# **Name:** Shixiang WANG, Jianbo Zhang

**EIDs:** sxwang6, bojzhang2

**Kaggle Competition:** PetFinder.my - Pawpularity Contest

**Kaggle Team Name:** Just Right

# CS5489 - Course Project (2021A)

Due date: See canvas site.

## Possible Projects

For the course project, you may select **one** of the following competitions on Kaggle **or** define your own course project:

### PetFinder.my - Pawpularity Contest: Predict the popularity of shelter pet photos

> A picture is worth a thousand words. But did you know a picture can save a thousand lives? Millions of stray animals suffer on the streets or are euthanized in shelters every day around the world. You might expect pets with attractive photos to generate more interest and be adopted faster. But what makes a good picture? With the help of data science, you may be able to accurately determine a pet photo's appeal and even suggest improvements to give these rescue animals a higher chance of loving homes.
>
> PetFinder.my is Malaysia's leading animal welfare platform, featuring over 180,000 animals with 54,000 happily adopted. PetFinder collaborates closely with animal lovers, media, corporations, and global organizations to improve animal welfare.
>
> Currently, PetFinder.my uses a basic Cuteness Meter to rank pet photos. It analyzes picture composition and other factors compared to the performance of thousands of pet profiles. While this basic tool is helpful, it's still in an experimental stage and the algorithm could be improved.
>
> In this competition, you'll analyze raw images and metadata to predict the "Pawpularity" of pet photos. You'll train and test your model on PetFinder.my's thousands of pet profiles. Winning versions will offer accurate recommendations that will improve animal welfare.
>
> If successful, your solution will be adapted into AI tools that will guide shelters and rescuers around the world to improve the appeal of their pet profiles, automatically enhancing photo quality and recommending composition improvements. As a result, stray dogs and cats can find their "furever" homes

> much faster. With a little assistance from the Kaggle community, many precious lives could be saved and more happy families created.
>
> Top participants may be invited to collaborate on implementing their solutions and creatively improve global animal welfare with their AI skills.

## G-Research Crypto Forecasting: Use your ML expertise to predict real crypto market data

> Over $40 billion worth of cryptocurrencies are traded every day. They are among the most popular assets for speculation and investment, yet have proven wildly volatile. Fast-fluctuating prices have made millionaires of a lucky few, and delivered crushing losses to others. Could some of these price movements have been predicted in advance?
>
> In this competition, you'll use your machine learning expertise to forecast short term returns in 14 popular cryptocurrencies. We have amassed a dataset of millions of rows of high-frequency market data dating back to 2018 which you can use to build your model. Once the submission deadline has passed, your final score will be calculated over the following 3 months using live crypto data as it is collected.
>
> The simultaneous activity of thousands of traders ensures that most signals will be transitory, persistent alpha will be exceptionally difficult to find, and the danger of overfitting will be considerable. In addition, since 2018, interest in the cryptomarket has exploded, so the volatility and correlation structure in our data are likely to be highly non-stationary. The successful contestant will pay careful attention to these considerations, and in the process gain valuable insight into the art and science of financial forecasting.
>
> G-Research is Europe's leading quantitative finance research firm. We have long explored the extent of market prediction possibilities, making use of machine learning, big data, and some of the most advanced technology available. Specializing in data science and AI education for workforces, Cambridge Spark is partnering with G-Research for this competition.

## Student-defined Course Project

The goal of the student-defined project is to get some hands-on experience using the course material on your own research problems. Keep in mind that there will only be about 4 weeks to do the project, so the scope should not be too large. Following the major themes of the course, here are some general topics for the project:

- *regression* (supervised learning) - use regression methods (e.g. ridge regression, Gaussian processes) to model data or predict from data.
- *classification* (supervised learning) - use classification methods (e.g., SVM, BDR, Logistic Regression, NNs) to learn to distinguish between multiple classes given a feature vector.

- *clustering* (unsupervised learning) - use clustering methods (e.g., K-means, EM, Mean-Shift) to discover the natural groups in data.
- *visualization* (unsupervised learning) - use dimensionality reduction methods (e.g., PCA, kernel-PCA, non-linear embedding) to visualize the structure of high-dimensional data.

You can pick any one of these topics and apply them to your own problem/data.

- *Can my project be my recently submitted or soon-to-be submitted paper?* If you plan to just turn in the results from your paper, then the answer is no. The project cannot be be work that you have already done. However, your course project can be based on extending your work. For example, you can try some models introduced in the course on your data/problem.

Before actually doing the project, you need to write a **project proposal** so that we can make sure the project is doable within the 3-4 weeks. I can also give you some pointers to relevant methods, if necessary.

- The project proposal should be at most one page with the following contents: 1) an introduction that briefy states the problem; 2) a precise description of what you plan to do - e.g., What types of features do you plan to use? What algorithms do you plan to use? What dataset will you use? How will you evaluate your results? How do you define a good outcome for the project?
- The goal of the proposal is to work out, in your head, what your project will be. Once the proposal is done, it is just a matter of implementation!
- *You need to submit the project proposal to Canvas 1 week after the Course project is released.*

## Groups

Group projects should contain 2 students. To sign up for a group, go to Canvas and under "People", join one of the existing "Project Groups". *For group projects, the project report must state the percentage contribution from each project member.*

## Methodology

You are free to choose the methodology to solve the task. In machine learning, it is important to use domain knowledge to help solve the problem. Hence, instead of blindly applying the algorithms to the data you need to think about how to represent the data in a way that makes sense for the algorithm to solve the task.

## Kaggle: Kaggle Notebooks

The Kaggle competitions have Kaggle Notebooks enabled, which provide free GPU/TPU computing resources (up to a limit). You can develop your model in the Kaggle Notebook, CS5489 JupyterHub, or on your own computers.

## Kaggle: Evaluation on Kaggle

For Kaggle projects, the final evaluation will be performed on Kaggle. Note that for these competitions you need to submit your code via the Kaggle Notebook, which will then generate the submission file for processing.

## Project Presentation

Each project group needs to give a presentation at the end of the semester. You will record your presentation and upload it to FlipGrid. The presentation is limited to 5 minutes. You *must* give a presentation. See the details in the "Project Presentations" Canvas assignment.

## What to hand in

You need to turn in the following things.

The following files should be uploaded to "Course Project" on Canvas:

1. This ipynb file `CourseProject-2021A.ipynb` with your source code and documentation. **You should write about all the various attempts that you make to find a good solution.** You may also submit .py files, but your documentation should be in the ipynb file.
2. A PDF version of your ipynb file.
3. Presentation slides.
4. (Kaggle projects) Your final submission file to Kaggle.
5. (Kaggle projects) A downloaded copy of your Kaggle Notebook that is submitted to Kaggle. This file should contain the code that generates the final submission file on Kaggle. This code will be used to verify that your Kaggle submission is reproducible.

Other things that need to be turned in:

- Upload your Project presentation to FlipGrid and the submit the URL to the "Project Presentations" assignment on Canvas. See the detailed instructions in the assignment.
- Enter the percentage contribution for each project member using the "Project Group Contribution" assignment on Canvas.
- (Student-defined projects) submit your project proposal to the "Project Proposal" assignment on Canvas. The project proposal is due 1 week after the course project is released. Kaggle projects do not need to submit a proposal.

## Grading

The marks of the assignment are distributed as follows:

- 40% - Results using various feature representations, dimensionality reduction methods, classifiers, etc.
- 25% - Trying out feature representations (e.g. adding additional features, combining features from different sources) or methods not used in the tutorials.
- 15% - Quality of the written report. More points for insightful observations and analysis.
- 15% - Project presentation

- 5% - (Kaggle projects) Final ranking on the Kaggle test data, or (student-defined projects) Project proposal.

**Late Penalty:** 25 marks will be subtracted for each day late.

**Group contribution:** marks for a group member with less than equal contribution will be deducted according to the following formula:

- Let A% and B% be the percentage contributions for group members Alice and Bob. A%+B%=100%
- Let x be the group project marks.
- If A>B, then Bob's marks will be reduced to be: x*B/A

---

# YOUR METHODS HERE

Initialize Python

In [1]:

```
# import packages
%matplotlib inline
import IPython.core.display
# setup output image format (Chrome works best)
IPython.core.display.set_matplotlib_formats("svg")
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib
import matplotlib.mlab as mlab
from numpy import *
from sklearn import *
import glob
import os
import csv
import string
random.seed(100)
import pandas as pd
import xgboost as xgb
from scipy import stats
import zipfile
import cv2
import seaborn as sns; sns.set_theme()
from PIL import Image
from pandas import Series,DataFrame

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Activation, Conv2D, Flatten, Dropo
                                    GlobalAveragePooling2D, Concatenate,MaxPo
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
import logging
logging.basicConfig()
import struct
# use keras backend (K) to force channels-last ordering
K.set_image_data_format('channels_last')
import tensorflow.keras.applications.resnet50 as resnet
print(f"Tensor Flow Version: {tf.__version__}")
```

```
print(f"Keras Version: {tf.keras.__version__}")
gpu = len(tf.config.list_physical_devices('GPU'))>0
print("GPU is", "available" if gpu else "NOT AVAILABLE")
```

```
/var/folders/3h/729_37h57cj7htnjk8qv4tkw0000gn/T/ipykernel_77308/1392842561.p
y:5: DeprecationWarning: `set_matplotlib_formats` is deprecated since IPython
7.23, directly use `matplotlib_inline.backend_inline.set_matplotlib_formats()`
  IPython.core.display.set_matplotlib_formats("svg")
Init Plugin
Init Graph Optimizer
Init Kernel
Tensor Flow Version: 2.5.0
Keras Version: 2.5.0
GPU is available
```

### Loading train data and test data

In [2]:

```
# Unzip the data file
# f = zipfile.ZipFile("petfinder-pawpularity-score.zip",'r')
# for file in f.namelist():
#     f.extract(file,"petfinder-pawpularity-score/")
# f.close()
```

In [3]:

```
train_csv_path = 'petfinder-pawpularity-score/train.csv'
test_csv_path = 'petfinder-pawpularity-score/test.csv'
train_imgs_path = 'petfinder-pawpularity-score/train'
test_imgs_path = 'petfinder-pawpularity-score/test'
dataset_path = 'petfinder-pawpularity-score'
train_df = pd.read_csv(train_csv_path)
test_df = pd.read_csv(test_csv_path)
```

### Define util functions

In [4]:

```
def write_csv_kaggle_sub(name, Y):
    test_df = pd.read_csv(test_csv_path)
    test_df["Pawpularity"] = Y
    test_df = test_df[["Id", "Pawpularity"]]
    test_df.to_csv(name, index=False)
```

In [5]:

```
# function to evaluate root mean squared error (MSE) and mean absolute error
def eval_predict(trueY, predY):
    MAE = metrics.mean_absolute_error(trueY, predY)
    RMSE = sqrt(metrics.mean_squared_error(trueY, predY))
    return RMSE, MAE
```

In [6]:

```
# function to show images
def showImgs(nrows, ncols, df):
    for i in range(nrows*ncols):
        img = mpimg.imread(train_imgs_path_dic.get(df.Id.values[i]))
        plt.subplot(nrows,ncols,i+1)
#         img = cv2.resize(img, (imgsize, imgsize))
        plt.axis("off")
        plt.tight_layout()
        plt.imshow(img)
```

In [7]:

```
def plot_history(history):
    fig, ax1 = plt.subplots()
```

```
    ax1.plot(history.history['loss'], 'r', label="training loss ({:.6f})".for
    ax1.plot(history.history['val_loss'], 'r--', label="validation loss ({:.6
    ax1.grid(True)
    ax1.set_xlabel('iteration')
    ax1.legend(loc="best", fontsize=9)
    ax1.set_ylabel('loss', color='r')
    ax1.tick_params('y', colors='r')
```

## Exploratory MetaData Analysis

- Explore features of metadata
  - Get the feature columns (We need to drop Id and Pawpularity since Id is useless to the model and Pawpularity is lable)
  - Show some simple statistics of the features.
  - Show the distribution of the features.

