Evan Hollier, Paul Homuth

CS 4531 MW 5-7

6/3/2024

Pet Image Recognition with Keras

Introduction

Image recognition is an important concept in the world of Deep Learning. It has applications in many different fields such as medical diagnosis, facial recognition, image classification, fraud detection, and much more.[1] Recent advancements in deep learning and the new availability of larger pretrained models have made this task much more feasible to perform. Coupling this with the digital age and the large amounts of data available, these models are becoming more common. The goal of this project is to produce a model that can accept an image as input and classify what the image is. This was done in the context of pet images with one of 37 breed classifications as the output.

Dataset Description and Preparation

The data used for the work performed in this project was taken from the Oxford IIIT Pet Dataset.[2] This dataset consists of over seven thousand pet images of cats and dogs. There are 37 different categories of pets, 25 of which are dogs and 12 of which are cats, with roughly 200 images per category. The images within the dataset are non-standardized, meaning that each image has varying levels of light, angle of photo, distance to subject, age of the pet, and background of photo. This is one of the reasons the dataset was selected, as it was a realistic setting for the model to learn.

The initial reporting of the data set reported 7,349 photos, but this was found to be inconsistent with the actual data. Inside the dataset, there were an additional 12 dog images and an additional 32 cat images for a total of 2,403 cat images and 4,990 dog images. Even with slightly different classes sizes, the overall dataset was close enough in size per category to be treated as a balanced dataset. Once the totals were found, the next issue was damaged images. Various images within the dataset were found to have image corruption issues. These issues included missing sections of photos, warped colors, and mirrored areas in the image. These corrupted images needed to be removed from the model as they were not able to be saved into the correct format. The images in the dataset

were downloaded as JPG; reindexed to have correct labeling; and then split into training, validation, and testing subsets at a 60:20:20 ratio. Finally, the images were converted to JPEG format and submitted to the model.

Model Design

The first step in designing the model was to select a pre-trained model. The goal was to find a baseline model that would perform well on the data, so that a more complex and consistent model could be leveraged to improve model performance. To select a pre-trained model, four pre-trained models from Keras were selected for initial testing. Initial tests were performed to get a baseline loss and accuracy score to see which of the four selected for our model. The four models tested were VGG_16, ResNet50, EfficentNetB5, and Xception. These specific models were selected due to their reported performance on the ImageNet validation dataset and their varying complexities when looking at total parameters and depth of the models.[3] The selected pre-trained models were then fit to the preprocessed images from our dataset with a standard output layer. They were each trained for 20 epochs. The plots of the training and validation metrics are shown in the figure below.

