Ross Brancati
CS 589 Final Project Report
December 6, 2021

Introduction

Throughout this semester, we have learned a variety of powerful machine learning models of both the supervised and unsupervised types. We have also learned about both classification and regression, where classification involves predicting a class label and regression predicts the value of a continuous variable. In this project, we are provided both meta data and image data of animals that are up to be adopted and find their forever home. The goal of the project is to predict a samples "pawpularity" which is a score that describes how attractive a photo is to a potential adopter. Just for fun, here is an example of my dog's adoption photo (left) and a photo of her from a couple months ago (right). For reference, I changed her name from Sassy to Bailey when I first adopted.

The idea behind this project is that if the pawpularity of a photo can be accurately predicted, the photos with higher scores may give the animals a better chance of being adopted and avoid euthanizing them, which is important to many people because every good boy and good girl deserves a good home. So, Kaggle offered a competition to see who can generate the best predictions on the pawpularity of animals given two sets of data for each: some meta data and an image of the animal.

The first step is to develop a scheme of model selection and evaluation. Pawpularity scores are whole numbers ranging from 0-100, so this problem can be thought of as either regression or classification. Given that I am more comfortable with classification, I am going to move forward with this method. For reference, here is a histogram of the pawpularity scores from my exploratory data analysis.

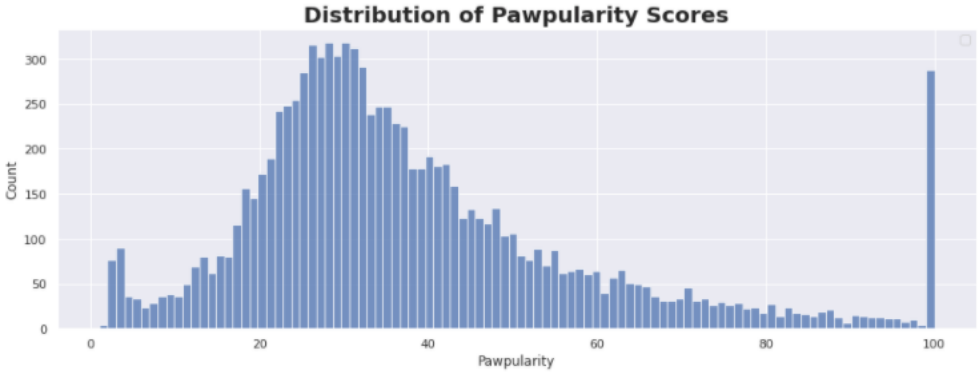


Figure 1: Distribution of parpularity scores. The scores can be considered as their own classes. Note that there are nearly 300 animals that have a pawpularity score of close to 300.

Part 1 – Model Validation

Model validation can be completed in a couple of different ways. In my models, I explored splitting the data into training and testing (and sometimes holding out validation data in the training set to tune hyperparameters) along with cross validation. The details of these methodologies are outlined in parts 3, 4, and 5 when I outlined the actual models I used. Depending on the dataset, either the meta data or image data, tuning the hyperparameters did not always improve the root mean square error (RMSE), which is the evaluation metric used in this competition. More specifically, it was easy and quick to train linear and ensemble models using only the meta data, so various hyperparameters could be tested to achieve the best results. Briefly, RMSE with different models and hyperparameters did not seem to improve, which is telling that a limitation is using only the meta data and ignoring the image data.

Part 2 – Handling of Training Data

Given the results of using only meta data (which typically resulted in RMSE of around 20), it is evident that the training data should include image data (and it is also a requirement of this project). As we learned in the class, images must be pre-processed and transformed into a format that is readable by computers. To do this, we can represent each pixel of an image by a number. In a previous assignment, we were tasked with classifying via a neural network and running a principal component analysis on handwritten digit data. The data is first transformed into matrices, where each entry into the matrix represents the grayscale intensity of a single pixel in the image. In this dataset, there are a few steps that must be followed to generate these matrices:

1. Images must be resized to the same dimensions because they are different sizes to begin with.
2. Images are transformed to arrays (either single or multidimensional). I used TensorFlow for the deep learning portion of the project which has functions that can convert images to tensors. In essence, images are being converted to numbers.

I conducted my analysis using Jupyter Notebooks and the online notebooks in Kaggle. The linear and ensemble models were quick to train, so I was able to complete these two directly in Jupyter Notebooks using my local CPU. However, the deep learning model I employed was much more complex and took much more computational power to train. Luckily, Kaggle has online GPUs to support deep learning and it is easy to download these notebooks at .ipynb files for local saving. On Kaggle, it is also very easy to save data and read it in, which was a nice addition and a good learning experience. So, when I developed the linear and ensemble models using the meta data, data was downloaded locally, and I did not pre-process it. The only possible pre-processing of meta data that I could envision is running dimensionality reduction using principal component analysis or another method, but time-constraints prevented that. However, when I used the image data and a deep learning network, the images had to be preprocessed. Appendix C of this report includes the deep learning notebook which has the code to process the images, so they are ready for TensorFlow. In brief, there is a function that reads an image file, decodes the image into red, green, and blue (RGB) values, casts the image to a float, then resizes and returns the resulting tensor. Each image is looped over, and the tensors are saved in a data matrix, namely X. This was done for both the training and the testing images, although it should be noted that the testing images are just random noise. This data matrix X was split into testing and training (90% train and 10% test) to evaluate performance of the network via RMSE value. As noted in the project guidelines, test image data was not pre-processed because that would destroy the point of this competition.

Lastly, I tried to combine the meta and image data using inspiration from Gessert et al. 2019 [2], and train/test this data on a linear regression model, but results were poor. If time permitted, I would have also tried a deep learning approach with the combined image and meta data. In this process, I converted each image into an array of grayscale (90x90x1) or RGB (50x50x3) pixel values, flattened each array into a vector, and combined the vector with the meta data each image. This resulted in two large datasets, one using grayscale pixel intensities and the other using RGB pixel values.

Part 3 - Linear Model

Initially, I attempted the approach combining image and meta data, but the results were not great using a linear regression model. I split the training and testing data into 80% train and 20% test, fit the linear regression model with no regularization, and calculated both R^2 and RMSE on the test set, which were astronomically horrible. This is not surprising because image processing with linear models is not a great approach due to the large number of dimensions and parameters that must be learned with image data, which is why deep learning is often the better approach for this application.

Table 1: Astronomically poor test set RMSE and R^2 values attempting linear regression with combined image and meta data.

RMSE and R Squared Values	
Metric	**Value**
RMSE	27042.00
R Squared	-63.81

Given these results, I decide to save image data for the deep learning question, and only used meta data for the linear model. When picking a linear model, it is important to look at the labels of the data points. In this case, one could approach this as either a regression or a classification problem. However, using logistic regression on this dataset brings the challenge of needing to perform 101 one versus all classifications due to the binary nature of logistic regression. For example, you would need to treat a pawpularity score of 0 as its own class, while pawuplarity scores 1, 2, 3 … 100 are all one class, then treat 1 as its own class with the other class being 0, 2, 3 … 100 and so on. I thought that linear regression was a much better approach which would treat the pawpularity score as a continuous distribution. I performed linear regression, then also attempted to regularize with both Ridge and Lasso regression independently. Ridge regression uses the L2 regularization function and Lasso regression uses the L1 regularization function. I will admit to attempting a logistic regression classification just for fun, but the performance on the test set was very poor. Linear regression RMSE scores were decent compared to other scores I was seeing on the leaderboard, but this was only performed on meta data whereas the top scores utilized deep learning on the image data. To tune the hyperparameters of the Ridge and Lasso models, I looped over various values of alpha, fit the model, predicted the test scores, and computed RMSE. The table below shows the best RMSE scores achieved, the value of alpha, and the model I used. I should also note the 80-20 train-test split in these models and the use of 5-fold cross validation evaluated by RMSE. When cross-validating for the Lasso and Ridge models, the best alpha was used. The notebook for the linear models can be found in Appendix A of this report.

