CARLSON SCHOOL
OF MANAGEMENT
UNIVERSITY OF MINNESOTA

PetFinder.my - Pawpularity Contest
Project Report

Predict the popularity of shelter pet photos

Norah Lu, Skylar Pu, Weijing Luo, Jiaqing Zhang

MSBA 6421
Dr. Yicheng Song

Table of Content

I. Business Context & Project Goal

The timeless saying, "A picture is worth a thousand words," extends beyond art and storytelling, especially in animal welfare. Stray animals face uncertainty, navigating city landscapes or the harsh reality of over-crowded shelters. A single image captures their cuteness and the potential to bring joy to human lives.

Petfinder.my is Malaysia's leading animal welfare platform It contains thousands of pet photos uploaded by animal lovers, media, corporations, and global organizations.

In this project, our objective is to develop a predictive model, that is capable of accurately estimating the 'Pawpularity' of pet photos. The model's reliability is being assessed based on the Root Mean Square Error (RMSE) metric based on the dataset from PetFinder.my.

II. Technical Specifications & Solution Overview

2.1 Tool Used

This project is primarily executed using **Python 3.10.12**, encompassing a range of activities including data cleaning, image loading, model training, testing, and ensembling. The software packages utilized in this project are as follows:

- Pandas: Version 1.5.3
- NumPy: Version 1.23.5
- Scikit Learn: Version 1.2.2
- Torch: Version 2.1.0
- Timm: Version 0.9.12
- Ftfy: Version 6.0.2

2.2 Dataset Overview

All data used for this analysis are provided by the PetFinder.my. The **photos** are all in jpg format, themed around dogs and cats, each has a unique profile ID that corresponds to the photo's file name. Additionally, we are also provided with hand-labeled **meta-data** for each photo in CSV format. Each row shows the profile ID and links to key visual quality and composition parameters, such as focus, eyes, and face. However, it is important to note that in the original calculation of **Pawpularity**, our target variable, these meta-data are not utilized in the derivation process.

The dataset for this project therefore comprises both images and tabular data and is organized as follows:

1. **Training Photos (train/)**: Folder containing training set photos of the form {id}.jpg, where {id} is a unique Pet Profile ID.

2. **Training Metadata (train.csv)**: Metadata for each photo in the training set as well as the target, the photo's Pawpularity score. The Id column gives the photo's unique Pet Profile ID corresponding to the photo's file name.

    a. **Focus** - Pet stands out against an uncluttered background, not too close / far.
    b. **Eyes** - Both eyes are facing front or near-front, with at least 1 eye/pupil decently clear.
    c. **Face** - Decently clear face, facing front or near-front.
    d. **Near** - Single pet taking up a significant portion of the photo (roughly over 50% of photo width or height).
    e. **Action** - Pet in the middle of an action (e.g., jumping).
    f. **Accessory** - Accompanying physical or digital accessory/prop (i.e. toy, digital sticker), excluding collar and leash.
    g. **Group** - More than 1 pet in the photo.
    h. **Collage** - Digitally-retouched photo (i.e. with digital photo frame, a combination of multiple photos).
    i. **Human** - Human in the photo.
    j. **Occlusion** - Specific undesirable objects blocking part of the pet (i.e. human, cage, or fence). Note that not all blocking objects are considered occlusion.
    k. **Blur** - Noticeably out of focus or noisy, especially for the pet's eyes and face. For Blur entries, the "Eyes" column is always set to 0.

