Petfinder ‘Pawpularity’ -

Predicting the appeal of pet images

CARLSON SCHOOL
OF MANAGEMENT
UNIVERSITY OF MINNESOTA

MSBA 6420 - Predictive Analytics - Team Project

Abhishikth Peri, Amulya Konda, Kasturi Pal, Gopala Venkata Sandeep Gangisetty, Sharmistha Patra

TABLE OF CONTENTS

1. Business context

A photograph speaks volumes where words may falter. But how can one grasp its true essence amongst a myriad of them? This is an especially relevant dilemma in the world of animal welfare where several scores of stray animals suffer due to lack of a proper home.

PetFinder.my is a leading Malaysian leading animal welfare platform that aims to leverage data science and artificial intelligence to enhance the adoption rates of stray animals. It wishes to leverage data science to better understand and determine the appeal and attractiveness of adoption photos posted on its platform so as to bolster the chances of adoption. Poised at the intersection of technology, data and welfare, this organization aims for the following -

1) Animal welfare - The primary mission is to improve the lives of stray animals by increasing their chances of adoption. Through data-driven insights, the platform seeks to optimize the presentation of pet profiles, making them more intriguing to potential adopters.
2) Platform Engagement - By implementing advanced predictive algorithms for assessing pet photo appeal, PetFinder.my aims to boost user engagement on its platform. More engaging and attractive pet profiles can lead to increased time spent by users, higher click-through rates, and ultimately more successful adoptions.

2. Project scope

Petfinder.my uses a cuteness ranking meter to rank the appeal of pet photos. While it does a good job from analyzing photo composition and texture to comparing with several thousands of pet pictures, it is still a basic tool with a scope for improvement. We are tasked with developing a reliable prediction algorithm through analysis of meta and image data to determine the features that are deterministic in understanding the attractiveness of a photo to the user population. We then use these features to build a prediction algorithm to predict how well a photo can stimulate the interest of the user population, which can directly contribute to increased adoption rates.

Financial sustainability - The success of this initiative could potentially attract more donations and funding from individuals, corporations, and organizations that support animal welfare. The positive impact on adoption rates may contribute to the financial sustainability of PetFinder.my.

Competitive edge - Employing advanced algorithms can help Petfinder.my establish itself as a leader in leveraging technology for animal welfare. This can enhance the platform's reputation and attract more users, contributors, and partners who are aligned with the mission of improving animal lives.

Global collaboration - PetFinder.my collaborates with various stakeholders, including animal lovers, media, corporations, and global organizations. This initiative presents an opportunity for global collaboration, as the improved algorithm could be shared with shelters and rescuers worldwide, fostering a collective effort to address the issue of stray animal overpopulation.

3. Technical Specifications

Programming Language: Python
Deep Learning Framework: TensorFlow, PyTorch
Machine Learning Library: Scikit-learn
Data Manipulation and Analysis: Pandas
Numerical Operations: NumPy
Data Visualization: Matplotlib, Seaborn
Image Processing: TensorFlow's ImageDataGenerator
Model Training: Adam optimizer, Mean Squared Error (MSE) loss function
Data Augmentation: ImageDataGenerator for random transformations on training images

4. Data Exploration

Along with the images of pets, metadata of the images is also available, for predicting purposes.Metadata - This is one section of data which hosts information about various parameters of the pet photo images.

4.1 Photo Metadata

Each pet photo is labeled with the value of **1** (Yes) or **0** (No) for each of the following features:

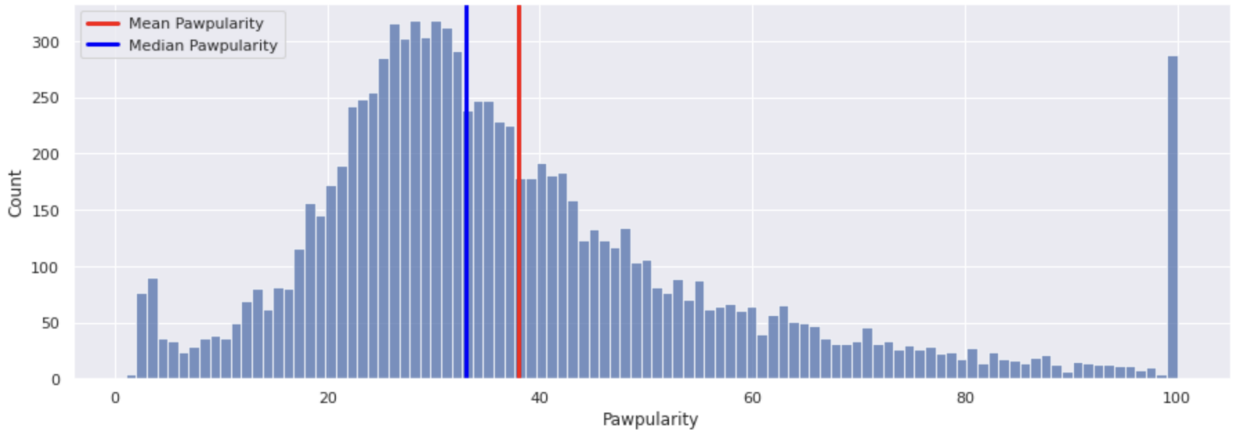
- **Focus** - Pet stands out against an uncluttered background, not too close / far.
- **Eyes** - Both eyes are facing front or near-front, with at least 1 eye / pupil decently clear.
- **Face** - Decently clear face, facing front or near-front.
- **Near** - Single pet taking up a significant portion of the photo (roughly over 50% of photo width or height).
- **Action** - Pet in the middle of an action (e.g., jumping).
- **Accessory** - Accompanying physical or digital accessory / prop (i.e. toy, digital sticker), excluding collar and leash.
- **Group** - More than 1 pet in the photo.
- **Collage** - Digitally-retouched photo (i.e. with digital photo frame, combination of multiple photos).
- **Human** - Human in the photo.
- **Occlusion** - Specific undesirable objects blocking part of the pet (i.e. human, cage or fence). Note that not all blocking objects are considered occlusion.
- **Info** - Custom-added text or labels (i.e. pet name, description).

- **Blur** - Noticeably out of focus or noisy, especially for the pet's eyes and face. For Blur entries, "Eyes" column is always set to 0.

These provide crucial insights into specific visual attributes and quality aspects that can aid in training a model to distinguish and categorize pet images based on their visual characteristics.

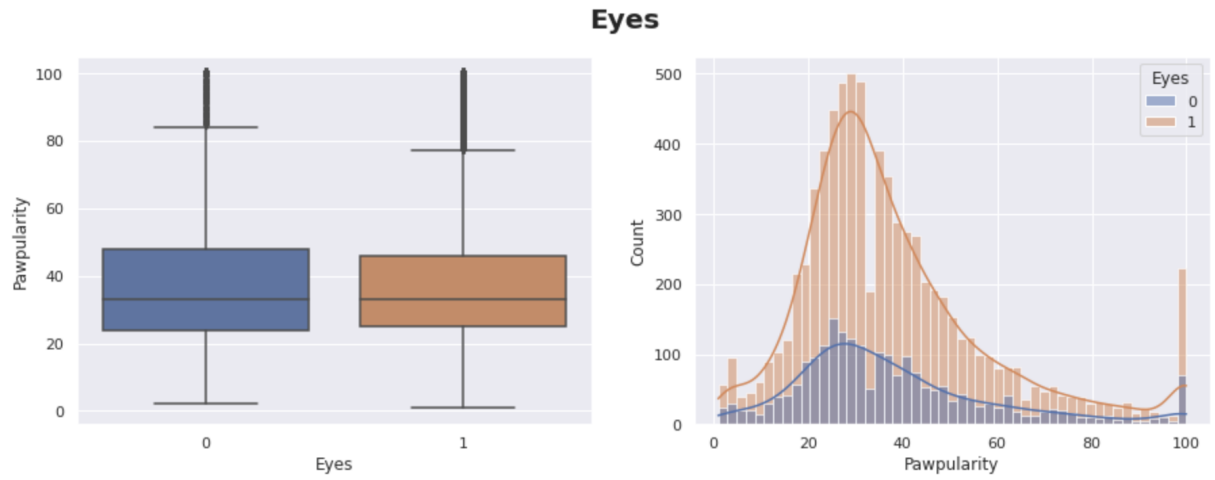
- Train.csv - This file details the metadata parameters associated with the training set of images. It includes information on Subject Focus, Eyes, Face, Blur, and various other attributes, providing a comprehensive source of contextual data for model training.
- Test.csv - Similar to Train.csv, this file contains metadata information specifically for the testing set of images. It plays a crucial role in evaluating the model's performance and generalizability on unseen data.

First, let us look at distribution of 'Pawpularity' scores across training data.