Table 2: Test set RMSE for linear regression models with ridge and lasso regression. Lasso regression performed the best when evaluating by RMSE.

Linear Model Test Scores		
Model	**Alpha**	**RMSE**
Linear		19.92768
Ridge	0.001	19.92768
Lasso	0.001	19.92707

Part 4 - Ensemble Models:

A few different approaches for ensemble models were attempted. For reasons mentioned above, I only used the meta data with the ensemble models for fear of overfitting and poor RMSE. There were four models in total that I tried. I split the data into 80% train and 20% test for each model and tried different hyperparameters.

Model 1: Random Forest and Support Vector Machine (SVM) with a Logistic Regression final estimator. Random forest includes the number of estimators (n_estimators) hyperparameter, which I tried tuning by using estimator values between 10 and 50. Interestingly, the RMSE did not improve by altering the number of estimators, so to keep the model more generalizable, I went with 10 estimators for this hyperparameter. The SVM's kernel can impact the results of the model, so I ended up trying the three kernel's which were linear, RBF, and polynomial (degree=3). I used Logistic Regression as the final estimator here and tried no penalty and an L2 penalty. The best RMSE was with the model that had n_estimators = 10 for the Random Forest, an RBF kernel for the SVM, and no penalty on the Logistic Regression classifier. I also tried using Linear Regression as the final estimator, but ensemble models require classification algorithms, so this threw an error in python.

Model 2: In the second model, I dropped the Logistic Regression final estimator to try to make the model more generalizable to unsee test data, so this model only included a Random Forest and SVM as the final estimator. Once again, the number of estimators did not seem to improve RMSE. However, I found that the linear kernel SVM generated a better RMSE than the RBF kernel. So, the final hyperparameters for this model were n_estimators = 10 for the Random Forest and a linear kernel for the final SVM classifier. This model outperformed the previous model as well in terms of RMSE.

Model 3: This model was very similar to the previous one, except I added the Logistic Regression classifier back into the base estimators. So, this model consisted of Random Forest and Logistic Regression Base Classifiers with SVM as the final estimator. The results were the same as model 2.

Model 4: I wanted to try a model without Random Forest as it is very powerful, so in this model I used KNN and Decision Tree as the base models with a linear kernel SVM as the final estimator. I started with 20 neighbors and a depth of 10 for the decision tree, which yielded very similar results to the previous models. Next, I increased the number of neighbors and depth to 40 and 50, respectively, but the results were very similar. Lastly, I boosted the number of neighbors all the way up to 1000 and kept the depth of the decision tree at 25, which resulted in a similar RMSE.

Model 5: The last model was a simple Random Forest with number of estimators set to 50 because I wanted to test if just one model resulted in better RMSE, but it did not. This showed me that ensemble models are more accurate at classifying this meta data.

Table 3: Ensemble model RMSE scores. Model 3 performed the best in terms of RMSE on the held out test data.

Ensemble Model Test Scores	
Model	**RMSE**
Model 1	23.18305
Model 2	21.69116
Model 3	21.69111
Model 4	21.69715
Model 5	23.87247

Part 5 – Deep Learning

Deep learning is extremely powerful for image processing. At the time of this project, the leading RMSE score of the pawpularity competition was 17.65210. Since the meta data approaches outlined above did not achieve this high of a score, I knew that the leading score must be using image data which was probably analyzed using deep learning. There are virtually endless configurations of neural networks that can be used as they can be customized easily with python packages such as TensorFlow, which is what I used to build my model. Given the large number of possible configurations, I did background research on pre-established convolutional neural networks and found SqueezeNet, which was found to achieve the same Accuracy as AlexNet in terms of classifying 1000s of objects but is much smaller than the AlexNet. I thought this may work well for classifying pawpularity, so I implemented it as my deep learning model. For the scope of this project, a convolutional neural network (CNN) with multiple layers utilizing backpropagation seems sufficient. CNNs are typically successful with image classification. Using Iandola et al. 2017 [1], I was able to implement SqueezeNet using TensorFlow and achieved a decent RMSE, which I felt sufficed for this project scope. I believe you could combine other models used with deep learning, but I think that the CNN approach is best fit for image processing and was well suited for this competition. The loss and RMSE for each epoch are plotted below.

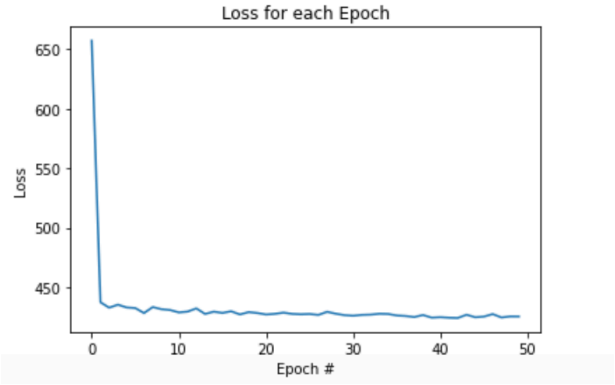
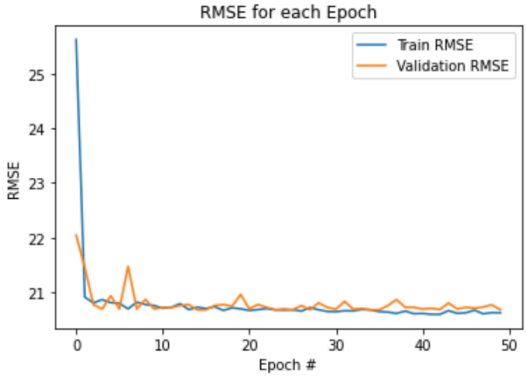


Figure 2: Loss for each epoch of SqueezeNet training.

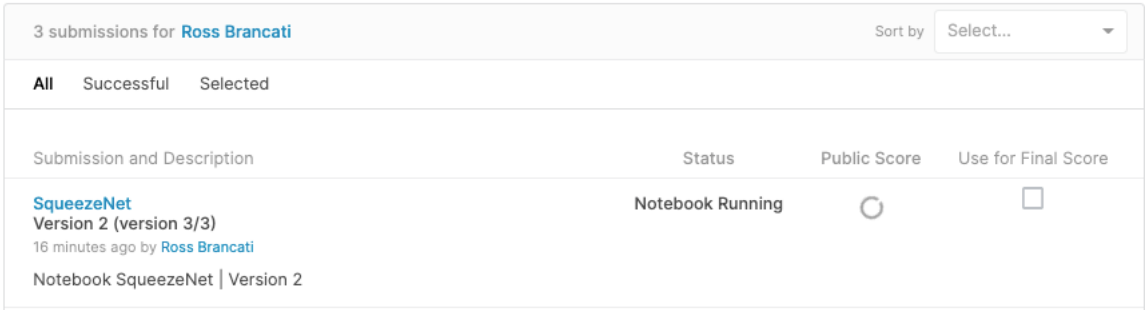


RMSE for training and validation data for each epoch of SqueezeNet training. From these plots, I could have ended the training around epoch 10 and most likely would have achieved similar results to using 50 epochs.

Table 4: Validation RMSE for SqueezeNet convolutional neural network. SqueezeNet outperformed all previous models in terms of RMSE when image data was used for training and testing.

Deep Learning Validation Score	
Model	**RMSE**
SqueezeNet	20.7191

Part 6 – Submission to Kaggle Competition



Submission to Kaggle competition.

References

[1] Iandola, FN., Han, S., Moskewicz, MW., Ashraf, K., Dally, W. and Kuetzer, K. *SqueezeNet: AlexNet Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size*. 2016. arXiv:1602.07360.