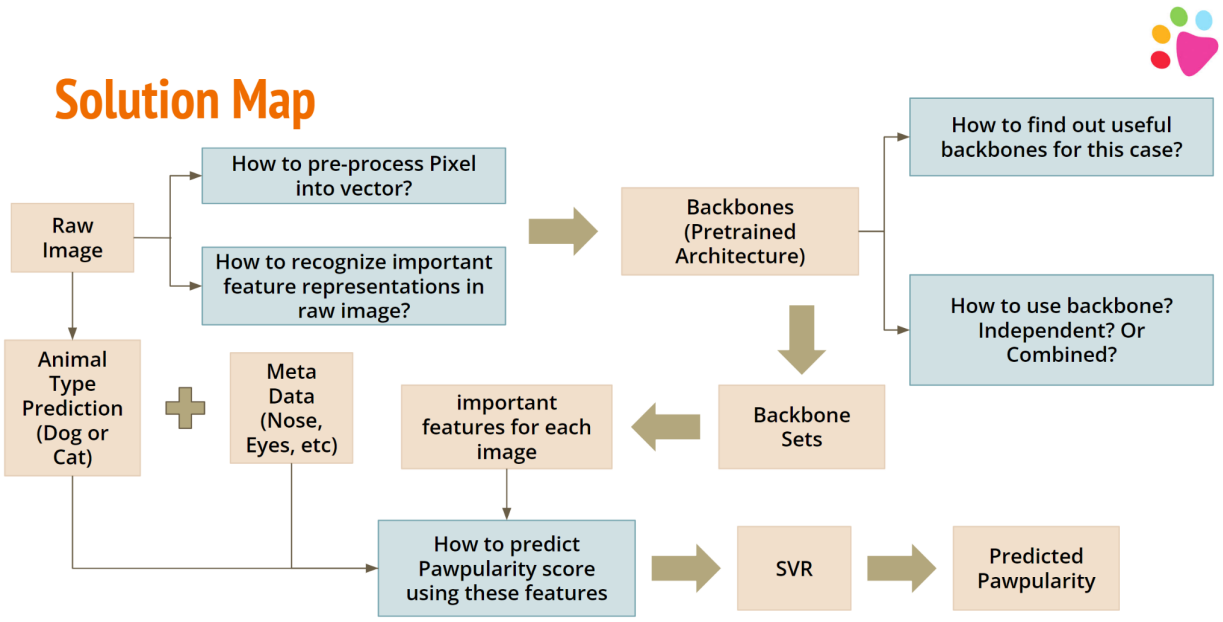
3. **Test Photos (test/)**: Folder containing randomly generated images in a format similar to the training set photos. The actual test data comprises about 6800 pet photos similar to the training set photos.

4. **Test Metadata (test.csv)**: Metadata for each photo in the testing set

5. **Sample Submission (sample_submission.csv)**: A sample submission file in the correct format.

2.3 Solution Structure

As the solution map is shown below, our solution structure could mainly focus on 3 parts.



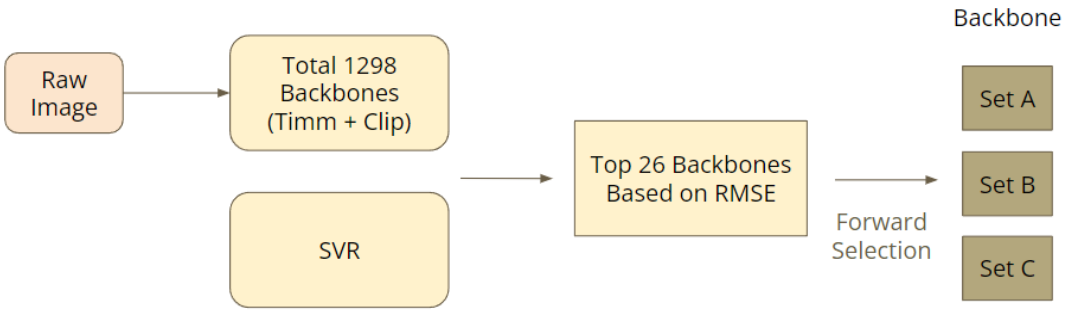
Firstly, three sets of selected backbones are employed. Backbone typically refers to a pre-trained Convolutional Neural Network (CNN) Architecture used for feature extraction from raw images. The backbone is instrumental in understanding complex patterns in the data and significantly reduces the effort required to recognize important features in the raw image. We find a subset of backbones that perform well in terms of RMSE, from all available models and then combine them into three backbone sets effectively.

Secondly, we utilize meta-data (such as the presence of a nose, eyes, etc.) along with the classification results of the animal type (whether the animal is a dog or a cat). The inclusion of meta-data and animal-type prediction allows the model to consider additional, non-visual information that may be necessary to predict popularity accurately.