In [8]:

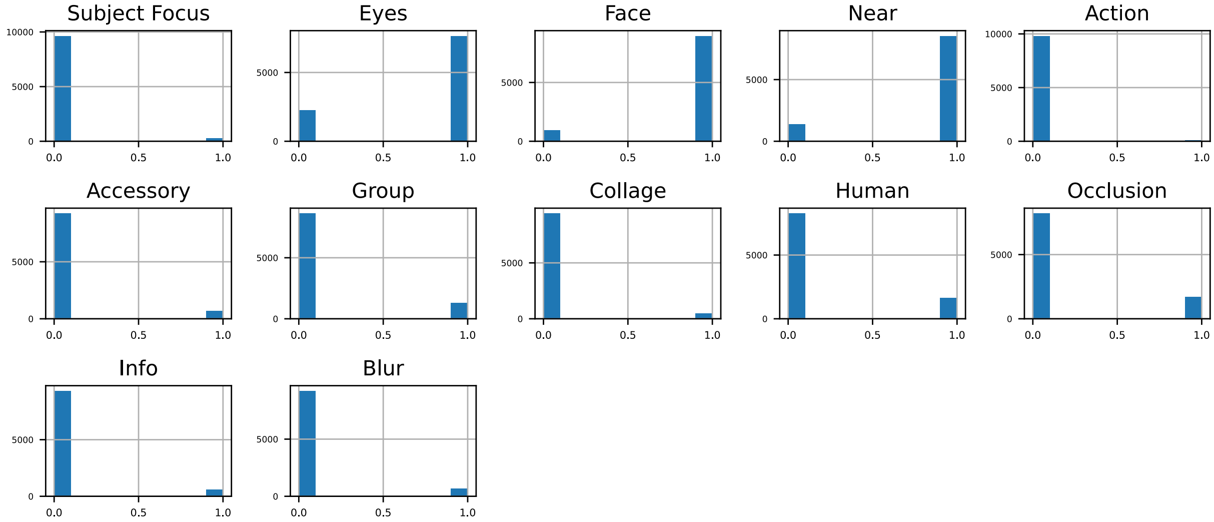
```
train_features=train_df.get(["Subject Focus","Eyes","Face","Near","Action","A
test=test_df.get(["Subject Focus","Eyes","Face","Near","Action","Accessory","
train_features.describe()
```

Out[8]:

| | Subject Focus | Eyes | Face | Near | Action | Accessory | |
|---|---|---|---|---|---|---|---|
| **count** | 9912.000000 | 9912.000000 | 9912.000000 | 9912.000000 | 9912.000000 | 9912.000000 | 9912 |
| **mean** | 0.027643 | 0.772599 | 0.903955 | 0.861582 | 0.009988 | 0.067797 | ( |
| **std** | 0.163957 | 0.419175 | 0.294668 | 0.345356 | 0.099444 | 0.251409 | ( |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | C |
| **25%** | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | C |
| **50%** | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | C |
| **75%** | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | C |
| **max** | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1 |

In [9]:

```
# show the features distributions
foo = train_features.hist(layout=(14,5), figsize=(10,20), xlabelsize=6, ylabe
plt.tight_layout()
```

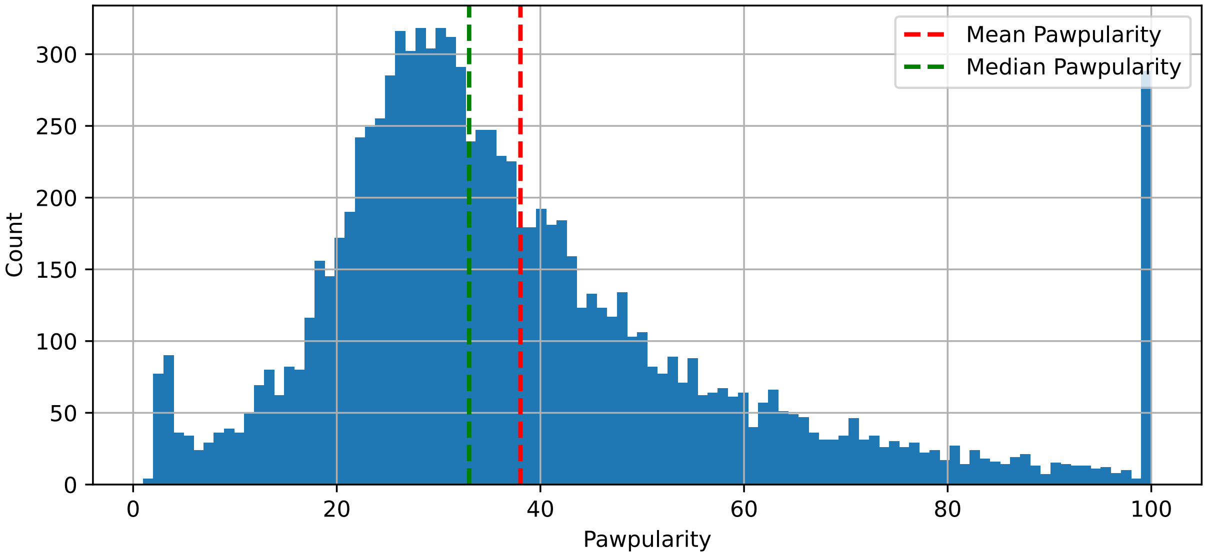



- Explore lable of metadata
  - Examine the distribution of the labels(Pawpularity)
  - Calculate the statistical information of lablels

In [10]:

```
import numpy as np
Y = train_df["Pawpularity"]
Y.hist(bins=100, figsize=(9,4))
plt.axvline(Y.mean(), c='red', ls='--', lw=2, label='Mean Pawpularity')
plt.axvline(Y.median(),c='green',ls='--',lw=2, label='Median Pawpularity')
plt.legend()
plt.xlabel('Pawpularity')
plt.ylabel('Count')
print("Number of unique values:", len(train_df['Pawpularity'].unique()))
# lable size
# print(Y.shape)
print("Mean of Pawpularity:",Y.mean())
print("Median of Pawpularity:",Y.median())
print("Sigma of Pawpularity:",Y.std())
print("Min value of Pawpularity:",Y.min())
print("Max value of Pawpularity:",Y.max())
```

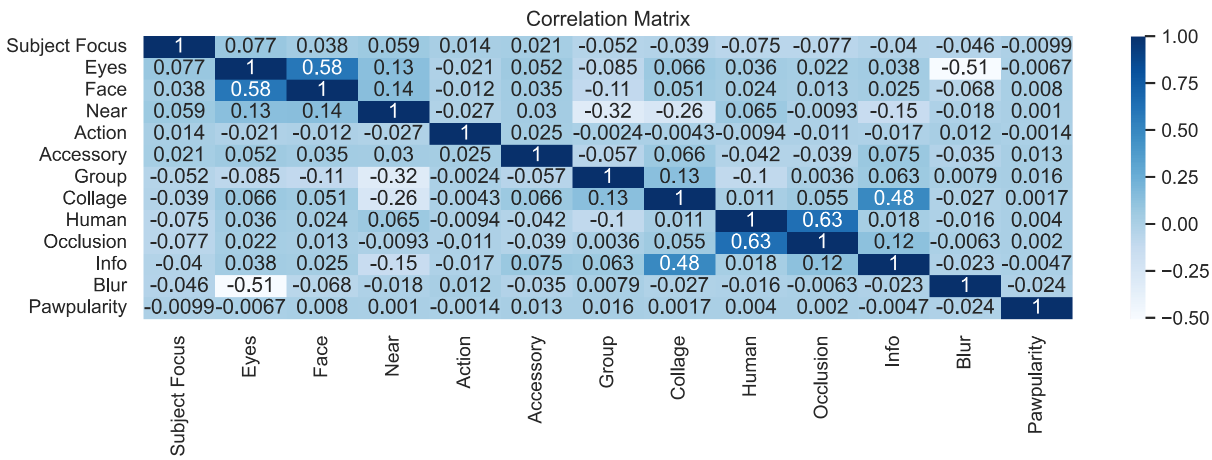
```
Number of unique values: 100
Mean of Pawpularity: 38.03904358353511
Median of Pawpularity: 33.0
Sigma of Pawpularity: 20.59199010577444
Min value of Pawpularity: 1
Max value of Pawpularity: 100
```




- Examine the correlation
  - between features and features
  - between features and label

In [12]:

```
import seaborn as sns
sns.set(rc={'figure.figsize':(12,3)})
sns.heatmap(train_df.corr(), cmap="Blues",annot=True)
plt.title('Correlation Matrix')
plt.rc('font',size=1)
plt.show()
```




- Analysis
  - There are only twelve features in the train metadata.
  - Only Eyes, Face, Near these three features have more one value than zero.But actually I calculate the mean Pawpularity of the train data whose Eyes, Face, and Near equal to 0, the mean is 37.374046 that is similiar to the mean of all train data.
  - The Action value of 9813 pieces train data is 0, while only 99 pieces have 1, the mean Pawpularity of these 99 pieces is 37.757576 which is very close to the mean of all trainning data. So we could get that Action does not have many contribution to the Pawpularity. The Correlation Matrix also verifies this truth, Action has the lowest correlation value.

- From the correlation matrix, we could get that Each individual feature has very little contribution to pawpularity.
- There are total 12 features in the train metadata, each feature's value is zero or one, so there are at most 2^12=4096 different train data but here we have 9912 pieces train data, so there must be lots of data that have same value of the all twelve features but different Pawpularity, these data will make our model confused.
- Actually, there are only 272 pieces of train data have total different value of features, if we we drop the other 9640 pieces, the train data will be too few. But if we do not drop it, the same data but have different Pawpularity will misleading our model. So if we use these metadata to train our model,the outcome may be not good.