[2] Gessert, N., Nielsen, M., Shaikh, M., Werner, R. and Schlaefer, A. *Skin lesion classification using ensembles of multi-resolution EfficientNets with meta data.* MethodsX. 7: 100864 (2020).

[3] Teboul, A. *Tutorial Part 3: CNN Image Modeling 1*. https://www.kaggle.com/alexteboul/tutorial part-3-cnn-image-modeling-1

Appendix A – Linear Model Notebook

This approach is only using the meta data with linear regression because the performance with the image data was very poor.

Models Attempted in this Notebook:

1. Linear Regression
2. Ridge Regression
3. Lasso Regression
4. Logistic Regression

In [53]:
```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

import math
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline

import warnings # supress warnings
warnings.filterwarnings('ignore')
```

In [54]:
```
#import the data
meta_data = pd.read_csv('petfinder-pawpularity-score/train.csv')
```

In [55]:
```
#show data header
meta_data.head()
```

Out[55]:

	Id	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur	Pawpularity
0	0007de18844b0dbbb5e1f607da0606e0	0	1	1	1	0	0	1	0	0	0	0	0	63
1	0009c66b9439883ba2750fb825e1d7db	0	1	1	0	0	0	0	0	0	0	0	0	42
2	0013fd999caf9a3efe1352ca1b0d937e	0	1	1	1	0	0	0	0	1	1	0	0	28
3	0018df346ac9c1d8413cfcc888ca8246	0	1	1	1	0	0	0	0	0	0	0	0	15
4	001dc955e10590d3ca4673f034feeef2	0	0	0	1	0	0	1	0	0	0	0	0	72

In [56]:
```
#split the data into train and test split so that we can still test our hyperparameters
meta_train, meta_test = train_test_split(meta_data, train_size=0.8, test_size=0.2, random_state=10)
```

In [57]:
```
#split data frames into X_train, y_train, X_test, y_test
#assign training labels
y_train = meta_train.pop('Pawpularity')
#assign training data
X_train = meta_train
#remove Id from training data
X_train.pop('Id')

#assign testing labels
y_test = meta_test.pop('Pawpularity')
#assign testing data
X_test = meta_test
#remove Ids from testing data
X_test.pop('Id')
```

Out[57]:
```
9161    ecbb48fc9d345f6e2b03deaf8f1645f0
9695    fa4a3d69e1e0e21b62bb33538bc54e61
9033    e97d059f75a50e9c9805b0dba4d0d84e
4617    76420f02afab76d2a6eab95efc816347
4220    6bb7f0653725b30118199fd763945713
                      ...
6500    a8028d608d5a1916c5482616e5838b6c
8934    e6f6665772e5e67240d46d899e01ad78
2756    4708f6747261af730735ff0272cfc73e
3508    5a7fa7cfeb8d5e32116574e6be7ecb6a
3923    6411e12dd43ef887ac45984c01ebf850
Name: Id, Length: 1983, dtype: object
```

In [58]:
```
#quick look at training data
X_train.head()
```

Out[58]:

	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur
6916	0	1	1	1	0	0	0	0	0	0	0	0
5837	0	1	1	1	0	0	0	0	0	0	0	0
2600	0	1	1	1	0	0	0	0	0	0	0	0
2167	0	0	0	0	1	0	1	0	0	1	0	0
7026	0	1	1	1	0	0	0	0	0	0	0	0

In [59]:
```
#quick look at training labels
y_train.head()
```

Out[59]:
```
6916     6
5837    65
2600    23
2167    20
7026     2
Name: Pawpularity, dtype: int64
```

In [61]:
```
#Create a linear regression model
lm = LinearRegression()
lm.fit(X_train, y_train)
```

Out[61]:
```
LinearRegression()
```

In [62]:
```
#above cell started at 7:11PM and ended around 7:20
y_pred_lin_reg = lm.predict(X_test)

#calculate R squared value
r2 = sklearn.metrics.r2_score(y_test, y_pred_lin_reg)
print(r2)
#calculate mean absolute error
neg_mean_abs_err = sklearn.metrics.mean_absolute_error(y_test, y_pred_lin_reg)
print(neg_mean_abs_err)
#calculate root mean square error
lin_reg_RMSE = math.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred_lin_reg))
lin_reg_RMSE
```

```
0.0020122486417494256
15.20007403789931
```

Out[62]:
```
20.405777172912124
```

Attmepting Cross Validation Scoring

In [63]:
```
#get RMSE score using cross validation
lm_scores = cross_val_score(lm, X_train, y_train, scoring='neg_root_mean_squared_error', cv=5)
lm_scores
```

Out[63]:
```
array([-20.82178305, -19.92767999, -20.75745571, -21.06901215,
       -20.66647169])
```

Lets also try L2 regularization using ridge regression, which minimizes the loss function of linear least squares. The following tests multiple alpha values, fitting the model for each value of alpha as it runs over the for loop

In [50]:
```
#list of alphas to test
alpha = [0.001, 0.1, 1, 2, 5]

#loop over each value of alpha, fit model, print results
for i in alpha:
    ridge = Ridge(alpha=i)
    ridge.fit(X_train, y_train)
    y_pred_ridge = ridge.predict(X_test)
    print('-'*20)
    print('Alpha: '+str(i))
    r2_ridge = sklearn.metrics.r2_score(y_test, y_pred_ridge)
    print('R^2: '+str(r2_ridge))
    neg_mean_abs_err_ridge = sklearn.metrics.mean_absolute_error(y_test, y_pred_ridge)
    print('Neg. Mean Abs. Error: '+str(neg_mean_abs_err_ridge))
    RMSE_ridge = math.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred_ridge))
    print('RMSE: '+str(RMSE_ridge))
```

```
--------------------
Alpha: 0.001
R^2: 0.002012237843247666
Neg. Mean Abs. Error: 15.200074073392154
RMSE: 20.405777283310183
--------------------
Alpha: 0.1
R^2: 0.00201116907894594
Neg. Mean Abs. Error: 15.200077584783395
RMSE: 20.405788209777068
--------------------
Alpha: 1
R^2: 0.0020014791683298228
Neg. Mean Abs. Error: 15.200109289408063
RMSE: 20.405887273903698
--------------------
Alpha: 2
R^2: 0.0019907679298079017
Neg. Mean Abs. Error: 15.200144061614234
RMSE: 20.405996778945447
--------------------
Alpha: 5
R^2: 0.0019589849859891206
Neg. Mean Abs. Error: 15.20024555990199
RMSE: 20.406321704539742
```

Lets try fitting the Ridge model with the best alpha from above and running the cross validation score function

In [64]:
```
#calculating RMSE on ridge model using cross validation (L2 regularization)
ridge = Ridge(alpha=0.001)
ridge_scores = cross_val_score(ridge, X_train, y_train, scoring='neg_root_mean_squared_error', cv=5)
ridge_scores
```

Out[64]:
```
array([-20.82178284, -19.92767993, -20.75745556, -21.06901208,
       -20.66647163])
```