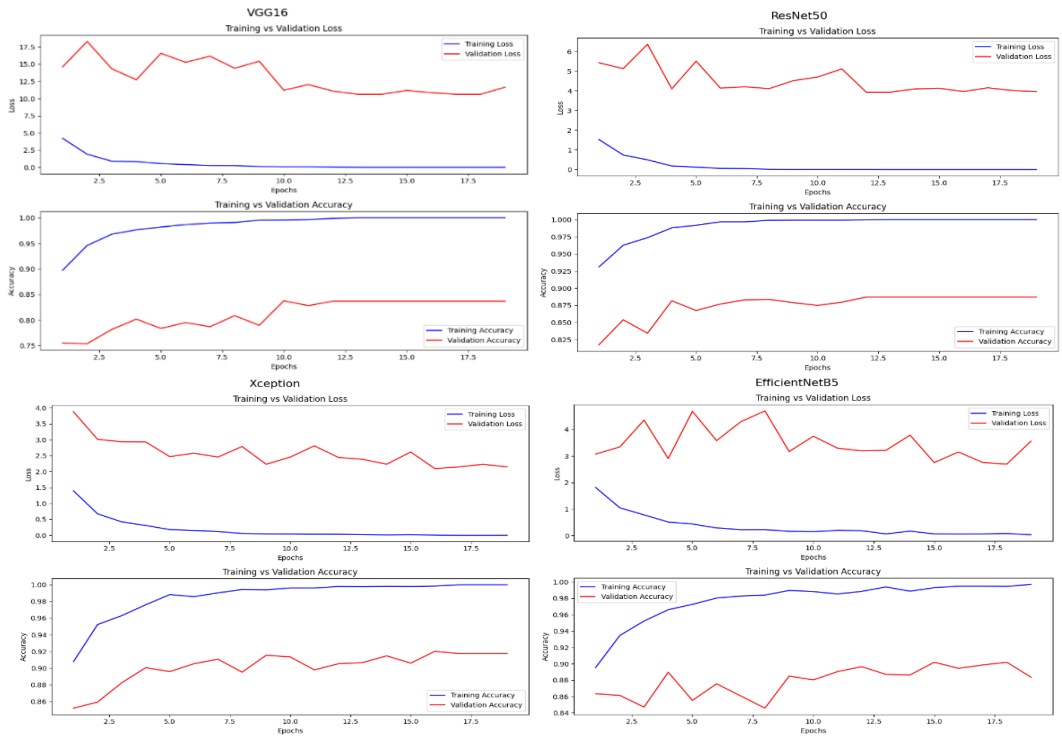

Figure 1: The training and validation metrics for baseline trials. From top left to bottom right, the models tried were VGG_16, ResNet50, Xception, and EfficentNetB5

Once the baseline training information was collected, it was seen that the ResNet50 and VGG_16 baselines performed worse than the other models. Their validation losses began to diverge from the training loss much earlier, and the validation accuracy never reached the heights of the other group of models. The other models, Xception and EfficentNetB5, both performed similarly well on the baseline runs. These models peaked near 90% accuracy and had losses that improved for some portion of the training.

To pick out a superior model between Xception and EfficentNetB5, the remaining baseline models were submitted to a grid search process on a simple custom layer to pick the best model and hyperparameters to continue with. This process focused on adjusting the optimizer of the model, the total number of neurons in the custom layer, and a final dropout to assist in preventing overfitting. The search iteratively tried RMSProp or Adam optimizers; 16, 32, 64, or 128 neurons in the last dense layer; and either no, 25%, or 50% dropout percentages when training the model. Thus, these 24 different combinations of hyperparameters on each model resulted in 48 models. Every model was trained for 10 epochs with early stopping using a patience of 4. After each trial in the grid search, the training and validation metrics were written to a JSON to review later. This was done so the grid search could be continued in batches, since many times the GPU would run out of memory. The validation metrics were then checked to find the best hyperparameters for both pre-trained models. This ended up being the RMSProp optimizer, 32 neurons in the dense layer, and no dropouts for the EfficentNetB5 model. The Xception model settled on the exact opposite hyperparameters: Adam optimizer, 128 neurons, and 50% dropouts. The best Xception model peaked at a validation accuracy of 91.89%, while the best EfficentNetB5 model peaked at 90.67%. In addition to having a higher accuracy score, the Xception models also ran noticeably faster than their EfficentNetB5 counterparts during the grid search.

The results of the grid search were used as a starting point for more complex adjustments to the custom layer. First, the loss function was switched from binary cross entropy to categorical cross entropy, since our data was multi-class. Surprisingly, categorical cross entropy was seen to produce worse validation accuracies, so binary cross entropy was selected as the loss function. Next, since the Xception model settled on the highest number of neurons in the grid search, it was decided that the overall complexity of the custom layers should increase. This was done gradually by adding more neurons to the first layer, and additional dense layers with fewer neurons than the first layer. This increase in complexity also allowed for the potential increase in overfitting on the training data, so batch normalization and regularization were tested to keep the model properly fit. Three forms of regularization were tried: Lasso, Ridge, and Elastic Net. However, the models including regularization underperformed. After trying many iterations, we honed in

on the selection of increased complexity, batch normalization, and no regularization. As complexity was increased, so was model performance, up to a point. Eventually, the performance dropped when the first layer had 512 neurons. Thus, it was elected that only an additional 256-neuron layer would be kept as it produced the best validation accuracy of 92.3%.