## Model Training, Testing, and Prediction using metadata

- Split data into train and validation set

In [12]:
```
# randomly split data into 80% train and 20% validation set
trainX, valX, trainY, valY = \
  model_selection.train_test_split(train_features, Y,
  train_size=0.90, test_size=0.10, random_state=4487)

print(trainX.shape)
print(valX.shape)
```

```
(8920, 12)
(992, 12)
```

### Prediction with Linear Regression

- OLS

In [14]:
```
# using ordinary least squares
ols = linear_model.LinearRegression()
ols.fit(trainX, trainY)

print("ols MSE,RMAE =", eval_predict(valY, ols.predict(valX)))
write_csv_kaggle_sub("submissions/my_submission_ols.csv", ols.predict(test))
```

```
ols MSE,RMAE = (20.73824764611121, 15.76785972406625)
```

- Ridge Regression

In [15]:
```
# using Ridge Regression
# alpha values to try
alphas = logspace(-3,3,10)
# train RR with cross-validation
rr = linear_model.RidgeCV(alphas=alphas, cv=5)
rr.fit(trainX, trainY)

print("Ridge Regression RMSE,MAE =", eval_predict(valY, rr.predict(valX)))
write_csv_kaggle_sub("submissions/my_submission_rr.csv", rr.predict(test))
```

```
Ridge Regression RMSE,MAE = (20.743712171549397, 15.756259950364006)
```

- LASSO

In [16]:
```
# fit with cross-validation
las = linear_model.LassoCV()
las.fit(trainX, trainY)

print("lasso Regression RMSE,MAE =", eval_predict(valY, las.predict(valX)))
write_csv_kaggle_sub("submissions/my_submission_las.csv", las.predict(test))
```

```
lasso Regression RMSE,MAE = (20.75573688225275, 15.767690402140891)
```

Prediction with Non-Linear Regression

- Kernel Ridge Regression

In [17]:
```
# parameters for cross-validation
paramgrid = {'alpha': logspace(-2,2,5),
             'gamma': logspace(-2,2,5)}

# do cross-validation
krrcv = model_selection.GridSearchCV(
  kernel_ridge.KernelRidge(kernel='rbf'),  # estimator
  paramgrid,                               # parameters to try
  scoring='neg_mean_squared_error',        # score function
  cv=5,                                    # number of folds
  n_jobs=-1, verbose=True)
krrcv.fit(trainX, trainY)

print(krrcv.best_score_)
print(krrcv.best_params_)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
-423.17703544278993
{'alpha': 10.0, 'gamma': 0.1}
```

In [18]:
```
print("Kernel Ridge Regression RMSE,MAE =", eval_predict(valY, krrcv.predict(
write_csv_kaggle_sub("submissions/my_submission_krrcv.csv", krrcv.predict(tes
```

```
Kernel Ridge Regression RMSE,MAE = (20.717123104065067, 15.736461514585494)
```

- Support Vector Regression

In [19]:
```
# parameters for cross-validation
paramgrid = {'C':       logspace(-2,2,5),
             'gamma':   logspace(-2,2,5),
             'epsilon': logspace(-2,2,5)}

# do cross-validation
svrcv = model_selection.GridSearchCV(
    svm.SVR(kernel='rbf'),  # estimator
    paramgrid,                         # parameters to try
    scoring='neg_mean_squared_error',  # score function
    cv=5,
    n_jobs=-1, verbose=1)              # show progress
svrcv.fit(trainX, trainY)

print(svrcv.best_score_)
print(svrcv.best_params_)
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits
-430.85835459826467
{'C': 10.0, 'epsilon': 10.0, 'gamma': 0.1}
```

In [20]:
```
print("Support Vector Regression RMSE,MAE =", eval_predict(valY, svrcv.predic
write_csv_kaggle_sub("submissions/my_submission_svrcv.csv", svrcv.predict(tes
```

```
Support Vector Regression RMSE,MAE = (20.817379745431104, 15.129932661840236)
```

- Random Forest Regression

In [21]:
```
# parameters for cross-validation
paramgrid = {'max_depth': array([1,2,3,4,5,6,7,8,9,10,11,12,13,14]),
            }

# do cross-validation
rfcv = model_selection.GridSearchCV(
    ensemble.RandomForestRegressor(n_estimators=1000, random_state=4487),  #
    paramgrid,                      # parameters to try
    scoring='neg_mean_squared_error',  # score function
    cv=5,
    n_jobs=-1, verbose=True
)
rfcv.fit(trainX, trainY)

print(rfcv.best_score_)
print(rfcv.best_params_)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
-423.26150715885416
{'max_depth': 3}
```

In [22]:
```
print("Random Forest Regression RMSE,MAE =", eval_predict(valY, rfcv.predict(
write_csv_kaggle_sub("submissions/my_submission_rfcv.csv", rfcv.predict(test)
```

```
Random Forest Regression RMSE,MAE = (20.716464671286573, 15.739780845467504)
```

- Here I tried linear and Non-linear regression and did cross-validation, respectively. The best model is Random Forest Regression the validation MRES is 20.716464671286573, which is not so good as we analyzed before.

### Prediction with Using PCA

- Let us try to use PCA to reduce the input dimension on the metadata and using the best model above (Random Forest Regression) to make predict.
    - As the number of features is only 12, whcih is not too many compared to the unique of pawpularity, so we would better only reduce few dimensions.

In [13]:
```
# run PCA
pca = decomposition.PCA(n_components=10)
trainW   = pca.fit_transform(trainX)  # returns the coefficients
valW   = pca.transform(valX)
testW   = pca.transform(test)
v = pca.components_  # the principal component vector
m = pca.mean_        # the data mean
(trainW.shape, valW.shape)
```

Out[13]:
```
((8920, 10), (992, 10))
```

In [14]:
```
# parameters for cross-validation
paramgrid = {'max_depth': array([1,2,3,4,5,6,7,8,9,10,11,12,13,14]),
```

```
            }

# do cross-validation
PCA_rfcv = model_selection.GridSearchCV(
    ensemble.RandomForestRegressor(n_estimators=1000, random_state=4487),  # 
    paramgrid,                    # parameters to try
    scoring='neg_mean_squared_error',  # score function
    cv=5,
    n_jobs=-1, verbose=True
)
PCA_rfcv.fit(trainW, trainY)

print(PCA_rfcv.best_score_)
print(PCA_rfcv.best_params_)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
-423.2653393710858
{'max_depth': 2}
```

In [15]:

```
print("PCA RMSE,MAE =", eval_predict(valY, PCA_rfcv.predict(valW)))
write_csv_kaggle_sub("submissions/my_submission_PCA.csv", PCA_rfcv.predict(te
```

```
PCA RMSE,MAE = (20.718815193707346, 15.747754263136668)
```

Prediction with Cluster

- So far, we did not try cluster. Here, there are one hundred different values of pawpularity, so we make one hundred clusters

In [16]:

```
km = cluster.KMeans(n_clusters=100, random_state=4487)
trainXk = km.fit_predict(trainX)
valXk = km.predict(valX)
testk = km.predict(test)
```

In [17]:

```
print("Kmeans Cluster RMSE,MAE =", eval_predict(valY, valXk))
write_csv_kaggle_sub("submissions/my_submission_Cluster.csv", testk)
```

```
Kmeans Cluster RMSE,MAE = (37.62706965693371, 31.336693548387096)
```

- Because there are too many training data with different pawpularity but the same feature value, the outcome of kmeans cluster is not good

# Exploratory RawImage Analysis

Mapping the id of training images and test images corresponding to the file path

In [23]:

```
# dictionary. key:name of the image file, value: the path of the image file
train_imgs_path_dic={}
train_imgs_path_list= os.listdir(train_imgs_path)
for i in range(len(train_imgs_path_list)):
    train_imgs_path_dic[os.path.splitext(train_imgs_path_list[i])[0]]=train_i
# sample value of the dictionary
print(train_imgs_path_dic.get(os.path.splitext(train_imgs_path_list[0])[0]))
# size of the dictionary equals to total number of images
print(len(train_imgs_path_dic))
```

```
petfinder-pawpularity-score/train/0007de18844b0dbbb5e1f607da0606e0.jpg
9912
```

In [24]:

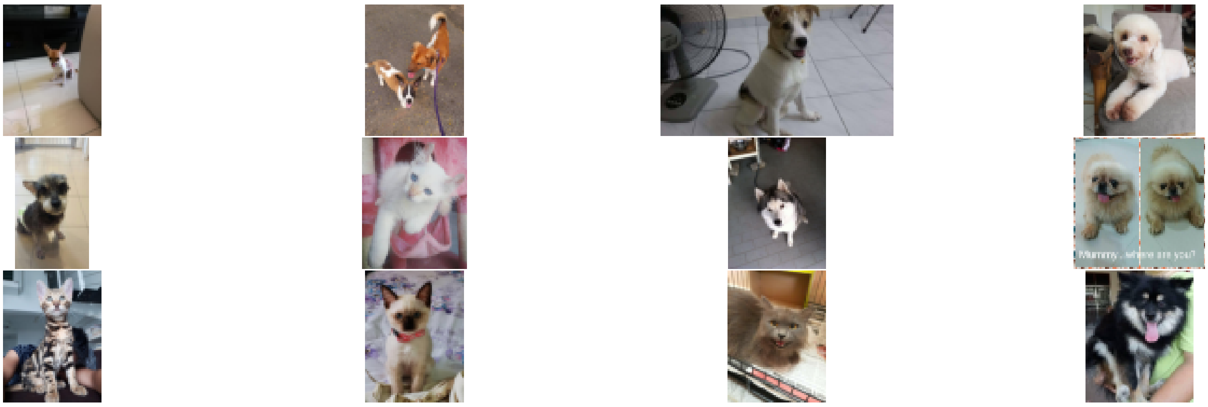
```
# dictionary. key:name of the image file, value: the path of the image file
test_imgs_path_dic={}
test_imgs_path_list= os.listdir(test_imgs_path)
for i in range(len(test_imgs_path_list)):
    test_imgs_path_dic[os.path.splitext(test_imgs_path_list[i])[0]]=test_imgs_
# sample value of the dictionary
print(test_imgs_path_dic.get(os.path.splitext(test_imgs_path_list[0])[0]))
# size of the dictionary equals to total number of images
print(len(test_imgs_path_dic))
```

```
petfinder-pawpularity-score/test/4128bae22183829d2b5fea10effdb0c3.jpg
8
```

- Show the first twelve images with 100 Pawpularity

In [26]:

```
showImgs(3,4,train_df[train_df.Pawpularity==100])
```



- Show images with only 1 Pawpularity

In [27]:

```
showImgs(2,2,train_df[train_df.Pawpularity==1])
```

We could firstly find that different raw images have different size, so look at the relationship between the size of the image and Pawpularity

