pets as expected. Sometimes, even if the accessories are highlighted, the pets are not popular. Hence we do not have enough evidence to conclude whether accessories or the decorative setup play an important role in the popularity score.

Conclusion: Reject Hypothesis 3.

Hypothesis 4: Pictures with atypical postures and poses are unpopular

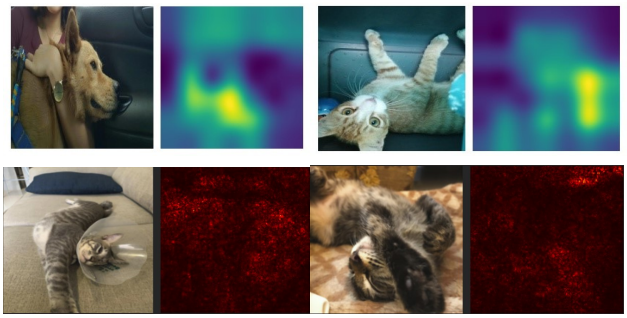


Figure 15: GradCAM & Vanilla Gradient saliency maps (from top-left) of pets with atypical postures

As highlighted in the above saliency map examples, the images with uncommon angles of the pets almost always have low popularity. These images also trouble the neural network in identifying the subject which reflects the low popularity score.

Conclusion: Failed to reject Hypothesis 4.

Hypothesis 5: Pictures with multiple pets or multiple images of same pet are popular

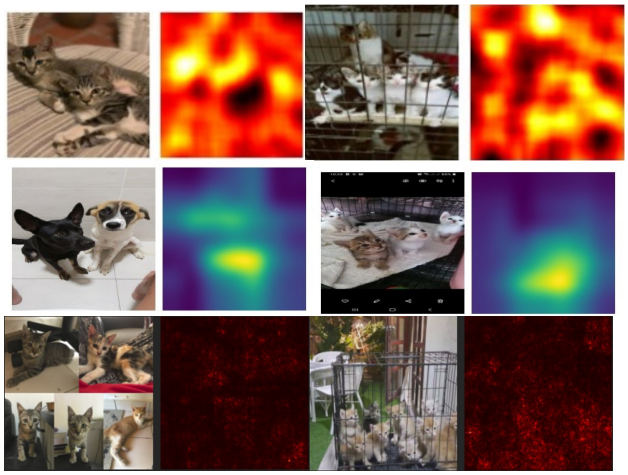


Figure 16: RISE , GradCAM & Vanilla Gradient saliency maps (rowwise) of least popular pets

Pictures with multiple images of the same pet or multiple pets have low popularity scores. From the images above, we can see that the saliency maps are highlighting several regions of the image that do not correspond to the pets.

Conclusion: Reject Hypothesis 5.

4.3. Limitations and Future Work

Due to limitations on time, computational resources, and available data, we leave the following explorations for future work:

- Training the Neural Networks on a Larger Hyperparameter and Architecture Search Space that better pick up the abstractions given by the image and metadata, leading to better popularity score predictions.
- The number of masks chosen for computing saliency maps for RISE was restricted to 500 due to computational constraints. There is a possibility that the map hasn't yet fully converged as measured by L2 distance between consecutive iterations.
- We analyzed the saliency maps visually and in a qualitative manner. While it gives us overall insights, quantitative metrics like insertion / deletion metrics could give us a different perspective about the quality of the saliency map itself.
- Sometimes, we see that the model which is expected to replicate human perception results in unexplainable saliency maps. This indicates that the model is picking up signals / patterns that are

not perceived consciously by humans which could further make it susceptible to adversarial attacks using imperceptible noises.

- Sometimes, we observe that the actual score of an image is way off from the one that the model predicted. For example, the image below satisfies all the postulated hypotheses of a good image with contrasting background, focused subject and, a classic posture, and consequently gives a much higher score than the actual. One of the possible explanations for this could be the recency bias. For the pets that were only recently posted on the adoption platform would not have gathered enough impressions to get a proper score.