Finally, data augmentation and a decaying learning rate were explored. Data augmentation was designed to create one of 10,250 variations on the data passed through the data generator. Training the model on this augmented data made the model more robust against the variations in the dataset and saw an improvement validation accuracy up to 92.96%.

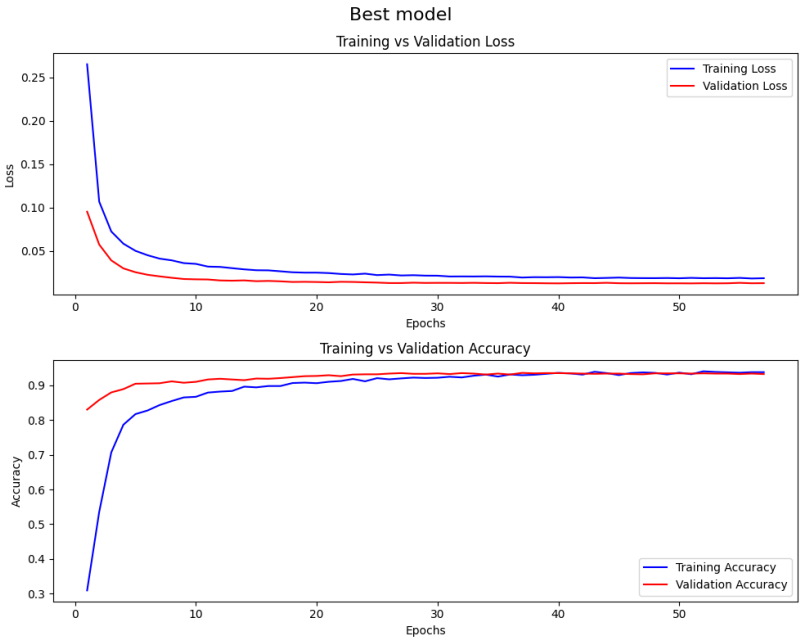

Figure 2: The training and validation history of the best model ran during the model design. This model added two dense layers with dropouts and normalization to the best Xception model found in grid search.

Following this, a variable learning rate was introduced to the model. The learning rate was relatively high for the first 10 epochs, then exponentially decayed for the remaining training. This allowed the model to adjust quickly at the start but more gradually toward the end of training. The model once again saw improvement from these changes, moving up to 93.64% validation accuracy.

Layer (type)	Output Shape	Param #
input_layer_79 (InputLayer)	(None, 255, 255, 3)	0
true_divide_39 (TrueDivide)	(None, 255, 255, 3)	0
subtract_39 (Subtract)	(None, 255, 255, 3)	0
xception (Functional)	(None, 8, 8, 2048)	20,861,480
flatten_39 (Flatten)	(None, 131072)	0
batch_normalization_250 (BatchNormalization)	(None, 131072)	524,288
dense_116 (Dense)	(None, 256)	33,554,688
batch_normalization_251 (BatchNormalization)	(None, 256)	1,024
dropout_78 (Dropout)	(None, 256)	0
dense_117 (Dense)	(None, 128)	32,896
batch_normalization_252 (BatchNormalization)	(None, 128)	512
dropout_79 (Dropout)	(None, 128)	0
dense_118 (Dense)	(None, 37)	4,773

Figure 3: The layer-by-layer breakdown of the best model found in the model design portion of the project.