In [28]:

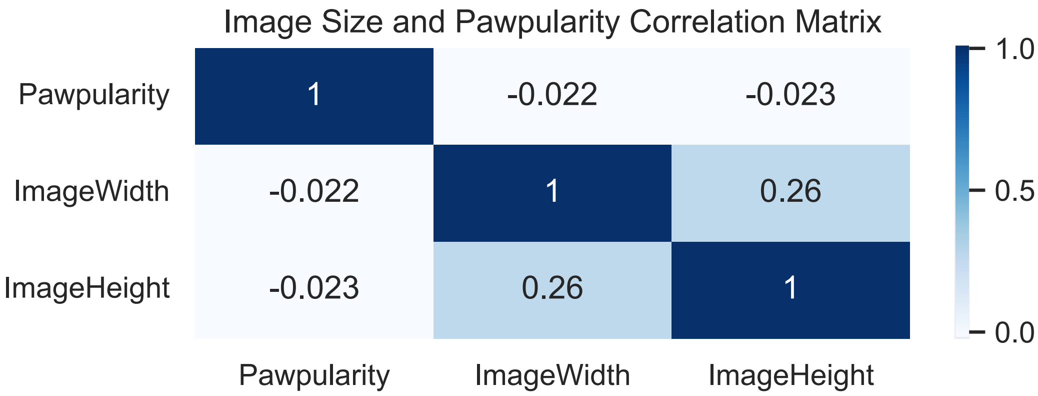
```
# a list of width of all images
Img_width=[]
# a list of height of all images
```

```
Img_height=[]
for i in range(len(train_imgs_path_dic)):
    img = Image.open(train_imgs_path_dic.get(os.path.splitext(train_imgs_path_
    Img_width.append(img.size[0])
    Img_height.append(img.size[1])
Image_size_data = {'Pawpularity':Y,
       'ImageWidth':Img_width,
       'ImageHeight':Img_height}
# Construct a dataframe
Image_size_data_df = DataFrame(Image_size_data)
```

Examine the correlation between image size and Pawpularity

In [29]:

```
sns.set(rc={'figure.figsize':(6,2)})
sns.heatmap(Image_size_data_df.corr(), cmap="Blues",annot=True)
plt.title('Image Size and Pawpularity Correlation Matrix')
plt.rc('font',size=1)
plt.show()
```




As we can get from the matrix, the width and height of the image has almost nothing to do with the pawpularity. This information is very important since when we would like to extract features from raw images, we usually hope the size of these photos to be consistent.

## Model Training, Testing, and Prediction using metadata

- Load taining and test images and set all images size to (128,128,3) (Here we choose this speciall width and height because my computer and the kaggle service cannnot do such a complicated compution.)

In [30]:

```
train_df = pd.read_csv(train_csv_path)
train_df['path'] = train_df['Id'].map(lambda x:str.format(dataset_path)+'/tra
print(train_df['path'][0])
train_df = train_df.drop(columns=['Id'])
train_df = train_df.sample(frac=1).reset_index(drop=True) #shuffle data
```

```
petfinder-pawpularity-score/train/0007de18844b0dbbb5e1f607da0606e0.jpg
```

In [31]:

```
test_df = pd.read_csv(test_csv_path)
test_df['path'] = test_df['Id'].map(lambda x:str.format(dataset_path)+'/test/
print(test_df['path'][0])
test_df = test_df.drop(columns=['Id'])
test_df = test_df.sample(frac=1).reset_index(drop=True) #shuffle data
```

```
petfinder-pawpularity-score/test/4128bae22183829d2b5fea10effdb0c3.jpg
```

- The same reason as choose images size (128,128,3), here I only used 6000 raw pictures to train

In [32]:
```
data_num = 6000
img_size =(128,128)
tunnel_size = (128,128,3)
trainimg = []
for i in range(data_num):
    image_string = tf.io.read_file(train_df['path'][i])
    image_decoded = tf.image.decode_jpeg(image_string)
    image_resized = tf.image.resize(image_decoded, img_size)
    trainimg.append(image_resized)
```

In [33]:
```
testimg = []
for i in range(len(test_df)):
    image_string = tf.io.read_file(test_df['path'][i])
    image_decoded = tf.image.decode_jpeg(image_string)
    image_resized = tf.image.resize(image_decoded, img_size)
    testimg.append(image_resized)
```

In [34]:
```
trainimg = array(trainimg)
print(trainimg.shape)
testimg = array(testimg)
print(testimg.shape)
Y = train_df["Pawpularity"]
Y = Y[0:data_num]
print(Y.shape)
```

```
(6000, 128, 128, 3)
(8, 128, 128, 3)
(6000,)
```

- Nomorlize the data to 0-1 which neural network prefers.

In [35]:
```
# map the data to [0,1]
train_X=trainimg/255.0
test_X = testimg/255.0
Itrain=train_X.reshape(train_X.shape[0],train_X.shape[1],train_X.shape[1]*3)
Itest=test_X.reshape(test_X.shape[0],test_X.shape[1],test_X.shape[1]*3)
```

- Multi-layer perceptron

- Randomly split data into 80% train and 20% validation set

In [36]:
```
train_x, val_x, train_Y, val_Y = \
  model_selection.train_test_split(Itrain, Y,
  train_size=0.8, test_size=0.2, random_state=4487)
((train_x.shape), (val_x.shape), (train_Y.shape), (val_Y.shape))
```

Out[36]:
```
((4800, 128, 384), (1200, 128, 384), (4800,), (1200,))
```

In [37]:
```
# initialize random seed
K.clear_session()
random.seed(4487); tf.random.set_seed(4487)
```

```
# build the network
nn = Sequential()
nn.add(Flatten(input_shape=(train_x.shape[1], train_x.shape[2])))
# 5 hidden layer
nn.add(Dense(units=1024, activation='relu'))
nn.add(Dense(units=512, activation='relu'))
nn.add(Dense(units=256, activation='relu'))
nn.add(Dense(units=64, activation='relu'))
nn.add(Dense(units=64, activation='relu'))
nn.add(Dense(units=1, activation='relu'))

# setup early stopping callback function
earlystop = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,       # threshold to consider as no change
    patience=5,             # stop if 5 epochs with no change
    verbose=1, mode='auto'
)

callbacks_list = [earlystop]

# compile and fit the network
nn.compile( loss = keras.losses.MeanSquaredError(), # use mean squared error
            optimizer=keras.optimizers.Adam(),
           # optimizer=keras.optimizers.SGD(lr=0.02, momentum=0.9, nesterov=T
            metrics = ['mean_squared_error']
          )

history_nn=nn.fit(train_x, train_Y, epochs=100, batch_size=100,
                 callbacks=callbacks_list,
                 validation_data=(val_x, val_Y),  # specify the validation se
                 verbose=1)
```

```
Epoch 1/100
48/48 [==============================] - 18s 360ms/step - loss: 700.7166 - mea
n_squared_error: 700.7166 - val_loss: 501.3509 - val_mean_squared_error: 501.3
509
Epoch 2/100
48/48 [==============================] - 16s 334ms/step - loss: 505.7188 - mea
n_squared_error: 505.7188 - val_loss: 487.9014 - val_mean_squared_error: 487.9
014
Epoch 3/100
48/48 [==============================] - 16s 337ms/step - loss: 512.3016 - mea
n_squared_error: 512.3017 - val_loss: 531.2977 - val_mean_squared_error: 531.2
977
Epoch 4/100
48/48 [==============================] - 16s 331ms/step - loss: 480.1839 - mea
n_squared_error: 480.1839 - val_loss: 473.3593 - val_mean_squared_error: 473.3
593
Epoch 5/100
48/48 [==============================] - 16s 329ms/step - loss: 481.0627 - mea
n_squared_error: 481.0627 - val_loss: 482.6629 - val_mean_squared_error: 482.6
629
Epoch 6/100
48/48 [==============================] - 16s 332ms/step - loss: 472.5297 - mea
n_squared_error: 472.5297 - val_loss: 463.1359 - val_mean_squared_error: 463.1
359
Epoch 7/100
48/48 [==============================] - 16s 332ms/step - loss: 457.4057 - mea
n_squared_error: 457.4057 - val_loss: 462.3861 - val_mean_squared_error: 462.3
861
Epoch 8/100
48/48 [==============================] - 16s 330ms/step - loss: 458.9314 - mea
n_squared_error: 458.9314 - val_loss: 466.0851 - val_mean_squared_error: 466.0
```

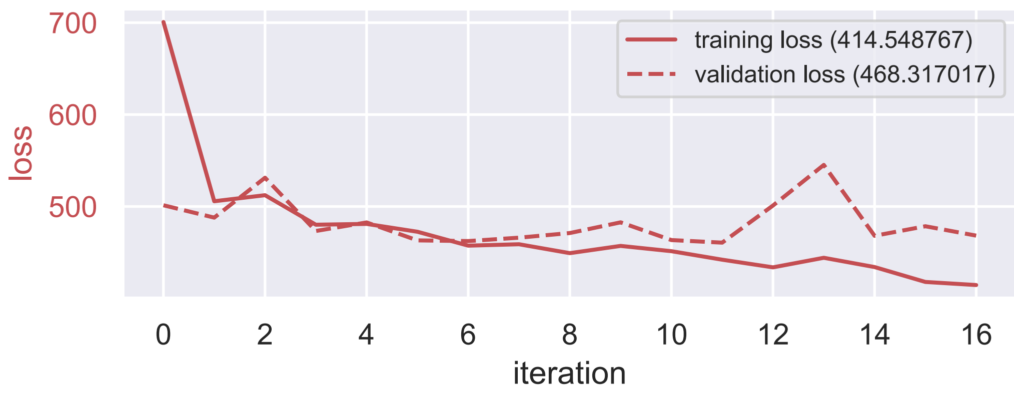
```
851
Epoch 9/100
48/48 [==============================] - 16s 327ms/step - loss: 449.2474 - mea
n_squared_error: 449.2474 - val_loss: 471.1196 - val_mean_squared_error: 471.1
196
Epoch 10/100
48/48 [==============================] - 16s 326ms/step - loss: 457.1255 - mea
n_squared_error: 457.1255 - val_loss: 482.7661 - val_mean_squared_error: 482.7
661
Epoch 11/100
48/48 [==============================] - 16s 328ms/step - loss: 451.3840 - mea
n_squared_error: 451.3840 - val_loss: 463.4683 - val_mean_squared_error: 463.4
683
Epoch 12/100
48/48 [==============================] - 16s 327ms/step - loss: 442.1187 - mea
n_squared_error: 442.1187 - val_loss: 460.6818 - val_mean_squared_error: 460.6
818
Epoch 13/100
48/48 [==============================] - 16s 340ms/step - loss: 433.7974 - mea
n_squared_error: 433.7974 - val_loss: 501.0725 - val_mean_squared_error: 501.0
725
Epoch 14/100
48/48 [==============================] - 16s 328ms/step - loss: 444.2019 - mea
n_squared_error: 444.2019 - val_loss: 545.2198 - val_mean_squared_error: 545.2
198
Epoch 15/100
48/48 [==============================] - 16s 339ms/step - loss: 434.1123 - mea
n_squared_error: 434.1123 - val_loss: 468.1295 - val_mean_squared_error: 468.1
295
Epoch 16/100
48/48 [==============================] - 17s 361ms/step - loss: 417.9078 - mea
n_squared_error: 417.9078 - val_loss: 478.5139 - val_mean_squared_error: 478.5
139
Epoch 17/100
48/48 [==============================] - 18s 372ms/step - loss: 414.5488 - mea
n_squared_error: 414.5488 - val_loss: 468.3170 - val_mean_squared_error: 468.3
170
Epoch 00017: early stopping
```