Lastly, we ensemble four Support Vector Regressors (SVR) models to get the final prediction. The ensembling method combines the predictions of several machine learning algorithms to make more accurate predictions than any individual model. By employing multiple SVRs, we are able to exploit the unique strengths of each model, potentially improving the robustness and accuracy of the predicted Pawpularity score.

III. Ensembling SVR models with Multiple Pre-trained CNN Architectures on Pytorh

3.1. Select and Combine Useful Pre-trained CNN Architectures



To deal with image data, the transformation of raw images into a structured format that a machine learning algorithm can interpret is a fundamental step. This transformation is known as feature extraction, where the raw pixel data is processed and distilled into a vector of features that capture the essence of the image.

To facilitate this process, we leverage a suite of sophisticated tools known as backbones. These are pre-trained convolutional neural network (CNN) architectures that have been trained on large datasets. When we input a raw image into such a model, it traverses through multiple layers of the network, each designed to recognize increasingly complex patterns. The final output is a vector—a condensed representation of the original image, now encoded with the most salient features that the network has learned to recognize.

However, there are 1298 pre-trained model architectures available in both Timm and clip library, and only partial models are helpful in our case.

The first part of the solution is basically to find a subset of backbones that perform well and use them effectively.

3.1.1 Prepare Image Path

The initial step involves loading data into our environment. We use the **get_image_id** and **get_image_paths** functions to retrieve the id and file paths of images based on their meta-data.

```
def get_image_id(meta_path):
    df_meta = pd.read_csv(meta_path)
    image_id = df_meta['Id'].values

    return image_id
```

```
def get_image_paths(meta_path, image_folder):
    df_meta = pd.read_csv(meta_path)
    image_paths = [os.path.join(image_folder, f"{id}.jpg") for id in df_meta['Id'].values]

    return image_paths
```

3.1.2 Extract Features for Raw Images

At this step, we define the **extract_and_export_features** function to extract features using all available backbones for each raw image.

```
def extract_and_export_features(image_id, image_paths, output_folder):

    device = "cuda" if torch.cuda.is_available() else "cpu"

    # extract features
    for j in range(len(image_paths)):
      for backbone in BACKBONES:
        # get model
        model = timm.create_model(backbone, pretrained=True, num_classes=0)
        model.eval()
        # get transform
        config = resolve_data_config({}, model=model)
        transform = create_transform(**config, is_training=False)
        # preprocess image
        image = transform(Image.open(image_paths[j])).unsqueeze(0)
        # get features
        with torch.no_grad():
          features = model(image).detach().numpy()[0]
```

3.1.3 Use SVR to Predict Pawpularity for Individual Backbone

Following this, we use Support Vector Regression (SVR) to get Pawpularity score prediction. Since predicting Pawpularity scores is a regression task, where the aim is to estimate a continuous numerical value, SVR can effectively handle this by focusing on getting a hyperplane that best represents the relationship between the input features and the target variable. Moreover, to capture non-linear relationships between image features

and the Pawpularity scores, SVR uses kernel functions to map the input data into a higher-dimensional space, where linear regression can be applied effectively.