Results and Discussion

Once the final model had been designed and trained, it was tested on the testing portion of the dataset. This dataset was comprised of 40 images from each category, giving a total of 1,480 photos for testing. These photos were passed into the model to be preprocessed by the Xception preprocessor and then evaluated for a classification. The model performed with an accuracy of 90.34% on this test dataset. These metrics showed that the model was able to find enough features within the images to be able to correctly classify most of the photos passed to it. This result was even more impressive since the image dataset was non-standardized. Finishing with a high testing accuracy meant that the techniques displayed in this work are feasible on real world images where there are variations on all major aspects of the photos.

Error analysis and confusing classes were investigated on the test dataset to gain a better understanding of what the model was misclassifying. The confusing classes were calculated by mapping the number of times an image from a category was misclassified, and what it was misclassified as. In total, 27 of the 37 classes had at least one mislabeled image. The classes with the most errors were Ragdoll cat, Russian Blue cat, and Staffordshire bull terrier dog, each having over ten images misclassified. The reason behind this was tied to their each having another class in the data set that the model kept misclassifying as. For example, 9 of the 12 errors in Ragdoll cat were when the model predicted Birman. The confusing classes were then combined in each direction to create

pairs of confusing classes. The main confused pairs were found to be Ragdoll cat and Birman cat being mislabeled as each other 16 times, Staffordshire Bull Terrier and American Pit Bull Terrier 11 times, and Egyptian Mau and Bengal 11 times. Samples of each of these three pairs were displayed as a sanity check. Lastly, the total errors for each class were quantified to calculate the potential improvement based on each class. This was done by taking the total error found in the testing set per category and multiplying it by the percentage that corresponds to the portion of the testing set made up by that category. Ragdoll and Russian Blue cats were found to offer the biggest potential improvement of 0.811% accuracy each since they both had a high error percentage of 30%.

Figure 4: A side-by-side comparison of the most confused classes found in the model. These three pairs each were confused over 10 times in the testing dataset.

Another interesting aspect of this dataset was the damaged photos. There were a handful of damaged photos present in the data that were withheld from the training sets. These 9 images were reintroduced as a unique dataset for the model to evaluate. The model accurately predicted 8 of the 9. These images were distorted in some way, with missing or corrupted areas of photos, but the model was still able to use what information it had to produce a strong prediction as to the class of the damaged image. This bodes well for the future of image classification models as these results show that a corrupted or damaged image can still be input. Other sources have also seen similar results when working with damaged images and even suggest Deep Learning models could be used for image repair and restoration.[4]

During the training process, the question of what loss function to use was addressed to find the best set up to train a model. Two obvious options were binary cross entropy and categorical cross entropy. Cross entropy is the measure of how similar the predicted probability of a class is compared to the actual label of the observation.[5] Binary cross entropy is designed for two classes while categorical cross entropy is designed for classifications with more than two categories. Naturally, we expected that categorical cross entropy would perform best on our model, but the opposite was found to be true. The training process more often reached a higher validation accuracy when the loss function was binary cross entropy. This surprising result is another example of how each model has its own set of best parameters, but it should be explored further in the future to better understand why the model performed better with this style of loss function.

Conclusion and Future Directions

The work performed in this project demonstrated the feasibility of image recognition using deep learning models by reaching 93.68% validation accuracy and 90.3% testing accuracy. Our designed model leveraged data augmentation, a pre-trained Xception model, two densely connected layers with batch normalization and dropouts, and a variable learning rate to achieve these metrics. This was an improvement over using the stock Xception model, which had a validation accuracy of only 91.75% after 20 epochs. The output of the model corresponded to one of 37 pet breeds within the data set. In the future, a double stream output, one for species and one for breed, could be added to the model to improve error detection. With the assumption that species would be a more reliable classification, the model could leverage the simpler problem of classifying the species to be used as error detection if the model outputs incompatible breed and species classes. Photos that were assigned incompatible breed and species can then be returned with an unknown breed but with the correctly identified species to still provide correct information. For further tests on our model, a Jupyter notebook is available on the project Github where personal pet images can be classified according to the 37 breeds of the original dataset.