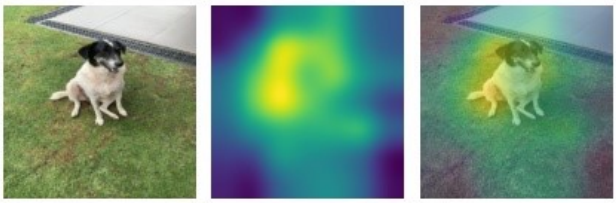


Figure 17: GradCAM image with actual score (24) & predicted score (43)

5. Conclusion

Given the understanding of some elements that contribute to the popularity of a pet, highlighting these features for new pet images could lead to high prediction scores on the platform which may ultimately result in faster and more successful adoptions. Below are some guidelines (not exhaustive) for capturing a pet image that better captivates human attention.

1. Maintain just one subject in the image;
2. Ensure a contrasting background against the pet;
3. Make the face clearly visible along with both eyes;
4. Avoid unusual poses of the pet; and
5. Bonus points if the dog sticks out its tongue indicating health and playful energy.

References

[1] Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." arXiv preprint arXiv:1312.6034 (2013).

[2] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization” arXiv:1610.02391 (2019).

[3] Petsiuk, Vitali, Abir Das, and Kate Saenko. "Rise: Randomized input sampling for explanation of black-box models." arXiv preprint arXiv:1806.07421 (2018).

[4] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. arXiv:1512.03385 (2016).

[5] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[6] Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. arXiv:1608.06993 (2017).

[7] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." International Conference on Machine Learning. PMLR. arXiv:1905.11946 (2019).

[8] Loshchilov, Ilya, and Frank Hutter. "Decoupled weight decay regularization." arXiv preprint arXiv:1711.05101 (2017)

Appendix A: The Machine Learning Reproducibility Checklist

For all **models** and **algorithms** presented, check if you include:

- A clear description of the mathematical setting, algorithm, and/or model. **Yes – we did not develop new models or algorithms but all models and methods used specifically reference the original study wherein these were first implemented.**
- A clear explanation of any assumptions. **Yes – these were specifically referenced in the report and in the code.**
- An analysis of the complexity (time, space, sample size) of any algorithm. **Not applicable – our project is an application/survey of established methods on a new domain.**

For any **theoretical claim**, check if you include:

- A clear statement of the claim. **Not applicable – our project is an application/survey of established methods on a new domain.**
- A complete proof of the claim. **Not applicable – our project is an application/survey of established methods on a new domain.**

For all **datasets** used, check if you include:

- The relevant statistics, such as number of examples. **Yes – these were specifically referenced in the code.**
- The details of train / validation / test splits. **Yes – these were specifically referenced in the code.**
- An explanation of any data that was excluded, and all pre-processing steps. **Yes – these were specifically referenced in the report and in the code.**
- A link to a downloadable version of the dataset or simulation environment. **Yes – the source was specifically referenced in the report and the link is included in the project code package.**
- For new data collected, a complete description of the data collection process, such as instructions to annotators and methods for quality control. **Not applicable – we did not collect new data.**

For all shared **code** related to this work, check if you include:

- Specification of dependencies. **Yes – all code includes libraries to import, data to load, and any other dependencies to run.**
- Training code. **Yes – these are included in the project code package.**
- Evaluation code. **Yes – these are included in the project code package.**
- (Pre-)trained model(s). **Yes – these were specifically referenced in the report and in the code. The best model states are also included in the project code package.**
- README file includes table of results accompanied by precise command to run to produce those results. **Yes – this is included in the project code package. There is a master readme in the root folder and specific readme documents in each visualization code folder. Each code/script can be run independently, provided that the required files are in the file location.**

For all reported **experimental results**, check if you include:

- The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results. **Yes – these were specifically referenced in the report and in the code.**
- The exact number of training and evaluation runs. **Yes – these were specifically referenced in the code.**
- A clear definition of the specific measure or statistics used to report results. **Yes – these were specifically referenced in the report and in the code. MSE/RMSE and visual/qualitative evaluation were used.**
- A description of results with central tendency (e.g. mean) & variation (e.g. error bars). **Yes – these were specifically referenced in the report and in the code, expressed in MSE/RMSE. Error bars are not applicable.**
- The average runtime for each result, or estimated energy cost. **Not applicable – this is highly dependent on the hyperparameter search space and the architecture considered, and the variance can be high.**
- A description of the computing infrastructure used. **Yes – these were specifically referenced in the code. GPU is used if available. The models can also be executed on CPU.**

Appendix B: Project Code and Content

Code package: https://tinyurl.com/mtcdbubw

Models trained: https://tinyurl.com/536a4at4

Output visualisations: https://tinyurl.com/2m9vpjfz