In [38]:

```
plot_history(history_nn)
nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 49152)             0
_________________________________________________________________
dense (Dense)                (None, 1024)              50332672
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 64)                16448
_________________________________________________________________
dense_4 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 51,009,473
Trainable params: 51,009,473
```

```
Non-trainable params: 0
_________________________________________________________________
```




In [39]:
```
mlp_predict=nn.predict(val_x)
print("MLP RMSE,MAE =", eval_predict(val_Y, mlp_predict))
write_csv_kaggle_sub("submissions/my_submission_mlp.csv", nn.predict(Itest))
```

```
MLP RMSE,MAE = (21.64063389579775, 15.9560591562589)
```

- We could get that deep learning model is not good from above, so we tried to use transfer learning

In [40]:
```
trainXim, valXim, trainYim, valYim = \
  model_selection.train_test_split(train_X, Y,
  train_size=0.8, test_size=0.2, random_state=4487)
((trainXim.shape), (valXim.shape), (trainYim.shape), (valYim.shape))
```

Out[40]:
```
((4800, 128, 128, 3), (1200, 128, 128, 3), (4800,), (1200,))
```

- Define a function to return different available models. Here I tried six.
  - ResNet152V2
  - ResNet50
  - InceptionResNetV2
  - DenseNet201
  - EfficientNetB7
  - Xception

In [41]:
```
# try different available models to get pre-trained weights.
# reference https://keras.io/api/applications/
def transferModel(model_name):
    print('model transfer start--'+model_name)
    if model_name == "ResNet152V2":
        model = tf.keras.applications.ResNet152V2( include_top=False, weights
    if model_name == "ResNet50":
        model = tf.keras.applications.resnet50.ResNet50(include_top=False, we
    if model_name == "InceptionResNetV2":
        model = tf.keras.applications.inception_resnet_v2.InceptionResNetV2(i
    if model_name == "DenseNet201":
        model = tf.keras.applications.densenet.DenseNet201(include_top=False,
    if model_name == "EfficientNetB7":
        model = tf.keras.applications.efficientnet.EfficientNetB7(include_top
    if model_name == "Xception":
        model = tf.keras.applications.Xception(include_top=False, weights='im
    model.trainable = False
```

```
    print('model transfer end--'+model_name)
    return model
```

- Define a function to preprocess the input of different Keras Application

In [42]:

```
# each Keras Application expects a specific kind of input preprocessing, so w
# to convert the input images from RGB to BGR, then will zero-center each col
# here I reference https://keras.io/api/applications/
def preprocessInput(model_name):
    print('preprocessInput start--'+ model_name)
    if model_name == "ResNet152V2":
        preprocess_Input=tf.keras.applications.resnet_v2.preprocess_input
    if model_name == "ResNet50":
        preprocess_Input=tf.keras.applications.resnet50.preprocess_input
    if model_name == "InceptionResNetV2":
        preprocess_Input=tf.keras.applications.inception_resnet_v2.preprocess_
    if model_name == "DenseNet201":
        preprocess_Input=tf.keras.applications.densenet.preprocess_input
    if model_name == "EfficientNetB7":
        preprocess_Input=tf.keras.applications.efficientnet.preprocess_input
    if model_name == "Xception":
        preprocess_Input=tf.keras.applications.xception.preprocess_input
    print('preprocessInput end--'+model_name)
    return preprocess_Input
```

- Define a function to give different noise on the raw picture. Here I tried 4.
    - Add No Noise
    - Add Guass Noise
    - Add Corrupt Noise
    - Add Scale Shift Noise

In [43]:

```
def add_no_noise(X):
    return X

def add_gauss_noise(X, sigma2=0.05):
    # add Gaussian noise with zero mean, and variance sigma2
    return X + random.normal(0, sigma2, X.shape)

def add_corrupt_noise(X, p=0.1):
    # apply pixel corruption (zero out value) with probability p
    return X * random.binomial(1, 1-p, X.shape)

def add_scale_shift(X, sigma2=0.1, alpha2=0.2):
    # randomly scale and shift the pixel values (same for each image)
    # Xnew = a X + b
    # a is sampled from a Gaussian with mean 1, and variance sigma2
    # b is sampled from a Gaussian with mean 0, and variance alpha2
    if X.ndim == 3:
        dshape = (X.shape[0],1,1)
    elif X.ndim == 4:
        dshape = (X.shape[0],1,1,1)
    else:
        dshape = (1,)
    a = random.normal(1,sigma2, dshape)
    b = random.normal(0,alpha2, dshape)
    return minimum(maximum( a*X + b, 0.0), 1.0)
```

- Define a function to add noise to the raw data

In [44]:
```
def dataAugmentation(aug_name):
    print('dataAugmentation start--'+aug_name)
    if aug_name == "add_gauss_noise":
        ret = add_gauss_noise
    if aug_name == "add_corrupt_noise":
        ret = add_corrupt_noise
    if aug_name == "add_scale_shift":
        ret = add_scale_shift
    if aug_name == "add_no_noise":
        ret = add_no_noise
    print('dataAugmentation end--'+aug_name)
    return ret
```

- Define a function to preprocess train and validation data using above functions

In [45]:
```
def data_process(trans_model_name, agu_name, trainXim, valXim):
#   preprocess the input of different Keras Application
    trainXim = preprocessInput(trans_model_name)(trainXim)
    valXim = preprocessInput(trans_model_name)(valXim)
#   add augmentation
    train_x = dataAugmentation(agu_name)(trainXim)
    val_x = dataAugmentation(agu_name)(valXim)
    return train_x,val_x
```

- mode_1: tried to do feature extraction from InceptionResNetV2

In [46]:
```
# initialize random seed
K.clear_session()
random.seed(999); tf.random.set_seed(999)

# build model 1
model_1 = Sequential()
model_1.add(transferModel('InceptionResNetV2'))
model_1.add(BatchNormalization())
model_1.add(Flatten())
model_1.add(Dense(units=256, activation='relu'))
model_1.add(BatchNormalization())
model_1.add(Dense(units=128, activation='relu'))
model_1.add(Dense(units=64))
# model_1.add(Dropout(0.2))
model_1.add(Dense(units=1,activation=tf.keras.layers.ReLU(max_value = 100)))

model_1.summary()
```

```
model transfer start--InceptionResNetV2
model transfer end--InceptionResNetV2
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_resnet_v2 (Functio (None, 2, 2, 1536)        54336736
_________________________________________________________________
batch_normalization_203 (Bat (None, 2, 2, 1536)        6144
_________________________________________________________________
flatten (Flatten)            (None, 6144)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1573120
```

```
_________________________________________________________________
batch_normalization_204 (Bat (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 55,958,241
Trainable params: 1,617,921
Non-trainable params: 54,340,320
_________________________________________________________________
```

In [47]:

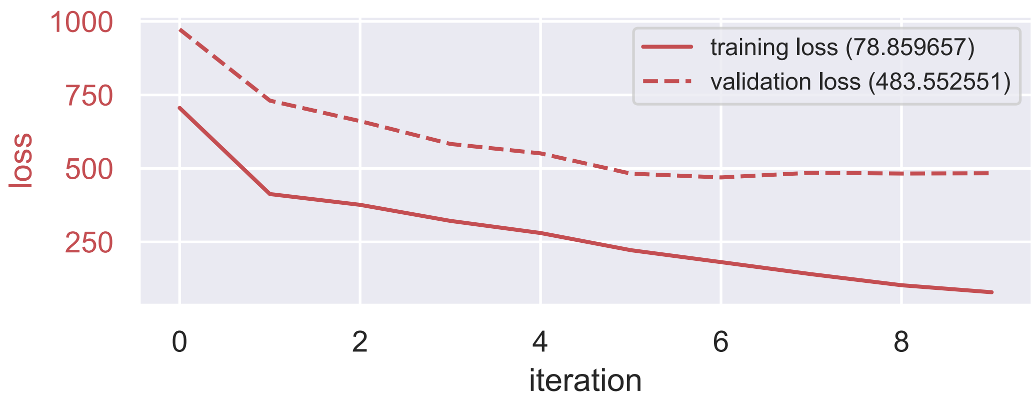
```
# early stopping criteria
earlystop = keras.callbacks.EarlyStopping(
              monitor='val_loss',
              min_delta=1, patience=3,
              verbose=1, mode='auto')
callbacks_list = [earlystop]

# compile and fit the network
model_1.compile( loss = keras.losses.MeanSquaredError(),
             optimizer=keras.optimizers.Adam(),
             metrics = ['mean_squared_error']
          )
train_x,val_x = data_process('InceptionResNetV2', 'add_gauss_noise',trainXim,
history_model_1 = model_1.fit(train_x, trainYim, epochs=100, batch_size=100,
                 callbacks=callbacks_list,
                 validation_data=(val_x,valYim),  # specify the validation se
                 verbose=1)
```

```
preprocessInput start--InceptionResNetV2
preprocessInput end--InceptionResNetV2
preprocessInput start--InceptionResNetV2
preprocessInput end--InceptionResNetV2
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
Epoch 1/100
48/48 [==============================] - 222s 4s/step - loss: 705.8855 - mean_
squared_error: 705.8855 - val_loss: 973.2528 - val_mean_squared_error: 973.252
8
Epoch 2/100
48/48 [==============================] - 226s 5s/step - loss: 412.8303 - mean_
squared_error: 412.8303 - val_loss: 730.6570 - val_mean_squared_error: 730.657
0
Epoch 3/100
48/48 [==============================] - 214s 4s/step - loss: 376.1781 - mean_
squared_error: 376.1781 - val_loss: 660.9352 - val_mean_squared_error: 660.935
2
Epoch 4/100
48/48 [==============================] - 203s 4s/step - loss: 321.5601 - mean_
squared_error: 321.5601 - val_loss: 583.3026 - val_mean_squared_error: 583.302
6
Epoch 5/100
48/48 [==============================] - 203s 4s/step - loss: 280.3304 - mean_
squared_error: 280.3304 - val_loss: 551.2075 - val_mean_squared_error: 551.207
5
Epoch 6/100
48/48 [==============================] - 202s 4s/step - loss: 221.9207 - mean_
squared_error: 221.9207 - val_loss: 482.3055 - val_mean_squared_error: 482.305
```

```
5
Epoch 7/100
48/48 [==============================] - 206s 4s/step - loss: 181.3041 - mean_
squared_error: 181.3041 - val_loss: 469.3669 - val_mean_squared_error: 469.366
9
Epoch 8/100
48/48 [==============================] - 204s 4s/step - loss: 140.3214 - mean_
squared_error: 140.3214 - val_loss: 485.2629 - val_mean_squared_error: 485.262
9
Epoch 9/100
48/48 [==============================] - 205s 4s/step - loss: 102.7220 - mean_
squared_error: 102.7220 - val_loss: 482.6056 - val_mean_squared_error: 482.605
6
Epoch 10/100
48/48 [==============================] - 204s 4s/step - loss: 78.8597 - mean_s
quared_error: 78.8597 - val_loss: 483.5526 - val_mean_squared_error: 483.5526
Epoch 00010: early stopping
```