References

1. *Image recognition applications: The basics and use cases*. Image Recognition Applications: The Basics And Use Cases. (n.d.). https://www.neurond.com/blog/image-recognition-applications
2. ml4py. (n.d.). *ML4PY/Dataset-IIIT-PET: Classification dataset for comparing cats and dogs images*. GitHub. https://github.com/ml4py/dataset-iiit-pet

3.
4. Team, K. (n.d.). *Keras Documentation: Keras applications*. https://keras.io/api/applications/
5. *Image restoration*. Papers With Code. (n.d.). https://paperswithcode.com/task/image-restoration
6. Nik. (2023, August 23). *Cross-entropy loss function in PYTORCH • datagy*. datagy. https://datagy.io/cross-entropy-loss-function-in-pytorch/

Supporting Information

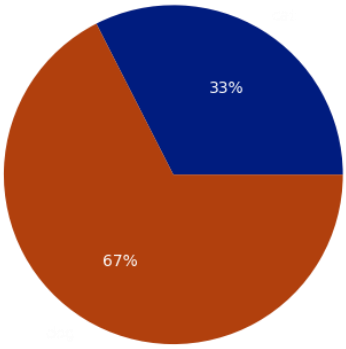

Figure S1: A pie chart depicting the ratio of cats to dogs in the dataset. In total, there were 2,403 cat images and 4,990 dog images found in the dataset.

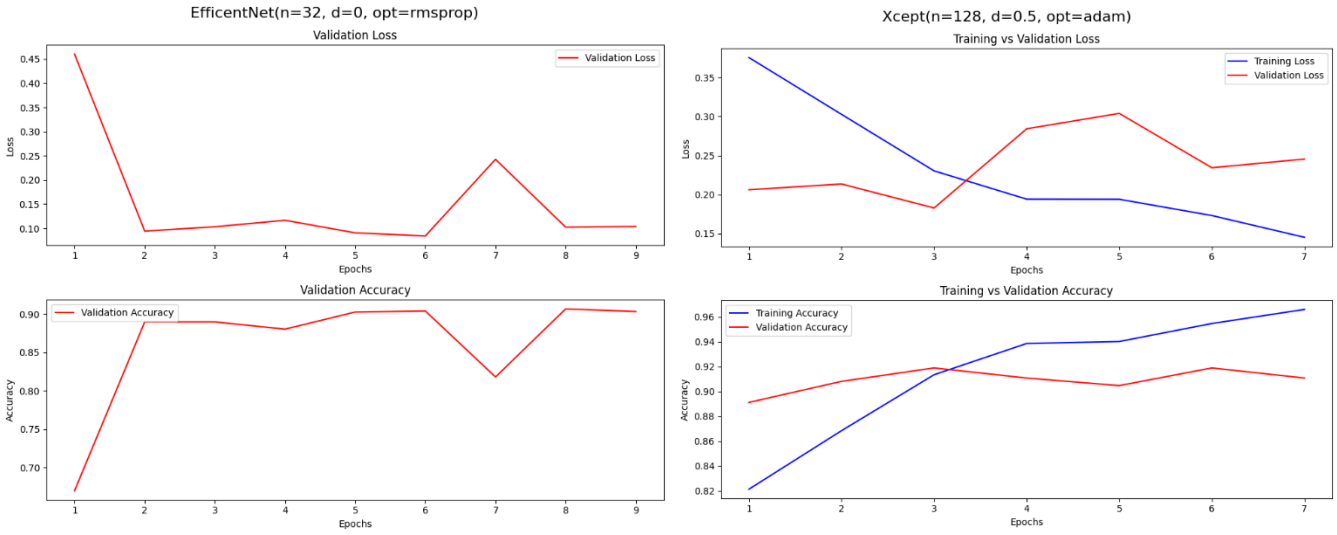

Figure S2: A side-by-side comparison of the best models found from the grid search. The best EfficentNet model (left) peaked at .9067 validation accuracy. The best Xception model (right) peaked at .9189 validation accuracy. The training data for EfficientNet model was lost during the saving to JSON process.