Lets also try the lasso

In [52]:
```
#list of alphas to test
alpha = [0.001, 0.1, 1, 2, 5]

#loop over each value of alpha, fit model, print results
for i in alpha:
    lasso = Lasso(alpha=i)
    lasso.fit(X_train, y_train)
    y_pred_lasso = lasso.predict(X_test)
    print('-'*20)
    print('Alpha: '+str(i))
    r2_lasso = sklearn.metrics.r2_score(y_test, y_pred_lasso)
    print('R^2: '+str(r2_lasso))
    neg_mean_abs_err_lasso = sklearn.metrics.mean_absolute_error(y_test, y_pred_lasso)
    print('Neg. Mean Abs. Error: '+str(neg_mean_abs_err_lasso))
    RMSE_lasso = math.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred_lasso))
    print('RMSE: '+str(RMSE_lasso))
```

```
--------------------
Alpha: 0.001
R^2: 0.0020368144345211947
Neg. Mean Abs. Error: 15.199691709677248
RMSE: 20.40552602394885
--------------------
Alpha: 0.1
R^2: -3.27384141818321e-05
Neg. Mean Abs. Error: 15.206452322861185
RMSE: 20.426673318625596
--------------------
Alpha: 1
R^2: -0.0002483024606003692
Neg. Mean Abs. Error: 15.205877083472856
RMSE: 20.428874756100438
--------------------
Alpha: 2
R^2: -0.0002483024606003692
Neg. Mean Abs. Error: 15.205877083472856
RMSE: 20.428874756100438
--------------------
Alpha: 5
R^2: -0.0002483024606003692
Neg. Mean Abs. Error: 15.205877083472856
RMSE: 20.428874756100438
```

Compute cross validation scores on lasso model

In [65]:
```
#calculating RMSE on lasso model using cross validation (L1 regularization)
lasso = Lasso(alpha=0.001)
lasso_scores = cross_val_score(lasso, X_train, y_train, scoring='neg_root_mean_squared_error', cv=5)
lasso_scores
```

Out[65]:
```
array([-20.82056275, -19.92707415, -20.75680592, -21.06863568,
       -20.66599196])
```

Logistic Regression Attempt (just for fun > I did not think it would perform well due to the large number of labels

In [30]:
```
#build and fit model
lr = LogisticRegression(random_state=0)
lr.fit(X_train, y_train)
```

Out[30]:
```
LogisticRegression(random_state=0)
```

In [31]:
```
y_pred = lr.predict(X_test)
```

In [36]:
```
f1score = sklearn.metrics.f1_score(y_test, y_pred, average='weighted')
f1score
```

Out[36]:
```
0.012483017046184233
```

Appendix B – Ensemble Models Notebook

Ensemble Model Approach

In [160]:
```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

import math
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.pipeline import make_pipeline
from sklearn import metrics

import warnings # supress warnings
warnings.filterwarnings('ignore')
```

In [78]:
```
#import the data
meta_data = pd.read_csv('petfinder-pawpularity-score/train.csv')
```

In [79]:
```
meta_data.head()
```

Out[79]:

	Id	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur	Pawpularity
0	0007de18844b0dbbb5e1f607da0606e0	0	1	1	1	0	0	1	0	0	0	0	0	63
1	0009c66b9439883ba2750fb825e1d7db	0	1	1	0	0	0	0	0	0	0	0	0	42
2	0013fd999caf9a3efe1352ca1b0d937e	0	1	1	1	0	0	0	0	1	1	0	0	28
3	0018df346ac9c1d8413cfcc888ca8246	0	1	1	1	0	0	0	0	0	0	0	0	15
4	001dc955e10590d3ca4673f034feeef2	0	0	0	1	0	0	1	0	0	0	0	0	72

In [80]:
```
#split the data into train and test split so that we can still test our hyperparameters
meta_train, meta_test = train_test_split(meta_data, train_size=0.8, test_size=0.2, random_state=10)
```

In [81]:
```
#split data frames into X_train, y_train, X_test, y_test
#assign training labels
y_train = meta_train.pop('Pawpularity')
#assign training data
X_train = meta_train
#remove Ids from training data
X_train.pop('Id')

#assign testing labels
y_test = meta_test.pop('Pawpularity')
#assign testing data
X_test = meta_test
#remove Ids from testing data
X_test.pop('Id')
```

Out[81]:
```
9161    ecbb48fc9d345f6e2b03deaf8f1645f0
9695    fa4a3d69e1e0e21b62bb33538bc54e61
9033    e97d059f75a50e9c9805b0dba4d0d84e
4617    76420f02afab76d2a6eab95efc816347
4220    6bb7f0653725b30118199fd763945713
                      ...
6500    a8028d608d5a1916c5482616e5838b6c
8934    e6f6665772e5e67240d46d899e01ad78
2756    4708f6747261af730735ff0272cfc73e
3508    5a7fa7cfeb8d5e32116574e6be7ecb6a
3923    6411e12dd43ef887ac45984c01ebf850
Name: Id, Length: 1983, dtype: object
```

In [82]:
```
#quick view of training data
X_train.head()
```

Out[82]:

	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur
6916	0	1	1	1	0	0	0	0	0	0	0	0
5837	0	1	1	1	0	0	0	0	0	0	0	0
2600	0	1	1	1	0	0	0	0	0	0	0	0
2167	0	0	0	0	1	0	1	0	0	1	0	0
7026	0	1	1	1	0	0	0	0	0	0	0	0

In [83]:
```
#quick view of testing data
y_train.head()
```

Out[83]:
```
6916     6
5837    65
2600    23
2167    20
7026     2
Name: Pawpularity, dtype: int64
```

Ensemble Classifier 1:

1. Random Forest
2. SVM with RBF Kernel
3. Logistic Regression

In [130]:
```
#Create Ensemble Model 1
estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42)), ('svr', make_pipeline(StandardScale
clf_1 = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
```

In [131]:
```
#fit the model with the training data and labels
clf_1.fit(X_train, y_train)
```

Out[131]:
```
StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(n_estimators=10,
                                                       random_state=42)),
                               ('svr',
                                Pipeline(steps=[('standardscaler',
                                                 StandardScaler()),
                                                ('svc',
                                                 SVC(random_state=42))]))],
                   final_estimator=LogisticRegression())
```

In [132]:
```
#generate predictions on the test data set
y_pred_1 = clf_1.predict(X_test)
```

In [133]:
```
#calculate RMSE
RMSE_1 = math.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred_1))
print('RMSE: ')
print(RMSE_1)
```

```
RMSE:
23.183051088914624
```

Ensemble Classifier 2:

1. Random Forest
2. SVM with Linear Kernel

In [142]:
```
#Create Ensemble Model 2
estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42))]
clf_2 = StackingClassifier(estimators=estimators, final_estimator=SVC(kernel='linear'))
```

In [143]:
```
#fit the model with the training data and labels
clf_2.fit(X_train, y_train)
```

Out[143]:
```
StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(n_estimators=50,
                                                       random_state=42))],
                   final_estimator=SVC(kernel='linear'))
```

In [144]:
```
#generate predictions on the test data set
y_pred_2 = clf_2.predict(X_test)
```

In [145]:
```
#calculate RMSE
RMSE_2 = math.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred_2))
print('RMSE: ')
print(RMSE_2)
```

```
RMSE:
21.69115726697019
```

Ensemble Classifier 3:

1. Random Forest
2. Logistic Regression
3. SVM with Linear Kernel

In [154]:
```
#Create Ensemble Model 3
estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42)), ('lr', LogisticRegression())]
clf_3 = StackingClassifier(estimators=estimators, final_estimator=SVC(kernel='linear'))
```

In [155]:
```
#fit the model with the training data and labels
clf_3.fit(X_train, y_train)
```

Out[155]:
```
StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(n_estimators=10,
                                                       random_state=42)),
                               ('lr', LogisticRegression())],
                   final_estimator=SVC(kernel='linear'))
```

In [158]:
```
#generate predictions on the test data set
y_pred_3 = clf_3.predict(X_test)
```

In [159]:
```
#calculate RMSE
RMSE_3 = math.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred_3))
print('RMSE: ')
print(RMSE_3)
```

```
RMSE:
21.691110769960467
```

Ensemble Classifier 4:

1. KNN
2. Decision Tree
3. SVM

In [176]:
```
#Create Ensemble Model 4
estimators = [('knn', KNeighborsClassifier(n_neighbors=1000)), ('dt', DecisionTreeClassifier(max_depth=25))]
clf_4 = StackingClassifier(estimators=estimators, final_estimator=SVC(kernel='linear'))
```

In [177]:
```
#fit the model with the training data and labels
clf_4.fit(X_train, y_train)
```

Out[177]:
```
StackingClassifier(estimators=[('knn', KNeighborsClassifier(n_neighbors=1000)),
                               ('dt', DecisionTreeClassifier(max_depth=25))],
                   final_estimator=SVC(kernel='linear'))
```

In [178]:
```
#generate predictions on the test data set
y_pred_4 = clf_4.predict(X_test)
```

In [179]:
```
#calculate RMSE
RMSE_4 = math.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred_4))
print('RMSE: ')
print(RMSE_4)
```

```
RMSE:
21.697154545717193
```

Ensemble 5: Only Random Forest

In [182]:
```
clf_5 = RandomForestClassifier(n_estimators=50, criterion='gini')
clf_5.fit(X_train, y_train)
```

Out[182]:
```
RandomForestClassifier(n_estimators=50)
```

In [183]:
```
y_pred_5 = clf_5.predict(X_test)
```

In [184]:
```
#calculate RMSE
RMSE_5 = math.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred_5))
print('RMSE: ')
print(RMSE_5)
```

```
RMSE:
23.87246539708768
```

Appendix C – Deep Learning Notebook

Part 1: Preprocess data

Load packages

[2]:
```
#load in packages
import os
import numpy as np
import pandas as pd
import cv2
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.regularizers import l2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.client import device_lib
import seaborn as sns
import matplotlib.pyplot as plt
```

[3]:
```
#Check to see which devices are availible for running network on
tf.config.get_visible_devices()
```

```
2021-12-06 21:08:52.577897: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:08:52.675807: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:08:52.676533: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
```

[3]:
```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
 PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

[4]:
```
#check if GPU available to use
#The GPU in the Kaggle online
if 'GPU' in str(device_lib.list_local_devices()):
    config = tf.compat.v1.ConfigProto(device_count = {'GPU': 0})
    sess = tf.compat.v1.Session(config=config)
```

```
2021-12-06 21:08:55.393185: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network L
ibrary (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-12-06 21:08:55.394603: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:08:55.395464: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:08:55.396144: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:08:57.300447: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:08:57.301279: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:08:57.301911: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:08:57.302498: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /device:GPU:0 with 15403 MB memory:  -> device: 0,
name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0
```

Kaggle has online GPUs that can be used, but they are time limited. Before running the notebook, turn the GPU on in the GUI on the right side of the notebook.

Part 2: Get the data

[5]:
```
#read training data and drop the meta data
train = pd.read_csv('../input/petfinder-pawpularity-score/train.csv')
train = train.drop(['Subject Focus', 'Eyes', 'Face', 'Near', 'Action', 'Accessory', 'Group', 'Collage', 'Human', 'Occlusion', 'Info', 'Blu

test = pd.read_csv('../input/petfinder-pawpularity-score/test.csv')
test = test.drop(['Subject Focus', 'Eyes', 'Face', 'Near', 'Action', 'Accessory', 'Group', 'Collage', 'Human', 'Occlusion', 'Info', 'Blur'
```

[6]:
```
#quick view of training data
train
```

[6]:

	Id	Pawpularity
0	0007de18844b0dbbb5e1f607da0606e0	63
1	0009c66b9439883ba2750fb825e1d7db	42
2	0013fd999caf9a3efe1352ca1b0d937e	28
3	0018df346ac9c1d8413cfcc888ca8246	15
4	001dc955e10590d3ca4673f034feeef2	72
...	...	...
9907	ffbfa0383c34dc513c95560d6e1fdb57	15
9908	ffcc8532d76436fc79e50eb2e5238e45	70
9909	ffdf2e8673a1da6fb80342fa3b119a20	20
9910	fff19e2ce11718548fa1c5d039a5192a	20
9911	fff8e47c766799c9e12f3cb3d66ad228	30

9912 rows × 2 columns

[7]:
```
#functions to add image paths to data frames for easy loading of images
def train_id_to_path(x):
    return '../input/petfinder-pawpularity-score/train/' + x + ".jpg"
def test_id_to_path(x):
    return '../input/petfinder-pawpularity-score/test/' + x + ".jpg"
```

[8]:
```
#also need to add .jpg extensions to image paths
train["img_path"] = train["Id"].apply(train_id_to_path)
test["img_path"] = test["Id"].apply(test_id_to_path)
```

[9]:
```
#quick view of training data with image paths included
train
```

[9]:

	Id	Pawpularity	img_path
0	0007de18844b0dbbb5e1f607da0606e0	63	../input/petfinder-pawpularity-score/train/000...
1	0009c66b9439883ba2750fb825e1d7db	42	../input/petfinder-pawpularity-score/train/000...
2	0013fd999caf9a3efe1352ca1b0d937e	28	../input/petfinder-pawpularity-score/train/001...
3	0018df346ac9c1d8413cfcc888ca8246	15	../input/petfinder-pawpularity-score/train/001...
4	001dc955e10590d3ca4673f034feeef2	72	../input/petfinder-pawpularity-score/train/001...
...	...	...	...
9907	ffbfa0383c34dc513c95560d6e1fdb57	15	../input/petfinder-pawpularity-score/train/ffb...
9908	ffcc8532d76436fc79e50eb2e5238e45	70	../input/petfinder-pawpularity-score/train/ffc...
9909	ffdf2e8673a1da6fb80342fa3b119a20	20	../input/petfinder-pawpularity-score/train/ffd...
9910	fff19e2ce11718548fa1c5d039a5192a	20	../input/petfinder-pawpularity-score/train/fff...
9911	fff8e47c766799c9e12f3cb3d66ad228	30	../input/petfinder-pawpularity-score/train/fff...

9912 rows × 3 columns

Part 3: Pre-processing image data

[10]:
```
#Set the size image you want to use
image_height = 128
image_width = 128

#Function that generates tensor
def generate_tensor(image_path):
    raw = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(raw, channels=3)
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.image.resize(image, (image_height, image_width))
    return image
```

[11]:
```
#get all the images in the training folder and put their tensors in a list
X = []
for img in train['img_path']:
    new_img_tensor = generate_tensor(img)
    X.append(new_img_tensor)
```