We create the **get_X_y**, **train_svr**, and **train_svr_using_extracted_features** functions to evaluate backbone performance through the RMSE score.

```
def get_X_y(combined_features, meta_path):

  # initialize df
  df = pd.read_csv(meta_path)

  # combine meta and features
  df = pd.concat([df, combined_features], axis=1)

  # return X and y for train
  X = df.drop(['Id', 'Pawpularity', 'Image_Id'], axis=1)
  X.columns = X.columns.astype(str)
  y = df['Pawpularity']

  return X, y
```

```
def train_svr(X_train, y_train):
    svr_model = SVR(C=16.0, kernel='rbf', degree=3, max_iter=4000, output_type='numpy')
    svr_model.fit(X_train, y_train)
    return svr_model
```

```python
def train_svr_using_extracted_features(meta_path, output_folder, image_ids):

  # initialize backbone_rmse_dict
  backbone_rmse_dict = {}

  # iterate over each backbone
  for backbone in BACKBONES:

    # iterate over each image
    for i in range(len(image_ids)):

      img_id = image_ids[i]
      img_df = pd.DataFrame([img_id], columns=['Image_Id'])

      # find the file
      if '/' in backbone:
        backbone = backbone.replace("/", "-")

      feature_file_path = os.path.join(output_folder, f"{img_id}_{backbone}.csv")
      img_features = np.loadtxt(feature_file_path, delimiter=',')

      # Convert the features list to a pandas DataFrame
      feature_df = pd.DataFrame(img_features).T

      # combine img_df and feature_df as one row
      img_df = pd.concat([img_df, feature_df], axis=1)
```

```
      # add the row to whole df
      if i == 0:
        # initialize combined_features_df
        backbone_df = img_df
      elif i > 0:
        backbone_df = backbone_df.append(img_df, ignore_index = True)

    # define X, y
    X, y = get_X_y(backbone_df, meta_path)

    # train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # train SVR for the backbone
    model = train_svr(X_train, y_train)

    # get y_pred
    y_pred = model.predict(X_test)

    # calculate RMSE
    backbone_rmse = rmse(y_test, y_pred)

    # save in dic
    backbone_rmse_dict[backbone] = backbone_rmse

# transfer dictionary to pandas dataframe
backbone_rmse_df = pd.DataFrame.from_dict(backbone_rmse_dict, orient='index')

# save the results
backbone_rmse_df.to_csv(f"{path_working}backbone_rmse.csv", index=True)
```

3.1.4 Select 26 Backbones with Low RMSE

The RMSE is a standard measure used to evaluate the differences between values predicted by a model and the values observed. A low RMSE indicates that the backbone architecture is able to extract features that lead to accurate predictions when used in subsequent machine learning models.

After we calculate the RMSE for each backbone, we select 26 backbones from CLIP and Timm libraries with the lowest RMSE values that serve as the most effective backbones for our case.

No.	Architecture	Feature Dimension	RMSE
1	tf_efficientnet_l2_ns_475	1000	17.56
2	tf_efficientnet_l2_ns_475_512	1000	17.57
3	tf_efficientnet_l2_ns_475_hflip_384	1000	17.62
4	clip_RN50x16	768	17.59

5	ig_resnext101_32x48d	1000	17.63
6	ig_resnext101_32x48d_hflip_384	1000	17.66
7	clip_ViT-B-16	512	17.65
8	tf_efficientnet_b6_ns	1000	17.66
9	tf_efficientnet_b7_ns	1000	17.67
10	deit_base_distilled_patch16_384	1000	17.68
11	deit_base_distilled_patch16_384_hflip_384	1000	17.66
12	tf_efficientnet_b8_ap	1000	17.70
13	clip_RN50x4	640	17.75
14	ig_resnext101_32x8d	1000	17.82
15	vit_base_patch16_384	1000	17.82
16	clip_ViT-B-32	512	17.84
17	vit_large_patch16_384	1000	17.89
18	resnest269e	1000	17.90
19	swsl_resnext101_32x8d	1000	17.91
20	vit_large_r50_s32_384	1000	17.92
21	clip_RN101	512	17.93
22	rexnet_200	1000	17.96
23	resnetv2_152x4_bitm	1000	18.06
24	clip_RN50	1024	18.11
25	repvgg_b0	1000	18.21
26	fbnetc_100	1000	18.52

3.1.5 Combine Backbones using Forward Selection

Forward Selection is an iterative method employed to refine a predictive model by judiciously combining a suite of neural network architectures known as backbones. The process begins with an empty set and methodically adds backbones, each selected for its proven ability to decrease the Root Mean Square Error (RMSE), thus enhancing model performance.

The selection commences by evaluating the standalone impact of each backbone on RMSE. The most effective backbone is chosen first, laying the foundation for the model. Subsequently, backbones are added sequentially, with each new inclusion thoroughly assessed for its performance contribution. This approach

harnesses the collective strengths of various architectures, potentially unlocking synergistic effects that bolster prediction accuracy.