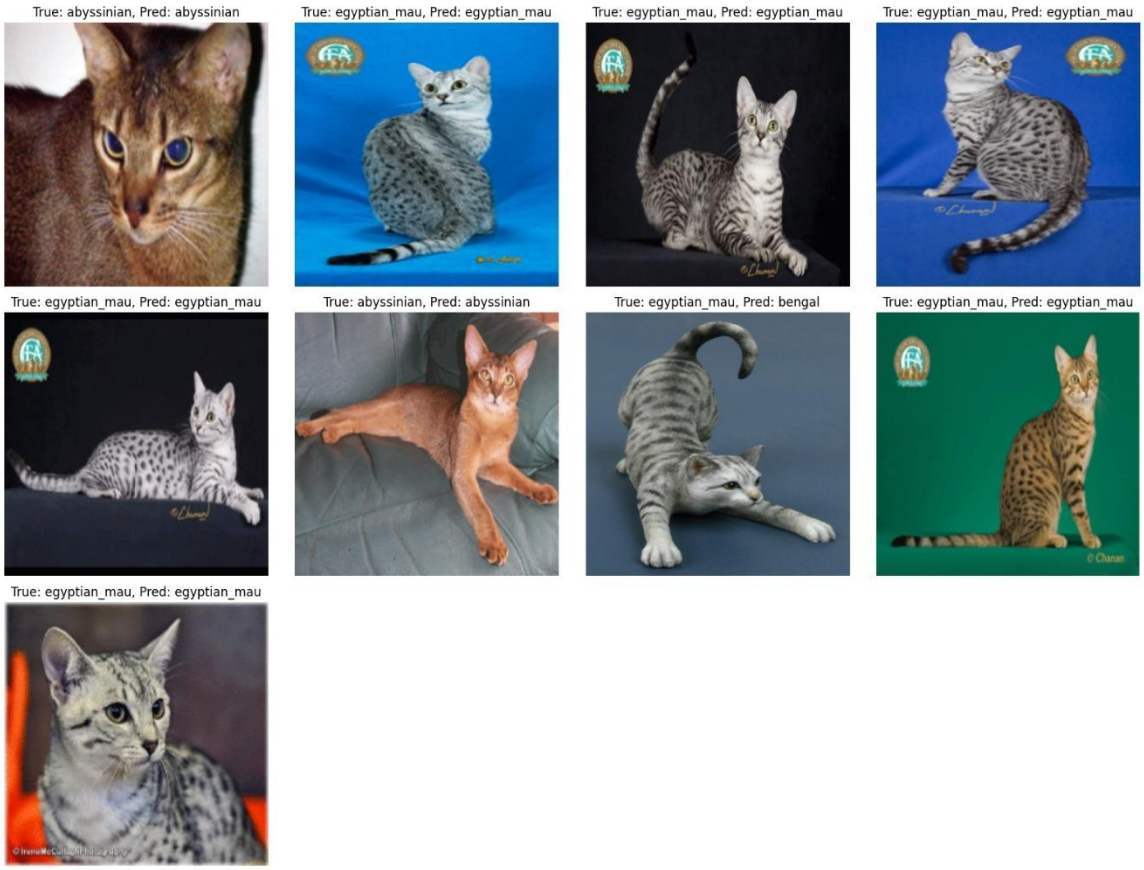

Figure S3: A look at how the model performed when predicting on the reported damaged photos in the data set. The model returned an accuracy of .88 when tested on the damaged dataset.