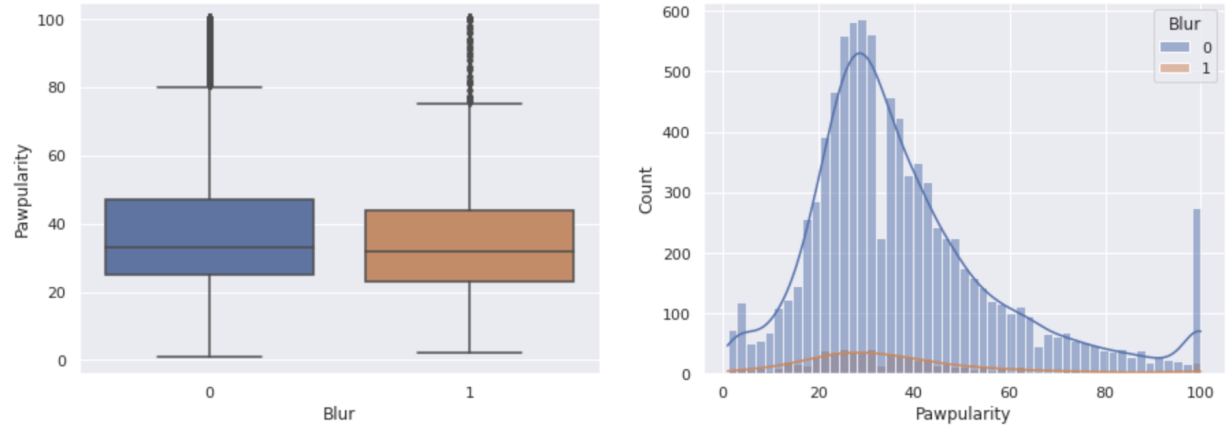


We notice a right skewed distribution, with a slight curve near zero score and close to 300 values of 100 score.

Comparing distribution of 'Eyes' with popularity score, we see that though there are more 1s than 0s in 'Eyes', the distribution of both values is similar with respect to the score which is evident from the nearly identical boxplots.



On the other hand, ‘Blur’ shows a different distribution for values 1 and 0.

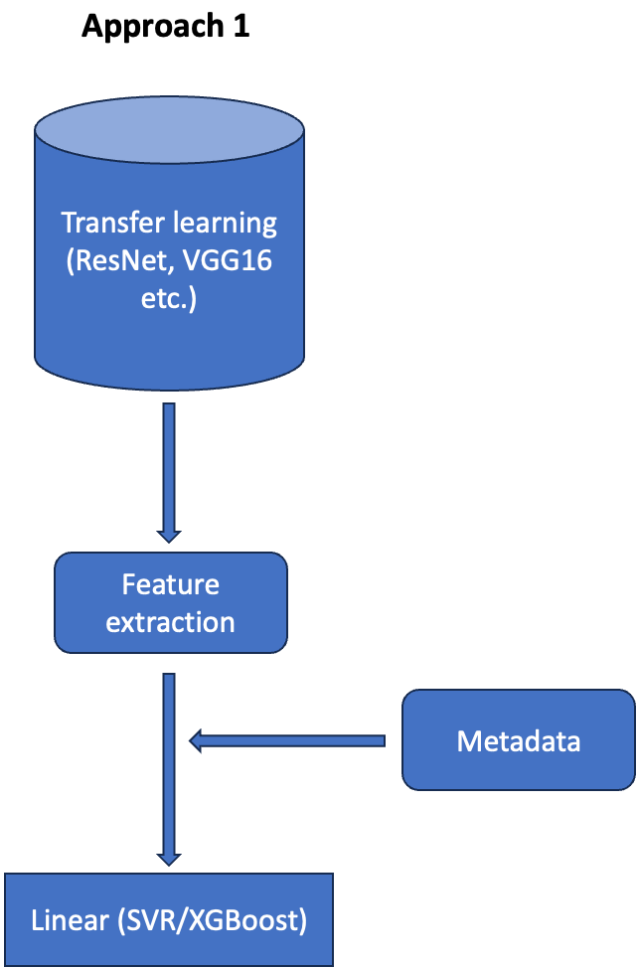


Although distribution charts cannot give us a complete picture of which features can solely contribute to the modeling, we can make a reasonable inference that given the difference in distributions of various metadata features, it is worthwhile to include them in the modeling to validate their contribution and importance to the problem statement at hand.

Image data - These contain training and testing image datasets - consisting of thousands of actual photos of pets for training the model and evaluating its performance. The images serve as the primary input for the machine learning model, enabling it to learn and recognize patterns associated with different visual attributes of pets.