The incremental nature of Forward Selection is beneficial as it discerns the most impactful combination of backbones, focusing on their collaborative performance rather than individual merit. It can expose beneficial interactions between different feature sets that might remain concealed in solitary evaluations.

```
# Foward Selection to get three sets of backbones
for i in range(3):

    # Start using the best architecture
    backbone_list = ['tf_efficientnet_l2_ns_475']

    # initialize
    next_choice = None
    bestRMSE = np.inf
    currentRMSE = 0

    # forward selection (keep adding models while rmse decreases)
    while currentRMSE < bestRMSE:
        bestRMSE = currentRMSE
        backbone_list.append(next_choice)
        rmse_scores= [compute_SVR_RMSE(backbone_list + [feat]) for feat in top_backbones]
        currentRMSE = np.min(rmse_scores)
        best_feat = np.argmin(rmse_scores)

    print("Set", i ,":", backbone_list)
```

After the forward selection process, backbones are distributed into three distinct sets as shown below:

Set Name	Components
SetA	'RN50x16', 'ViT-B/32', 'ViT-B/16', 'RN50x4', 'Deit_base_distilled_patch16_384', 'ig_resnext101_32x48d', 'repvgg_b0', 'resnetv2_152x4_bitm', 'swsl_resnext101_32x8d', 'tf_efficientnet_l2_ns_475', 'vit_base_patch16_384','vit_large_r50_s32_384'
SetB	'tf_efficientnet_l2_ns','deit_base_distilled_patch16_384','ig_resnext101_32x48d', 'ig_resnext101_32x48d', 'vit_large_r50_s32_384', 'RN50x4', 'ViT-B/16', 'RN50x16', 'ViT-B/32'
SetC	'Beit_large_patch16_224','swin_large_patch4_window12_384_in22k','beit_large_patch16_224','tf_efficientnet_b6_ns' , 'swin_large_patch4_window7_224'

3.2 Utilize Meta Data and Classification Information

3.2.1 Advantages of Integrating Meta Data and Classification Information

The inclusion of meta-data and animal-type prediction allows the model to consider additional, non-visual information that may be necessary to predict popularity accurately. For example, what someone finds cute in a dog might be different from what comes to mind when thinking about a cat

Metadata (binary)

Focus	Face	Eyes	Near	Action	Group
0	1	1	1	0	0

Classification result (Dog as [0,1], Cat as [1,0])

- Cat [1, 0]

To improve the overall performance of our model, we incorporate this additional information as part of the input for our Pawpularity prediction.

3.2.2 Use Pre-trained Classification Model to Predict Dog or Cat

In this step, we load a pre-trained customed model **cat_dog_classification**, which is specifically designed for classifying dogs and cats for pet images.

```
# set the model path and load the model
model_path = path_working+'cat_dog_classification.pt'
classification_model = TheModelClass(*args, **kwargs)
```

Then we use the classification model to predict the animal type (whether the image contains a dog or a cat) for each image.

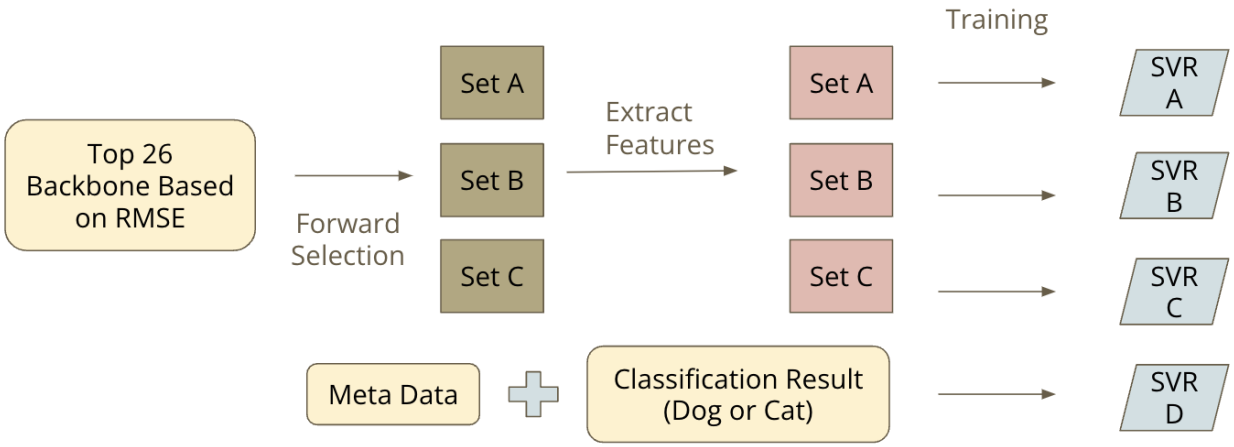
```
# initialize dic
image_classification = {}

# get prediction for each image
for i in range(len(image_id)):

  # get image id and file path
  id = image_id[i]
  path = image_paths[i]

  # open image
  image = transform(Image.open(path)).unsqueeze(0).to(device)

  # get prediction
  pred = classification_model.predict(image)

  # save the results
  image_classification[id] = pred
```