```
2021-12-06 21:09:27.637193: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:09:27.638053: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:09:27.638713: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:09:27.639414: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:09:27.640052: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
2021-12-06 21:09:27.640578: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/device:GPU:0 with
15403 MB memory:  -> device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0
```

[12]:
```
#same for the test images
X_submission = []
for img in test['img_path']:
    new_img_tensor = generate_tensor(img)
    X_submission.append(new_img_tensor)
```

[13]:
```
#convert X to array
X = np.array(X)
```

[16]:
```
#convert X_submission to array
X_submission = np.array(X_submission)
```

Now that all of our data is pre-processed and in tensor form, we can build the model

Part 3: split data into train and test

[17]:
```
#generate training data
y = train['Pawpularity']
```

[18]:
```
#split into training and testing data using a 90-10 split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=7)
```

[19]:
```
x_train.shape
```

[19…
```
(8920, 128, 128, 3)
```

Part 4: Define the model layers.

This is a SqueezeNet, which achieved the same accuracy as AlexNet, but is much smaller (reference: https://arxiv.org/pdf/1602.07360.pdf)

A fire module is a building block for CNN archetecture of the SqueezeNet which feeds 1x1 convolutional layers into an expanded layer with 1x1 and 3x3 convolutional layers

[20]:
```
#define the fire_module
def fire_module(x,s1,e1,e3):
    s1x = tf.keras.layers.Conv2D(s1,kernel_size = 1, padding = 'same')(x)
    s1x = tf.keras.layers.ReLU()(s1x)
    e1x = tf.keras.layers.Conv2D(e1,kernel_size = 1, padding = 'same')(s1x)
    e3x = tf.keras.layers.Conv2D(e3,kernel_size = 3, padding = 'same')(s1x)
    x = tf.keras.layers.concatenate([e1x,e3x])
    x = tf.keras.layers.ReLU()(x)
    return x
```

[22]:
```
#model layers
inputs = tf.keras.Input(shape=(image_height,image_width,3))
x = inputs
nclasses=1
x = tf.keras.layers.Conv2D(96,kernel_size=(7,7),strides=(2,2),padding='same')(x)
x = tf.keras.layers.MaxPool2D(pool_size=(3,3), strides = (2,2))(x)
x = fire_module(x, s1 = 16, e1 = 64, e3 = 64) #2
x = fire_module(x, s1 = 16, e1 = 64, e3 = 64) #3
x = fire_module(x, s1 = 32, e1 = 128, e3 = 128) #4
x = tf.keras.layers.MaxPool2D(pool_size=(3,3), strides = (2,2))(x)
x = fire_module(x, s1 = 32, e1 = 128, e3 = 128) #5
x = fire_module(x, s1 = 48, e1 = 192, e3 = 192) #6
x = fire_module(x, s1 = 48, e1 = 192, e3 = 192) #7
x = fire_module(x, s1 = 64, e1 = 256, e3 = 256) #8
x = tf.keras.layers.MaxPool2D(pool_size=(3,3), strides = (2,2))(x)
x = fire_module(x, s1 = 64, e1 = 256, e3 = 256) #9
x = tf.keras.layers.Dropout(0.5)(x)
x = tf.keras.layers.Conv2D(nclasses,kernel_size = 1)(x)
output = tf.keras.layers.AveragePooling2D(pool_size=(7,7))(x)
model = tf.keras.Model(inputs, output)
```

[23]:
```
#compile the model
model.compile(loss = 'mse', optimizer = 'Adam', metrics = [tf.keras.metrics.RootMeanSquaredError(name="rmse"), "mae", "mape"])
```

Part 5: Fit the model with the training data

This trick came from a tutorial which will help with randomness of orientation, zoom, and size shifts of images because not all images in the training set are oriented the same way (source: https://www.kaggle.com/alexteboul/tutorial-part-3-cnn-image-modeling-1)

[24]:

```
#augment data
data_augmentation = ImageDataGenerator(
    rotation_range = 15,
    zoom_range = 0.15,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.1,
    horizontal_flip = True,
    fill_mode = "nearest")
```

+ Code + Markdown

Fit the model using training data

[25]:

Add cell

```
#fit the model with training data
history = model.fit(
    data_augmentation.flow(x_train,y_train,batch_size=32),
    validation_data = (x_test,y_test),
    steps_per_epoch = len(x_train) // 32,
    epochs = 50
)
```

```
2021-12-06 21:16:37.775894: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/50
2021-12-06 21:16:40.013890: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
278/278 [==============================] - 41s 119ms/step - loss: 657.3150 - rmse: 25.6323 - mae: 18.9038 - mape: 82.5370 - val_loss: 485.9323 - val_rmse: 22.0439 - val_mae: 15.1072 - val_mape: 60.3186
Epoch 2/50
278/278 [==============================] - 33s 119ms/step - loss: 437.2376 - rmse: 20.9075 - mae: 15.6219 - mape: 79.0828 - val_loss: 460.0682 - val_rmse: 21.4492 - val_mae: 17.3500 - val_mape: 92.5578
Epoch 3/50
278/278 [==============================] - 32s 116ms/step - loss: 432.6881 - rmse: 20.8019 - mae: 15.5286 - mape: 78.7428 - val_loss: 431.4149 - val_rmse: 20.7705 - val_mae: 15.1205 - val_mape: 71.3278
Epoch 4/50
278/278 [==============================] - 32s 117ms/step - loss: 435.1990 - rmse: 20.8610 - mae: 15.5805 - mape: 78.5761 - val_loss: 428.0177 - val_rmse: 20.6886 - val_mae: 15.6939 - val_mape: 78.3945
Epoch 5/50
278/278 [==============================] - 33s 118ms/step - loss: 432.8888 - rmse: 20.8066 - mae: 15.5324 - mape: 78.5355 - val_loss: 438.1202 - val_rmse: 20.9313 - val_mae: 16.4305 - val_mape: 85.3156
Epoch 6/50
278/278 [==============================] - 33s 119ms/step - loss: 432.3567 - rmse: 20.7938 - mae: 15.5788 - mape: 79.1453 - val_loss: 428.1255 - val_rmse: 20.6912 - val_mae: 15.3279 - val_mape: 74.1682
Epoch 7/50
278/278 [==============================] - 33s 118ms/step - loss: 428.1245 - rmse: 20.6928 - mae: 15.4768 - mape: 78.8390 - val_loss: 461.0364 - val_rmse: 21.4718 - val_mae: 14.9382 - val_mape: 63.3372
```