In [48]:

```
plot_history(history_model_1)
```




In [49]:

```
model_1_predict=model_1.predict(valXim)
print("model_1 RMSE,MAE =", eval_predict(valYim, model_1_predict))
write_csv_kaggle_sub("submissions/my_submission_model_1.csv", model_1.predict
```

```
model_1 RMSE,MAE = (20.965820018274336, 15.996552220980327)
```

- mode_2: tried to do feature extraction from DenseNet201

In [50]:

```
# initialize random seed
K.clear_session()
random.seed(999); tf.random.set_seed(999)

# build mode2
model_2 = Sequential()
model_2.add(transferModel('DenseNet201'))
model_2.add(BatchNormalization())
model_2.add(Flatten())
model_2.add(Dense(units=256, activation='relu'))
model_2.add(BatchNormalization())
model_2.add(Dense(units=128, activation='relu'))
model_2.add(Dense(units=64, activation='relu'))
# model_2.add(Dropout(0.2))
model_2.add(Dense(units=1, activation=tf.keras.layers.ReLU(max_value = 100)))

model_2.summary()
```

```
model transfer start--DenseNet201
```

```
model transfer end--DenseNet201
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
densenet201 (Functional)     (None, 4, 4, 1920)        18321984  
_________________________________________________________________
batch_normalization (BatchNo (None, 4, 4, 1920)        7680      
_________________________________________________________________
flatten (Flatten)            (None, 30720)             0         
_________________________________________________________________
dense (Dense)                (None, 256)               7864576   
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024      
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896     
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65        
=================================================================
Total params: 26,236,481
Trainable params: 7,910,145
Non-trainable params: 18,326,336
_________________________________________________________________
```

In [51]:

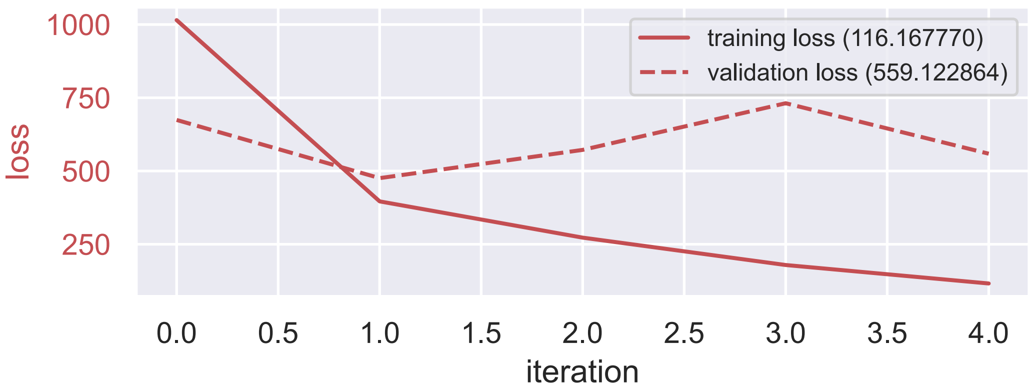
```
# early stopping criteria
earlystop = keras.callbacks.EarlyStopping(
              monitor='val_loss',
              min_delta=1, patience=3,
              verbose=1, mode='auto')
callbacks_list = [earlystop]

# compile and fit the network
model_2.compile( loss = keras.losses.MeanSquaredError(),
             optimizer=keras.optimizers.Adam(),
             metrics = ['mean_squared_error']
          )
train_x,val_x = data_process('DenseNet201', 'add_gauss_noise',trainXim, valXi
history_model_2 = model_2.fit(train_x, trainYim, epochs=100, batch_size=100,
                 callbacks=callbacks_list,
                 validation_data=(val_x,valYim),  # specify the validation se
                 verbose=1)
```

```
preprocessInput start--DenseNet201
preprocessInput end--DenseNet201
preprocessInput start--DenseNet201
preprocessInput end--DenseNet201
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
Epoch 1/100
48/48 [==============================] - 297s 6s/step - loss: 1015.0632 - mean
_squared_error: 1015.0632 - val_loss: 674.2979 - val_mean_squared_error: 674.2
979
Epoch 2/100
48/48 [==============================] - 283s 6s/step - loss: 396.3840 - mean_
squared_error: 396.3840 - val_loss: 475.7418 - val_mean_squared_error: 475.741
8
Epoch 3/100
48/48 [==============================] - 283s 6s/step - loss: 272.6093 - mean_
squared_error: 272.6093 - val_loss: 572.1353 - val_mean_squared_error: 572.135
```

```
2
Epoch 4/100
48/48 [==============================] - 293s 6s/step - loss: 179.1754 - mean_
squared_error: 179.1754 - val_loss: 731.3897 - val_mean_squared_error: 731.389
7
Epoch 5/100
48/48 [==============================] - 285s 6s/step - loss: 116.1678 - mean_
squared_error: 116.1678 - val_loss: 559.1229 - val_mean_squared_error: 559.122
9
Epoch 00005: early stopping
```

In [52]:

```
plot_history(history_model_2)
```




In [53]:

```
model_2_predict=model_2.predict(valXim)
print("model_2 RMSE,MAE =", eval_predict(valYim, model_2_predict))
write_csv_kaggle_sub("submissions/my_submission_model_2.csv", model_2.predict
```

```
model_2 RMSE,MAE = (21.230656647422844, 17.058412809371948)
```

- mode_3: tried to do feature extraction from ResNet152V2

In [54]:

```
# initialize random seed
K.clear_session()
random.seed(999); tf.random.set_seed(999)

# build mode3
model_3 = Sequential()
model_3.add(transferModel('ResNet152V2'))
model_3.add(BatchNormalization())
model_3.add(Flatten())
model_3.add(Dense(units=256, activation='relu'))
model_3.add(BatchNormalization())
model_3.add(Dense(units=128, activation='relu'))
model_3.add(Dense(units=64, activation='relu'))
# model_3.add(Dropout(0.2))
model_3.add(Dense(units=1, activation=tf.keras.layers.ReLU(max_value = 100)))

model_3.summary()
```

```
model transfer start--ResNet152V2
model transfer end--ResNet152V2
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet152v2 (Functional)     (None, 4, 4, 2048)        58331648
_________________________________________________________________
```

```
batch_normalization (BatchNo (None, 4, 4, 2048)        8192
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 256)               8388864
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 66,770,945
Trainable params: 8,434,689
Non-trainable params: 58,336,256
_________________________________________________________________
```

In [55]:

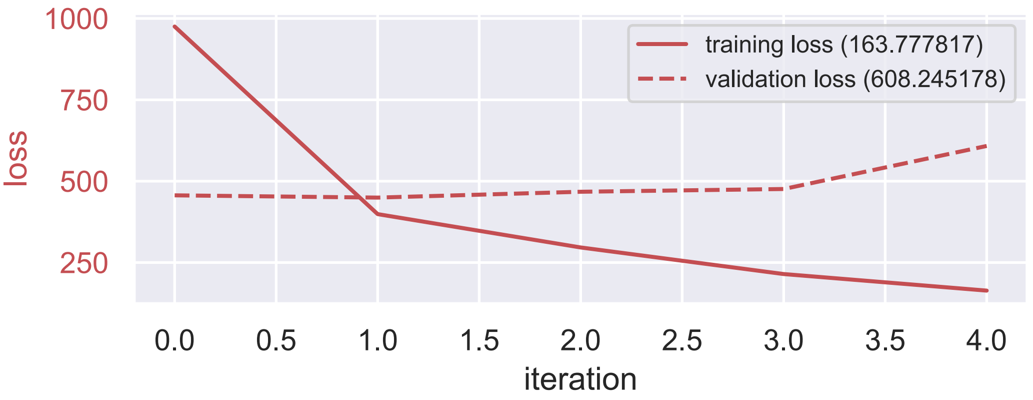
```
# early stopping criteria
earlystop = keras.callbacks.EarlyStopping(
              monitor='val_loss',
              min_delta=1, patience=3,
              verbose=1, mode='auto')
callbacks_list = [earlystop]

# compile and fit the network
model_3.compile( loss = keras.losses.MeanSquaredError(),
             optimizer=keras.optimizers.Adam(),
             metrics = ['mean_squared_error']
          )
train_x,val_x = data_process('ResNet152V2', 'add_gauss_noise',trainXim, valXi
history_model_3 = model_3.fit(train_x, trainYim, epochs=100, batch_size=100,
                 callbacks=callbacks_list,
                 validation_data=(val_x,valYim),  # specify the validation se
                 verbose=1)
```

```
preprocessInput start--ResNet152V2
preprocessInput end--ResNet152V2
preprocessInput start--ResNet152V2
preprocessInput end--ResNet152V2
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
Epoch 1/100
48/48 [==============================] - 484s 10s/step - loss: 975.2369 - mean
_squared_error: 975.2369 - val_loss: 456.5110 - val_mean_squared_error: 456.51
10
Epoch 2/100
48/48 [==============================] - 489s 10s/step - loss: 398.8136 - mean
_squared_error: 398.8137 - val_loss: 449.5734 - val_mean_squared_error: 449.57
33
Epoch 3/100
48/48 [==============================] - 489s 10s/step - loss: 296.3444 - mean
_squared_error: 296.3444 - val_loss: 467.5207 - val_mean_squared_error: 467.52
07
Epoch 4/100
48/48 [==============================] - 505s 11s/step - loss: 214.6590 - mean
_squared_error: 214.6590 - val_loss: 475.9548 - val_mean_squared_error: 475.95
48
Epoch 5/100
48/48 [==============================] - 533s 11s/step - loss: 163.7778 - mean
```

```
_squared_error: 163.7778 - val_loss: 608.2452 - val_mean_squared_error: 608.24
52
Epoch 00005: early stopping
```