3.2.3 Combine Metadata with Classification Results

To integrate our meta-data with the classification results, we concatenate two data frames: **meta-_df** (containing meta-data features) and **image_classification_df** (containing classification results). This combined data frame, named "df," now serves as a consolidated dataset with additional columns, ready for further analysis and modeling.

```
train_meta_path = path_input+files['train_meta']
image_classification_df = pd.DataFrame.from_dict(image_classification, orient='index')
meta_df = pd.read_csv(train_meta_path)
df = pd.concat([meta_df, image_classification_df], axis=0)
# df.head()
```

3.3 Train SVR for Three Finalized Backbone Sets and Metadata-Classification Set



3.3.1 SVR for Backbone Sets

So far, we have three sets of backbone architectures, and we are planning to apply SVR to each of these sets individually. For each of these sets, SVR will generate a prediction.

In this process, we create the **get_combine_features** function to concatenate extracted features from multiple backbones in the same set.

```
# Combine Features Based on Set
def get_combine_features(set_num, output_folder):

  # Get the list of image ids from the output folder
  image_ids = [filename.split('_')[0] for filename in os.listdir(output_folder) if filename.endswith('.csv')]
  image_ids = list(set(image_ids))  # Remove duplicates

  # iterate over each image
  for i in range(len(image_ids)):

    img_id = image_ids[i]
    img_df = pd.DataFrame([img_id], columns=['Image_Id'])

    # initialize
    img_features = []

    # Iterate over each set in the set_backbones dictionary
    for backbone in set_num:
      # find the file
      if '/' in backbone:
        backbone = backbone.replace("/", "-")
      feature_file_path = os.path.join(output_folder, f"{img_id}_{backbone}.csv")
      features = np.loadtxt(feature_file_path, delimiter=',')
      # concatenate
      img_features.extend(features)

    # Convert the features list to a pandas DataFrame
    feature_df = pd.DataFrame(img_features).T

    # combine img_df and feature_df as one row
    img_df = pd.concat([img_df, feature_df], axis=1)

    # add the row to whole df
    if i == 0:
      # initialize combined_features_df
      combined_features_df = img_df
    elif i > 0:
      combined_features_df = combined_features_df.append(img_df, ignore_index = True)

  return combined_features_df
```

We iterate sets A, B, and C to train SVR and save the model. Now taking SVR A as an instance:

SVR A

```
[ ] # get extraced features using setA
    set_A_extracted_features = get_combine_features(setA, output_folder)
```

```
[ ] # prepare X and y for training
    X, y = get_X_y(set_A_extracted_features, train_meta_path)
```

```
[ ] # train SVR using all datasets
    modelA = train_svr(X, y)

    # save model
    with open(f'{path_working}modelA.pkl','wb') as f:
      pickle.dump(modelA, f)
```

```
[ ] # get prediction
    ypredtrainA = modelA.predict(X)
```

```
[ ] del set_A_extracted_features, X, y, modelA
    gc.collect()
```

3.3.2 SVR for Metadata-Classification

In this step, we apply the SVR model to set D, using the input obtained by merging the Meta Data with the Classification Result.