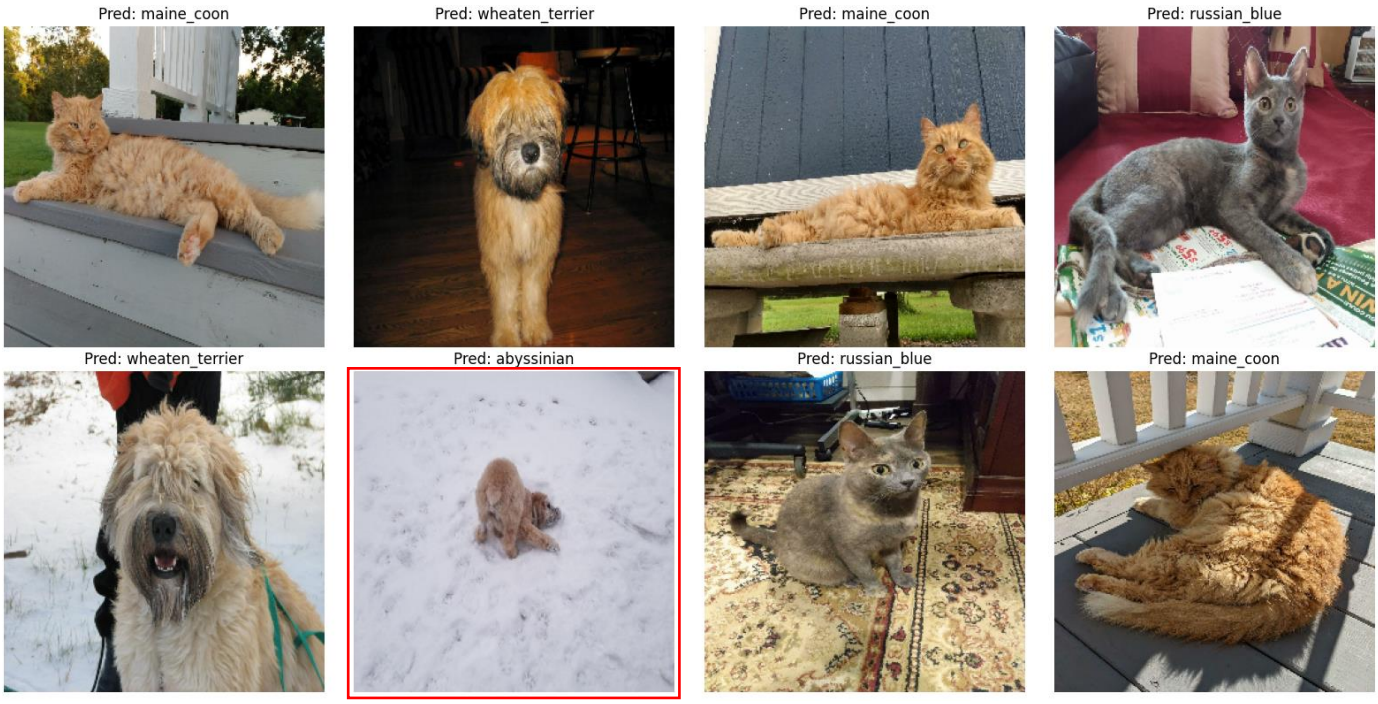

Figure S4: A trial run of the model performed on our own pets not seen in the dataset. The model correctly identified all but one photo highlighted in red.

File Catalog

Description of all files that can be found in the project github. [evanhollier/4531_Final (github.com)]

1. Baselines.ipynb – File containing the initial 4 baseline trials and the grid search process.
2. Final_project_scratch.ipynb – File containing the reindexing process of the dataset
3. Models.ipynb – File containing the different models tried post grid search
4. Running_eff_hist_list.json – JSON containing the training and validation histories of the EfficentNet grid search
5. Running_xcept_hists_list.json - JSON containing the training and validation histories of the Xception grid search
6. Scratch.ipynb – File containg code to convert and adjust images for data preparation
7. Test_your_data.ipynb – File to test own photos on the model
8. Testing.ipynb – Testing file to see metrics on test data and error analysis

9. Visuals.ipynb – scratch file for visuals