In [56]:

```
plot_history(history_model_3)
```




In [57]:

```
model_3_predict=model_3.predict(valXim)
print("model_3 RMSE,MAE =", eval_predict(valYim, model_3_predict))
write_csv_kaggle_sub("submissions/my_submission_model_3.csv", model_3.predict
```

```
model_3 RMSE,MAE = (32.38418958619872, 29.42764493306478)
```

- mode_4: tried to do feature extraction from ResNet50

In [58]:

```
# initialize random seed
K.clear_session()
random.seed(999); tf.random.set_seed(999)

# build mode4
model_4 = Sequential()
model_4.add(transferModel('ResNet50'))
model_4.add(BatchNormalization())
model_4.add(Flatten())
model_4.add(Dense(units=256, activation='relu'))
model_4.add(BatchNormalization())
model_4.add(Dense(units=128, activation='relu'))
model_4.add(Dense(units=64, activation='relu'))
# model_4.add(Dropout(0.2))
model_4.add(Dense(units=1, activation=tf.keras.layers.ReLU(max_value = 100)))

model_4.summary()
```

```
model transfer start--ResNet50
model transfer end--ResNet50
Model: "sequential"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| resnet50 (Functional) | (None, 4, 4, 2048) | 23587712 |
| batch_normalization (BatchNo | (None, 4, 4, 2048) | 8192 |
| flatten (Flatten) | (None, 32768) | 0 |
| dense (Dense) | (None, 256) | 8388864 |
| batch_normalization_1 (Batch | (None, 256) | 1024 |

```
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896     
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65        
=================================================================
Total params: 32,027,009
Trainable params: 8,434,689
Non-trainable params: 23,592,320
_________________________________________________________________
```

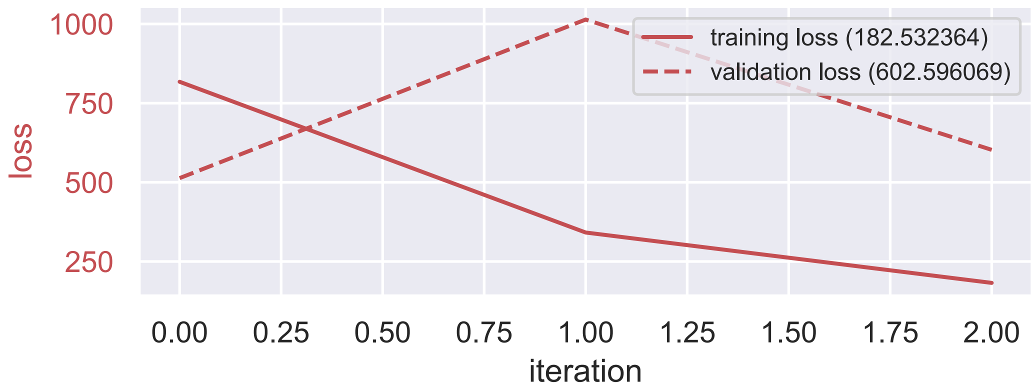
In [59]:

```
# early stopping criteria
earlystop = keras.callbacks.EarlyStopping(
              monitor='val_loss',
              min_delta=1, patience=2,
              verbose=1, mode='auto')
callbacks_list = [earlystop]

# compile and fit the network
model_4.compile( loss = keras.losses.MeanSquaredError(),
             optimizer=keras.optimizers.Adam(),
             metrics = ['mean_squared_error']
          )
train_x,val_x = data_process('ResNet50', 'add_corrupt_noise',trainXim, valXim
history_model_4 = model_4.fit(train_x, trainYim, epochs=100, batch_size=100,
                 callbacks=callbacks_list,
                 validation_data=(val_x,valYim),  # specify the validation se
                 verbose=1)
```

```
preprocessInput start--ResNet50
preprocessInput end--ResNet50
preprocessInput start--ResNet50
preprocessInput end--ResNet50
dataAugmentation start--add_corrupt_noise
dataAugmentation end--add_corrupt_noise
dataAugmentation start--add_corrupt_noise
dataAugmentation end--add_corrupt_noise
Epoch 1/100
48/48 [==============================] - 217s 4s/step - loss: 817.5500 - mean_
squared_error: 817.5500 - val_loss: 513.3467 - val_mean_squared_error: 513.346
7
Epoch 2/100
48/48 [==============================] - 214s 4s/step - loss: 341.5665 - mean_
squared_error: 341.5665 - val_loss: 1014.0956 - val_mean_squared_error: 1014.0
956
Epoch 3/100
48/48 [==============================] - 213s 4s/step - loss: 182.5324 - mean_
squared_error: 182.5324 - val_loss: 602.5961 - val_mean_squared_error: 602.596
1
Epoch 00003: early stopping
```

In [60]:

```
plot_history(history_model_4)
```




In [61]:
```
model_4_predict=model_4.predict(valXim)
print("model_4 RMSE,MAE =", eval_predict(valYim, model_4_predict))
write_csv_kaggle_sub("submissions/my_submission_model_4.csv", model_4.predict
```

```
model_4 RMSE,MAE = (31.414984160993658, 28.457210210164387)
```

- mode_5: tried to do feature extraction from EfficientNetB7

In [62]:
```
# initialize random seed
K.clear_session()
random.seed(999); tf.random.set_seed(999)

# build mode5
model_5 = Sequential()
model_5.add(transferModel('EfficientNetB7'))
model_5.add(BatchNormalization())
model_5.add(Flatten())
model_5.add(Dense(units=256, activation='relu'))
model_5.add(BatchNormalization())
model_5.add(Dense(units=128, activation='relu'))
model_5.add(Dense(units=64, activation='relu'))
# model_5.add(Dropout(0.2))
model_5.add(Dense(units=1, activation=tf.keras.layers.ReLU(max_value = 100)))

model_5.summary()
```

```
model transfer start--EfficientNetB7
model transfer end--EfficientNetB7
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
efficientnetb7 (Functional)  (None, 4, 4, 2560)        64097687  
_________________________________________________________________
batch_normalization (BatchNo (None, 4, 4, 2560)        10240     
_________________________________________________________________
flatten (Flatten)            (None, 40960)             0         
_________________________________________________________________
dense (Dense)                (None, 256)               10486016  
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024      
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896     
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65        
=================================================================
```

```
Total params: 74,636,184
Trainable params: 10,532,865
Non-trainable params: 64,103,319
```

In [63]:

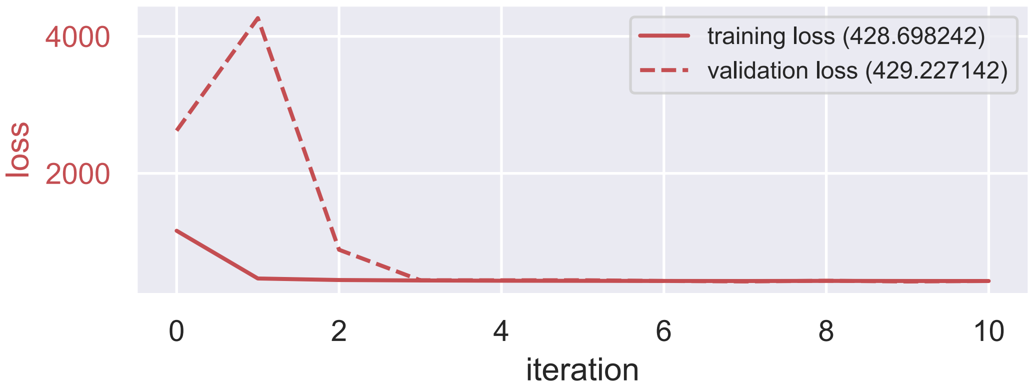
```
# early stopping criteria
earlystop = keras.callbacks.EarlyStopping(
              monitor='val_loss',
              min_delta=1, patience=3,
              verbose=1, mode='auto')
callbacks_list = [earlystop]

# compile and fit the network
model_5.compile( loss = keras.losses.MeanSquaredError(),
             optimizer=keras.optimizers.Adam(),
             metrics = ['mean_squared_error']
          )
train_x,val_x = data_process('EfficientNetB7', 'add_gauss_noise',trainXim, va
history_model_5 = model_5.fit(train_x, trainYim, epochs=100, batch_size=100,
                 callbacks=callbacks_list,
                 validation_data=(val_x,valYim),  # specify the validation se
                 verbose=1)
```

```
preprocessInput start--EfficientNetB7
preprocessInput end--EfficientNetB7
preprocessInput start--EfficientNetB7
preprocessInput end--EfficientNetB7
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
Epoch 1/100
48/48 [==============================] - 657s 13s/step - loss: 1162.4086 - mea
n_squared_error: 1162.4087 - val_loss: 2624.0459 - val_mean_squared_error: 262
4.0459
Epoch 2/100
48/48 [==============================] - 663s 14s/step - loss: 465.5693 - mean
_squared_error: 465.5693 - val_loss: 4266.1675 - val_mean_squared_error: 4266.
1675
Epoch 3/100
48/48 [==============================] - 657s 14s/step - loss: 443.2154 - mean
_squared_error: 443.2154 - val_loss: 885.1823 - val_mean_squared_error: 885.18
23
Epoch 4/100
48/48 [==============================] - 662s 14s/step - loss: 436.7862 - mean
_squared_error: 436.7862 - val_loss: 440.2504 - val_mean_squared_error: 440.25
04
Epoch 5/100
48/48 [==============================] - 648s 14s/step - loss: 433.0995 - mean
_squared_error: 433.0995 - val_loss: 439.3080 - val_mean_squared_error: 439.30
80
Epoch 6/100
48/48 [==============================] - 664s 14s/step - loss: 431.3997 - mean
_squared_error: 431.3997 - val_loss: 439.9918 - val_mean_squared_error: 439.99
18
Epoch 7/100
48/48 [==============================] - 653s 14s/step - loss: 429.7646 - mean
_squared_error: 429.7646 - val_loss: 429.8815 - val_mean_squared_error: 429.88
15
Epoch 8/100
48/48 [==============================] - 663s 14s/step - loss: 428.9112 - mean
_squared_error: 428.9112 - val_loss: 422.4149 - val_mean_squared_error: 422.41
49
```

```
Epoch 9/100
48/48 [==============================] - 645s 13s/step - loss: 430.7588 - mean
_squared_error: 430.7588 - val_loss: 432.8639 - val_mean_squared_error: 432.86
39
Epoch 10/100
48/48 [==============================] - 659s 14s/step - loss: 429.3492 - mean
_squared_error: 429.3492 - val_loss: 421.7596 - val_mean_squared_error: 421.75
96
Epoch 11/100
48/48 [==============================] - 669s 14s/step - loss: 428.6982 - mean
_squared_error: 428.6982 - val_loss: 429.2271 - val_mean_squared_error: 429.22
71
Epoch 00011: early stopping
```

In [64]:

```
plot_history(history_model_5)
```




In [65]:

```
model_5_predict=model_5.predict(valXim)
print("model_5 RMSE,MAE =", eval_predict(valYim, model_5_predict))
write_csv_kaggle_sub("submissions/my_submission_model_5.csv", model_5.predict
```

```
model_5 RMSE,MAE = (20.718224338984903, 16.092647444407145)
```

- mode_6: tried to do feature extraction from Xception

In [66]:

```
# initialize random seed
K.clear_session()
random.seed(999); tf.random.set_seed(999)

# build mode2
model_6 = Sequential()
model_6.add(transferModel('Xception'))
model_6.add(BatchNormalization())
model_6.add(Flatten())
model_6.add(Dense(units=256, activation='relu'))
model_6.add(BatchNormalization())
model_6.add(Dense(units=128, activation='relu'))
model_6.add(Dense(units=64, activation='relu'))
# model_6.add(Dropout(0.2))
model_6.add(Dense(units=1, activation=tf.keras.layers.ReLU(max_value = 100)))

model_6.summary()
```

```
model transfer start--Xception
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
ions/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 7s 0us/step
83697664/83683744 [==============================] - 7s 0us/step
```

```
model transfer end--Xception
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
xception (Functional)        (None, 4, 4, 2048)        20861480  
_________________________________________________________________
batch_normalization_4 (Batch (None, 4, 4, 2048)        8192      
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0         
_________________________________________________________________
dense (Dense)                (None, 256)               8388864   
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024      
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896     
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65        
=================================================================
Total params: 29,300,777
Trainable params: 8,434,689
Non-trainable params: 20,866,088
_________________________________________________________________
```

In [67]:

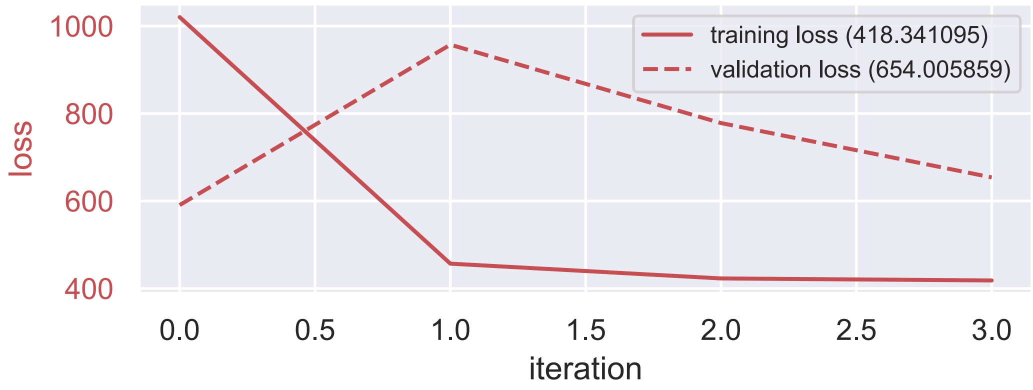
```
# early stopping criteria
earlystop = keras.callbacks.EarlyStopping(
              monitor='val_loss',
              min_delta=1, patience=3,
              verbose=1, mode='auto')
callbacks_list = [earlystop]

# compile and fit the network
model_6.compile( loss = keras.losses.MeanSquaredError(),
             optimizer=keras.optimizers.Adam(),
             metrics = ['mean_squared_error']
          )
train_x,val_x = data_process('Xception', 'add_gauss_noise',trainXim, valXim)
history_model_6 = model_6.fit(train_x, trainYim, epochs=100, batch_size=100,
                 callbacks=callbacks_list,
                 validation_data=(val_x,valYim),  # specify the validation se
                 verbose=1)
```

```
preprocessInput start--Xception
preprocessInput end--Xception
preprocessInput start--Xception
preprocessInput end--Xception
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
dataAugmentation start--add_gauss_noise
dataAugmentation end--add_gauss_noise
Epoch 1/100
48/48 [==============================] - 286s 6s/step - loss: 1020.5333 - mean
_squared_error: 1020.5332 - val_loss: 590.7047 - val_mean_squared_error: 590.7
048
Epoch 2/100
48/48 [==============================] - 285s 6s/step - loss: 456.7571 - mean_
squared_error: 456.7571 - val_loss: 957.4197 - val_mean_squared_error: 957.419
7
Epoch 3/100
48/48 [==============================] - 247s 5s/step - loss: 422.9342 - mean_
squared_error: 422.9342 - val_loss: 778.2110 - val_mean_squared_error: 778.211
```

```
0
Epoch 4/100
48/48 [==============================] - 239s 5s/step - loss: 418.3411 - mean_
squared_error: 418.3411 - val_loss: 654.0059 - val_mean_squared_error: 654.005
9
Epoch 00004: early stopping
```

In [68]:

```
plot_history(history_model_6)
```




In [69]:

```
model_6_predict=model_6.predict(valXim)
print("model_6 RMSE,MAE =", eval_predict(valYim, model_6_predict))
write_csv_kaggle_sub("submissions/my_submission_model_6.csv", model_6.predict
```

```
model_6 RMSE,MAE = (27.282674789492592, 24.244772679011028)
```

## Generate final result

- As the errors of model_5 and model_1, are the most minor and similar, we could ensemble the results. As the error of model_5 more was a little better than model_1, we gived model_5 a little more weight.

In [76]:

```
val_final_predict=model_1_predict*0.4+model_5_predict*0.6
print("final RMSE,MAE =", eval_predict(valYim, val_final_predict))
```

```
final RMSE,MAE = (20.669306288846442, 15.905551042556763)
```

In [77]:

```
write_csv_kaggle_sub("submissions/my_submission_final.csv",model_1.predict(te
```

## Conclusion and Remarks

- This is a perfect and fruitful project. In this project, we got the chance to experience a general machine learning workflow: formulate the question, exploratory data analysis, train different models, and finally generate a final result.

- We first analyzed the metadata and guessed that the results predicted with the metadata would not be perfect. Then we tried to extract features from the raw image and used a deep learning model for training. We found that the model we constructed by ourselves had a large RMSE, even inferior to the results obtained by traditional machine learning methods. Finally, we tried to use transfer learning for training, we got better results and made the final prediction using bagging.

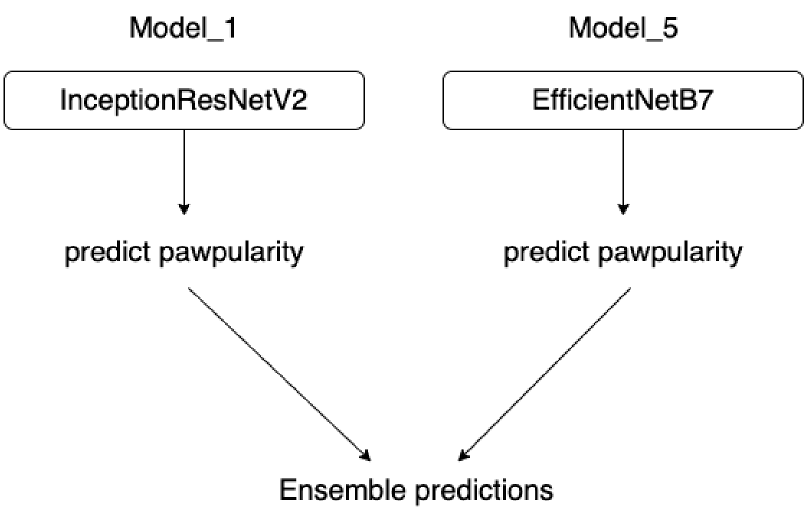
- Based machine learning models
  - Linear Regression
    - Ordinary Least Squares
    - Ridge Regression
    - LASSO
  - Non-Linear Regression
    - Kernel Ridge Regression
    - Support Vector Regression
    - Random Forest Regression
  - PCA
  - Kmeans Cluster
- Deep learning models
  - MLP
  - We want to try CNN but our computer does not allow us to do so :), the CNN model cannot be successfully excuted.
- Transfer learning models
  - ResNet152V2
  - ResNet50
  - InceptionResNetV2
  - DenseNet201
  - EfficientNetB7
  - Xception

- Due to equipment limitations, we cannot use all the raw pictures when performing transfer learning. In the case of less training data we could use to train on our computer, we tried to add the noise to increase the stability and prediction accuracy of the model. We used four different noises, and fianlly found that Gauss Noise was better.

  - Noise Type
    - No Noise
    - Guass Noise
    - Corrupt Noise
    - Scale Shift Noise
- The standard We used to measure the quality of the model was to evaluate the root mean squared error (RMSE), and the mean absolute error (MAE) is also a reference. The details of results are below.

| MODEL | RMSE | MAE |
|---|---|---|
| OLS | 20.73824764611121 | 15.76785972406625 |
| Ridge Regression | 20.743712171549397 | 15.756259950364006 |
| LASSO | 20.75573688225275 | 15.767690402140891 |
| Kernel Ridge Regression | 20.717123104065067 | 15.736461514585494 |
| Support Vector Regression | 20.817379745431104 | 15.129932661840236 |
| Random Forest Regression | 20.716464671286573 | 15.739780845467504 |
| PCA | 20.718815193707346 | 15.747754263136668 |
| Kmeans Cluster | 37.62706965693371 | 31.336693548387096 |
| MLP | 21.64063389579775 | 15.9560591562589 |
| Model_1(InceptionResNetV2) | 20.965820018274336 | 15.996552220980327 |
| Model_2(DenseNet201) | 21.230656647422844 | 17.058412809371948 |
| Model_3(ResNet152V2) | 32.38418958619872 | 29.42764493306478 |
| Model_4(ResNet50) | 31.414984160993658 | 28.457210210164387 |
| Model_5(EfficientNetB7) | 20.718224338984903 | 16.092647444407145 |
| Model_6(Xception) | 27.282674789492592 | 24.244772679011028 |

- From the table above, we found that the model_1 and model_5 had lower and similar RMES and MAE, so we used the average results of these two models as the final prediction, reducing error by bagging. The following is the overview workflow of our final model.




- Although the errors of traditional machine learning and transfer learning are similar, we still use the prediction results of the latter because the data in the metadata is not reliable. Many data with the same feature value but different pawpularity values will bring a lot confusion to the model.

- Overall, In this project, we felt the importance of feature extraction, especially when the amount of data is tremendous but the available resources are small, accurately extracting the most critical features will significantly reduce running time and error. We have practiced various traditional machine learning models and deep learning models. This processing helped us better understand these models(how to use and the principle behind the model). At the same time, we found that we need to further study the theoretical knowledge behind transfer learning. Now we are still in the stage of calling available models. We do not have a comprehensive understanding of the mathematical knowledge behind it. This also makes us not flexible and confident enough when calling.

In [ ]:

```

```