```
Epoch 8/50
278/278 [==============================] - 33s 118ms/step - loss: 433.2159 - rmse: 20.8156 - mae: 15.5581 - mape: 79.0157
- val_loss: 428.1639 - val_rmse: 20.6921 - val_mae: 15.7159 - val_mape: 78.6343
Epoch 9/50
278/278 [==============================] - 33s 119ms/step - loss: 431.4011 - rmse: 20.7727 - mae: 15.5396 - mape: 79.2088
- val_loss: 435.3568 - val_rmse: 20.8652 - val_mae: 15.0218 - val_mape: 69.4233
Epoch 10/50
278/278 [==============================] - 33s 118ms/step - loss: 430.6962 - rmse: 20.7527 - mae: 15.4941 - mape: 78.9399
- val_loss: 428.1924 - val_rmse: 20.6928 - val_mae: 15.7199 - val_mape: 78.6776
Epoch 11/50
278/278 [==============================] - 33s 119ms/step - loss: 428.6797 - rmse: 20.7048 - mae: 15.4867 - mape: 79.1012
- val_loss: 429.2330 - val_rmse: 20.7179 - val_mae: 15.2327 - val_mape: 72.9090
Epoch 12/50
278/278 [==============================] - 34s 121ms/step - loss: 429.3785 - rmse: 20.7189 - mae: 15.4792 - mape: 78.9899
- val_loss: 429.2071 - val_rmse: 20.7173 - val_mae: 15.2344 - val_mape: 72.9328
Epoch 13/50
278/278 [==============================] - 34s 121ms/step - loss: 432.0623 - rmse: 20.7877 - mae: 15.5532 - mape: 78.9689
- val_loss: 430.6809 - val_rmse: 20.7529 - val_mae: 15.9752 - val_mape: 81.1959
Epoch 14/50
278/278 [==============================] - 33s 119ms/step - loss: 427.4268 - rmse: 20.6782 - mae: 15.4672 - mape: 78.6255
- val_loss: 431.4371 - val_rmse: 20.7711 - val_mae: 16.0325 - val_mape: 81.7427
Epoch 15/50
278/278 [==============================] - 33s 119ms/step - loss: 429.3130 - rmse: 20.7230 - mae: 15.5150 - mape: 79.2855
- val_loss: 427.4276 - val_rmse: 20.6743 - val_mae: 15.4712 - val_mape: 75.9408
Epoch 16/50
278/278 [==============================] - 33s 118ms/step - loss: 428.2765 - rmse: 20.6983 - mae: 15.4830 - mape: 78.7021
- val_loss: 427.4700 - val_rmse: 20.6753 - val_mae: 15.5671 - val_mape: 77.0099
Epoch 17/50
278/278 [==============================] - 33s 120ms/step - loss: 429.6242 - rmse: 20.7309 - mae: 15.5292 - mape: 79.1327
- val_loss: 430.7576 - val_rmse: 20.7547 - val_mae: 15.1487 - val_mape: 71.7294
Epoch 18/50
278/278 [==============================] - 33s 118ms/step - loss: 427.0002 - rmse: 20.6645 - mae: 15.4767 - mape: 79.1874
- val_loss: 431.3976 - val_rmse: 20.7701 - val_mae: 15.1194 - val_mape: 71.3177
Epoch 19/50
278/278 [==============================] - 33s 119ms/step - loss: 428.9557 - rmse: 20.7115 - mae: 15.5067 - mape: 79.0674
- val_loss: 430.1656 - val_rmse: 20.7404 - val_mae: 15.9339 - val_mape: 80.7902
Epoch 20/50
278/278 [==============================] - 33s 118ms/step - loss: 428.2119 - rmse: 20.6930 - mae: 15.4639 - mape: 78.9116
- val_loss: 439.1562 - val_rmse: 20.9561 - val_mae: 16.4831 - val_mape: 85.7615
Epoch 21/50
278/278 [==============================] - 33s 118ms/step - loss: 426.9579 - rmse: 20.6633 - mae: 15.4841 - mape: 79.1030
- val_loss: 428.1435 - val_rmse: 20.6916 - val_mae: 15.7132 - val_mape: 78.6042
Epoch 22/50
278/278 [==============================] - 33s 119ms/step - loss: 427.4890 - rmse: 20.6785 - mae: 15.4808 - mape: 79.0719
- val_loss: 431.4950 - val_rmse: 20.7725 - val_mae: 16.0369 - val_mape: 81.7830
Epoch 23/50
278/278 [==============================] - 33s 118ms/step - loss: 428.5356 - rmse: 20.6982 - mae: 15.4985 - mape: 79.1194
- val_loss: 429.3293 - val_rmse: 20.7203 - val_mae: 15.2266 - val_mape: 72.8224
Epoch 24/50
278/278 [==============================] - 33s 118ms/step - loss: 427.4497 - rmse: 20.6738 - mae: 15.4729 - mape: 79.0794
- val_loss: 427.4417 - val_rmse: 20.6747 - val_mae: 15.5519 - val_mape: 76.8440
Epoch 25/50
278/278 [==============================] - 33s 119ms/step - loss: 427.1352 - rmse: 20.6696 - mae: 15.4870 - mape: 79.0860
- val_loss: 428.1530 - val_rmse: 20.6919 - val_mae: 15.3245 - val_mape: 74.1259
Epoch 26/50
278/278 [==============================] - 33s 117ms/step - loss: 427.3764 - rmse: 20.6751 - mae: 15.4530 - mape: 78.9167
- val_loss: 427.4052 - val_rmse: 20.6738 - val_mae: 15.5109 - val_mape: 76.3970
Epoch 27/50
278/278 [==============================] - 33s 120ms/step - loss: 426.6152 - rmse: 20.6519 - mae: 15.4625 - mape: 79.0578
- val_loss: 430.6983 - val_rmse: 20.7533 - val_mae: 15.9766 - val_mape: 81.2090
Epoch 28/50
278/278 [==============================] - 33s 118ms/step - loss: 429.2037 - rmse: 20.7162 - mae: 15.4853 - mape: 79.1124
- val_loss: 427.5102 - val_rmse: 20.6763 - val_mae: 15.4356 - val_mape: 75.5005
Epoch 29/50
278/278 [==============================] - 33s 118ms/step - loss: 427.6227 - rmse: 20.6763 - mae: 15.4656 - mape: 78.9082
- val_loss: 432.7023 - val_rmse: 20.8015 - val_mae: 16.1219 - val_mape: 82.5644
Epoch 30/50
278/278 [==============================] - 33s 119ms/step - loss: 426.3980 - rmse: 20.6452 - mae: 15.4633 - mape: 79.2464
- val_loss: 429.2804 - val_rmse: 20.7191 - val_mae: 15.2297 - val_mape: 72.8661
```

```
Epoch 31/50
278/278 [==============================] - 33s 119ms/step - loss: 425.9152 - rmse: 20.6417 - mae: 15.4492 - mape: 78.8510
- val_loss: 427.9364 - val_rmse: 20.6866 - val_mae: 15.3525 - val_mape: 74.4714
Epoch 32/50
278/278 [==============================] - 33s 120ms/step - loss: 426.6295 - rmse: 20.6585 - mae: 15.4506 - mape: 79.1394
- val_loss: 433.8553 - val_rmse: 20.8292 - val_mae: 15.0534 - val_mape: 70.0645
Epoch 33/50
278/278 [==============================] - 33s 118ms/step - loss: 426.8551 - rmse: 20.6557 - mae: 15.4679 - mape: 79.1179
- val_loss: 428.0286 - val_rmse: 20.6889 - val_mae: 15.3400 - val_mape: 74.3170
Epoch 34/50
278/278 [==============================] - 33s 118ms/step - loss: 427.5607 - rmse: 20.6819 - mae: 15.4711 - mape: 79.0063
- val_loss: 428.2610 - val_rmse: 20.6945 - val_mae: 15.3121 - val_mape: 73.9724
Epoch 35/50
278/278 [==============================] - 33s 117ms/step - loss: 427.3858 - rmse: 20.6747 - mae: 15.4579 - mape: 78.9757
- val_loss: 427.4339 - val_rmse: 20.6745 - val_mae: 15.5467 - val_mape: 76.7871
Epoch 36/50
278/278 [==============================] - 33s 118ms/step - loss: 426.1289 - rmse: 20.6420 - mae: 15.4755 - mape: 79.4679
- val_loss: 427.4662 - val_rmse: 20.6753 - val_mae: 15.4514 - val_mape: 75.6954
Epoch 37/50
278/278 [==============================] - 33s 118ms/step - loss: 425.6871 - rmse: 20.6334 - mae: 15.4503 - mape: 79.0954
- val_loss: 430.6795 - val_rmse: 20.7528 - val_mae: 15.1525 - val_mape: 71.7822
Epoch 38/50
278/278 [==============================] - 33s 119ms/step - loss: 424.7947 - rmse: 20.6090 - mae: 15.4242 - mape: 78.8279
- val_loss: 435.2810 - val_rmse: 20.8634 - val_mae: 16.2756 - val_mape: 83.9772
Epoch 39/50
278/278 [==============================] - 33s 119ms/step - loss: 426.5152 - rmse: 20.6531 - mae: 15.4775 - mape: 79.3706
- val_loss: 429.3323 - val_rmse: 20.7203 - val_mae: 15.8579 - val_mape: 80.0435
Epoch 40/50
278/278 [==============================] - 33s 118ms/step - loss: 424.3046 - rmse: 20.6034 - mae: 15.4386 - mape: 78.9292
- val_loss: 429.4013 - val_rmse: 20.7220 - val_mae: 15.2221 - val_mape: 72.7591
Epoch 41/50
278/278 [==============================] - 33s 119ms/step - loss: 424.6630 - rmse: 20.6073 - mae: 15.4172 - mape: 78.7758
- val_loss: 428.0382 - val_rmse: 20.6891 - val_mae: 15.6978 - val_mape: 78.4361
Epoch 42/50
278/278 [==============================] - 33s 119ms/step - loss: 424.2237 - rmse: 20.5926 - mae: 15.4088 - mape: 79.0463
- val_loss: 428.5819 - val_rmse: 20.7022 - val_mae: 15.2796 - val_mape: 73.5670
Epoch 43/50
278/278 [==============================] - 33s 120ms/step - loss: 424.0207 - rmse: 20.5912 - mae: 15.4131 - mape: 78.8117
- val_loss: 427.7048 - val_rmse: 20.6810 - val_mae: 15.6378 - val_mape: 77.7812
Epoch 44/50
278/278 [==============================] - 33s 118ms/step - loss: 426.7630 - rmse: 20.6601 - mae: 15.4621 - mape: 79.1355
- val_loss: 432.6046 - val_rmse: 20.7992 - val_mae: 16.1154 - val_mape: 82.5048
Epoch 45/50
278/278 [==============================] - 33s 119ms/step - loss: 424.6087 - rmse: 20.6110 - mae: 15.4142 - mape: 78.9796
- val_loss: 428.1494 - val_rmse: 20.6918 - val_mae: 15.3250 - val_mape: 74.1310
Epoch 46/50
278/278 [==============================] - 33s 118ms/step - loss: 425.1930 - rmse: 20.6204 - mae: 15.4325 - mape: 78.9595
- val_loss: 429.3487 - val_rmse: 20.7207 - val_mae: 15.2253 - val_mape: 72.8052
Epoch 47/50
278/278 [==============================] - 33s 118ms/step - loss: 427.3042 - rmse: 20.6691 - mae: 15.4930 - mape: 79.1574
- val_loss: 428.7442 - val_rmse: 20.7061 - val_mae: 15.2669 - val_mape: 73.3886
Epoch 48/50
278/278 [==============================] - 33s 119ms/step - loss: 424.4364 - rmse: 20.6021 - mae: 15.4064 - mape: 78.8320
- val_loss: 429.5565 - val_rmse: 20.7257 - val_mae: 15.8798 - val_mape: 80.2582
Epoch 49/50
278/278 [==============================] - 33s 118ms/step - loss: 425.3122 - rmse: 20.6234 - mae: 15.4598 - mape: 79.2933
- val_loss: 431.2535 - val_rmse: 20.7666 - val_mae: 15.1258 - val_mape: 71.4073
Epoch 50/50
278/278 [==============================] - 32s 117ms/step - loss: 425.2615 - rmse: 20.6205 - mae: 15.4192 - mape: 78.8456
- val_loss: 427.4351 - val_rmse: 20.6745 - val_mae: 15.5474 - val_mape: 76.7957
```

Part 6: Generate Predictions on Test Set

[51]:
```
#predict on submission
y_pred = model.predict(X_submission)
```

[54]:
```
#reshape y_pred
y_pred = y_pred.reshape(8,1)
```

[63]:
```
#put predictions into a dataframe
y_pred_df = pd.DataFrame()
y_pred_df['Id'] = test['Id']
y_pred_df['Pawpularity'] = y_pred
print(y_pred_df)
```