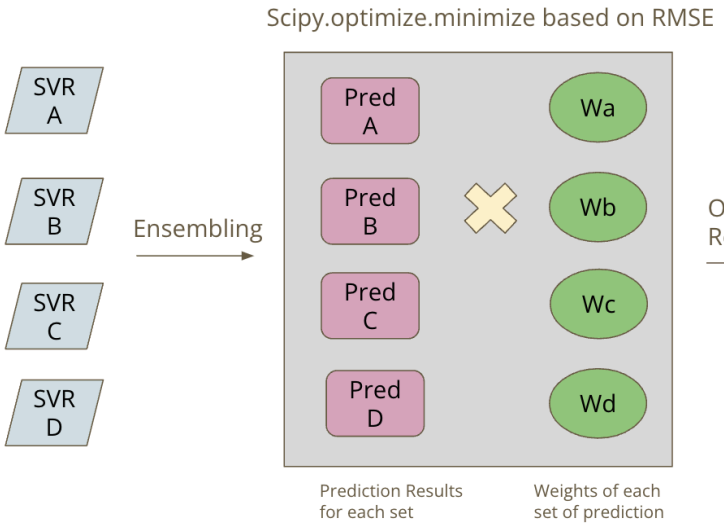
```
# generate X, y
X = df.drop(['Id', 'Pawpularity'], axis=1)
X.columns = X.columns.astype(str)
y = df['Pawpularity']
```

```
# train SVR D
modelD = train_svr(X, y)

# save model
with open(f'{path_working}modelD.pkl','wb') as f:
  pickle.dump(modelD, f)
```

```
# get prediction
ypredtrainD = modelD.predict(X)
```

3.4 Ensemble SVR to Get Final Prediction

In this step, we focus on ensembling SVR models A, B, C, and D to optimize the overall Root Mean Square Error (RMSE). The ensembling method combines predictions (Pred A, Pred B, Pred C, Pred D) from each SVR model, weighted by Wa, Wb, Wc, and Wd, respectively, to produce final output results. The optimization of weights for the predictions is performed based on minimizing the overall RMSE.



Id	Pawpularity
4128bae22183829d2b5fea10effdb0c3	37.660728965735856
43a2262d7738e3d420d453815151079e	37.79244580322117
4e429cead1848a298432a0acad014c9d	37.85309144466342
80bc3ccafcc51b66303c2c263aa38486	37.47744866636911
8f49844c382931444e68dffbe20228f4	37.77763117139125
b03f7041962238a7c9d6537e22f9b017	37.719434249016196
c978013571258ed6d4637f6e8cc9d6a3	37.6177015666982
e0de453c1bffc20c22b072b34b54e50f	37.6571715160411

We define the **min_func** function to optimize the assignment of the weight. The initial guess for the weights is [1/4]*4, implying an equal weighting. The optimization method used is 'Nelder-Mead', and a tolerance of 1e-6 is set for the convergence of the algorithm.