```
                                 Id  Pawpularity
0  4128bae22183829d2b5fea10effdb0c3    38.250275
1  43a2262d7738e3d420d453815151079e    38.250275
2  4e429cead1848a298432a0acad014c9d    38.250275
3  80bc3ccafcc51b66303c2c263aa38486    38.250275
4  8f49844c382931444e68dffbe20228f4    38.250275
5  b03f7041962238a7c9d6537e22f9b017    38.250275
6  c978013571258ed6d4637f6e8cc9d6a3    38.250275
7  e0de453c1bffc20c22b072b34b54e50f    38.250275
```

[64]:
```
#submit to csv
y_pred_df.to_csv('submission.csv',index=False)
```

[29]:
```
#plot the training and validation RMSE for each epoch
plt.figure()
plt.plot(history.history["rmse"], label="Train RMSE")
plt.plot(history.history["val_rmse"], label="Validation RMSE")
plt.xlabel("Epoch #")
plt.ylabel("RMSE")
plt.title("RMSE for each Epoch")
plt.legend(loc="upper right")
```

Expand

[30]:
```
#plot the training and validation RMSE for each epoch
plt.figure()
plt.plot(history.history["loss"])
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.title("Loss for each Epoch")
```

Expand

Appendix D – Meta and Image Data Pre-Processing

Attempted to use this data in linear model, but results were terrible

In [3]:
```
#import libraries
import os
import numpy as np
import pandas as pd

#image processing
from PIL import Image, ImageOps
```

In [4]:
```
#load in the meta data as pandas data frame
#create one frame to store data as rgb, another to store it as grayscale
meta_train_rgb = pd.read_csv('petfinder-pawpularity-score/train.csv')
meta_train_grayscale = pd.read_csv('petfinder-pawpularity-score/train.csv')
```

In [12]:
```
#create a dummy vector of 0's so that we can vstack with new gray and rgb image vectors
gray_image_array_all = np.zeros(8100)
rgb_image_array_all = np.zeros(7500)

#loop over each for in the pandas data frame to grab the image id
for i in range(len(meta_train_grayscale)):
    #load image for row of meta data
    image = Image.open('petfinder-pawpularity-score/train/'+meta_train_grayscale['Id'][i]+'.jpg')
    #convert image to grayscale, resize it, convert to array, and flatten array
    gray_image = ImageOps.grayscale(image)
    resized_gray_image = gray_image.resize((90, 90))
    gray_image_array = np.array(resized_gray_image)
    gray_image_flat = gray_image_array.flatten()
    #stack the current grayscale image with the dummy vector created above
    gray_image_array_all = np.vstack((gray_image_array_all, gray_image_flat))

    #now we will repeat for the RGB images
    #resize image, convert to array, and flatten array
    resized_rgb_image = image.resize((50, 50))
    rgb_image_array = np.array(resized_rgb_image)
    rgb_image_flat = rgb_image_array.flatten()
    #stack the current grayscale image with the dummy vector created above
    rgb_image_array_all = np.vstack((rgb_image_array_all, rgb_image_flat))
```

In [20]:
```
#lets export the arrays containing image vectors because the above for loop took about an hour to run
np.savetxt("petfinder-pawpularity-score/pre_processed_data/pre_processed_images_grayscale.csv", gray_image_array_all, d
np.savetxt("petfinder-pawpularity-score/pre_processed_data/pre_processed_images_RGB.csv", rgb_image_array_all, delimite
```

In [32]:
```
#remove the first row of zeros the grayscale and RGB arrays
gray_image_array_all_ = gray_image_array_all[1:]
rgb_image_array_all_ = rgb_image_array_all[1:]
```

In [13]:
```
#next, we need to create column id's so we can convert the np arrays to dataframes to merge with meta data
gray_column_ids = []
for i in range(8100):
    i_str = str(i+1)
    gray_column_ids.append('p'+i_str)

rgb_column_ids = []
for i in range(7500):
    i_str = str(i+1)
    rgb_column_ids.append('p'+i_str)
```

In [33]:
```
#lets create the data frames
image_train_rgb = pd.DataFrame(rgb_image_array_all_, columns=rgb_column_ids)
image_train_gray = pd.DataFrame(gray_image_array_all_, columns=gray_column_ids)
```

In [34]:
```
#lastly, concatenate the meta data and the image data into one matrix
train_data_rgb = pd.concat([meta_train_rgb, image_train_rgb], axis=1)
train_data_gray = pd.concat([meta_train_grayscale, image_train_gray], axis=1)
```

In [35]:
```
#and for ease of splitting labels and data, lets move the pawpularity score to the beginning of the data frame
rgb_pawpularity = train_data_rgb.pop('Pawpularity')
train_data_rgb.insert(0, 'Pawpularity', rgb_pawpularity)
gray_pawpularity = train_data_gray.pop('Pawpularity')
train_data_gray.insert(0, 'Pawpularity', rgb_pawpularity)
```

In [40]:
```
#export both data frames to csv files
train_data_rgb.to_csv('meta_image_rgb_data.csv')
```

In [39]:
```
train_data_gray.to_csv('petfinder-pawpularity-score/meta_image_gray_data.csv')
```