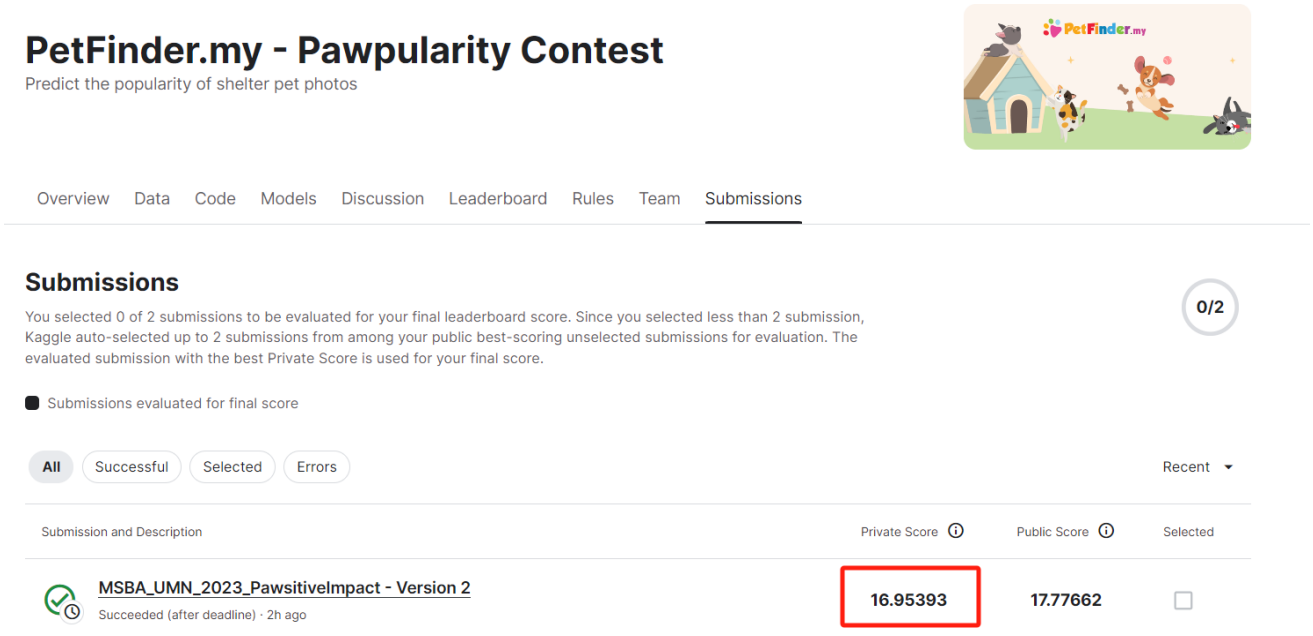
```
def min_func(K):
    ypredtrain = K[0]*ypredtrainA + K[1]*ypredtrainB + K[2]*ypredtrainC + K[3]*ypredtrainD
    return rmse(train.Pawpularity, ypredtrain)

res = minimize(min_func, [1/4]*4, method='Nelder-Mead', tol=1e-6)
K = res.x
res
```

We then apply the optimized weight to generate the predictions for the 'testing dataset and fill in the 'Pawpularity' score for each record.

```
# assign optimal weights to four SVR
test['Pawpularity'] = K[0]*ypredtestA + K[1]*ypredtestB + K[2]*ypredtestC + K[3]*ypredtestD
```

IV. Summary

Our model is ranked #18th in the Kaggle Private Leaderboard with RMSE 16.95393.



It goes beyond a traditional single CNN framework for the following reasons:

- Diversified Architecture: We opt for a more dynamic approach using multiple backbones and heads. This diversification allows us to capture various aspects of the data, enhancing the overall performance. We also employ techniques like Forward Selection to choose the combination of backbones that work best together, guided by performance metrics like RMSE. This ensures that we're not just adding complexity but enhancing the model's predictive capabilities with each addition. By integrating a variety of pre-trained backbones into sets, we aim to synergize the strengths of each model. The result is a robust final model that delivers consistent performance, making it adept at handling the complexity and variability inherent in real-world data.
- Beyond Raw Images: We incorporate meta-data and predictions about animal types into our model. This approach acknowledges that context matters, and by including this additional information, we empower the model with a more nuanced understanding of the subject matter, leading to richer feature extraction and improved accuracy.
- Ensembling for Excellence: By ensembling multiple SVRs, we can capture the unique strengths of each model that significantly improve overall model performance and robustness

In summary, our model architecture is not just a collection of parts but a carefully orchestrated symphony of

strategies, each playing a critical role in achieving a harmonious and high-performing machine-learning solution.

V. Appendix

5.1. Technical Deliverable Brief

- 1_get_RMSE_for_all_availble_backbones: find a subset of 26 useful backbones, that performs well in terms of RMSE, out of 1298 available in timm and clip libraries
- 2_select_and_combine_useful_backbones: combine top 26 backbones effectively as three backbone sets using forward selection
- 3_train_SVR_and_get_weights: combine features extracted from backbones with meta data and animal classification result to predicted pawpularity score through 4 SVR models and optimize the weights based on RMSE
- 4_extract_features_for testing_data: extract and save features for all testing data using top 26 backbones we have found
- 5_get_prediction_of _testing_data: combine extracted features for testing data and load trained SVR models and weight to get the final prediction

5.2 References & Others

Kaggle Community. (2023, Month Day).1st Place - Winning Solution - Full Writeup. Petfinder Pawpularity Score Competition. https://www.kaggle.com/c/petfinder-pawpularity-score/discussion/301686