5. Methodology

We opted for two approaches -



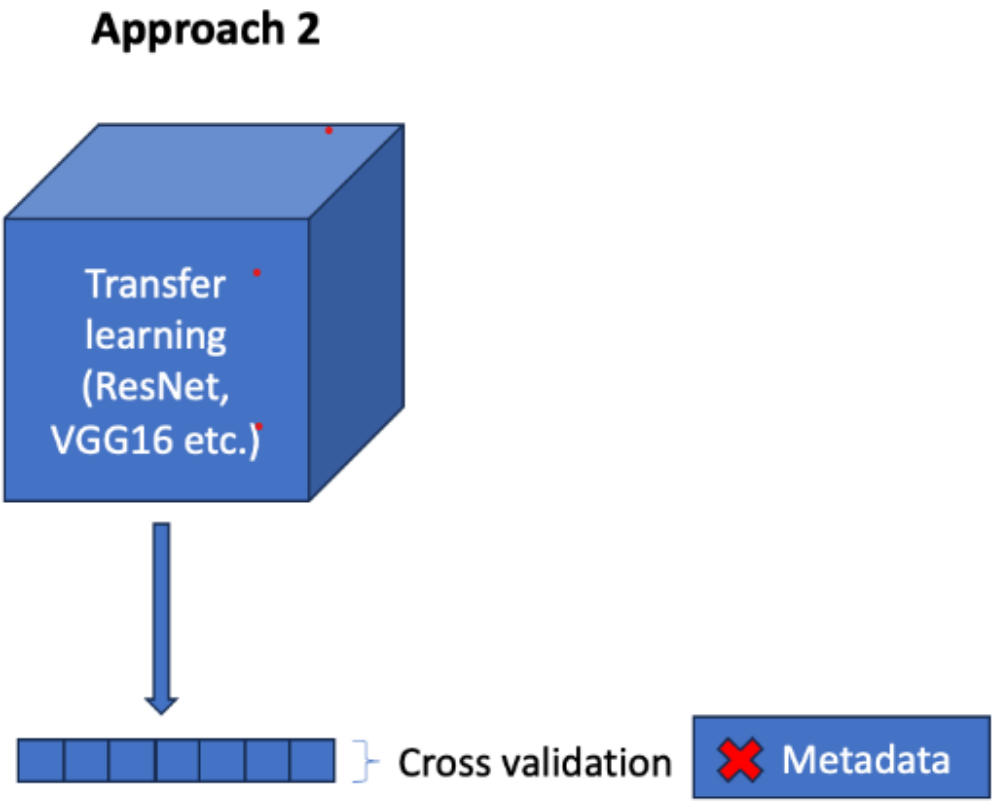
This approach involves leveraging transfer learning techniques such as ResNet, VGG16, InceptionV3, and others to extract meaningful features from a training dataset of images. Transfer learning is a common practice in computer vision tasks where pre-trained models, which have been trained on large datasets for tasks like image recognition, are utilized as a starting point for a new, related task.

The process begins by using these pre-trained models as feature extractors, where the learned representations from the images are captured in the form of feature vectors. These feature vectors encode hierarchical and abstract information about the visual content of the images. This step is crucial because pre-trained models have already learned to recognize general patterns and features in diverse images, saving computational resources and time.

Following the feature extraction, the metadata associated with each image, containing information about parameters such as Subject Focus, Eyes, Face, Blur, etc., is incorporated as

additional input. This results in a combined feature set, where both the image-derived features and metadata are considered as input variables.

To make a final prediction, a linear algorithm such as Support Vector Regression (SVR), XGBoost, or another regression model is employed. The purpose of this linear layer is to learn the relationship between the extracted features and the target variable, such as popularity score of the pet images. The goal is to find a mapping that minimizes the Root Mean Squared Error (RMSE), representing the difference between the predicted Pawpularity scores and the actual scores in the training dataset.



In the second approach, the focus is on training the image dataset directly using transfer learning techniques without using metadata. The process involves leveraging nested cross-validation to evaluate model performance and select the best-performing model.

One of the main goals of comparing these two approaches is to understand the importance of metadata and its contribution towards classification of images. This will help decide whether or not we should include metadata in the modeling moving forward.

6. Predictive Modeling

Over the course of this project duration, we have built five unique models by integrating either of the two aforementioned approaches.

6.1 Model 1: SVR on Metadata

As a precursor to approach 1, we initially wanted to gauge the predictive capability of metadata towards distinguishing images. Hence, we employed a linear Support Vector Regression (SVR) to capture and predict the variations in images based on specific visual attributes and quality aspects contained in the metadata.

6.1.1 Data preparation

Firstly, we load the two datasets train.csv and test.csv.

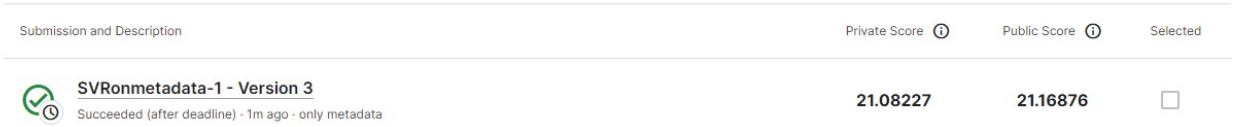
```
train = pd.read_csv('/kaggle/input/petfinder-pawpularity-score/train.csv')
test = pd.read_csv('/kaggle/input/petfinder-pawpularity-score/test.csv')
```

Now, we drop the 'ID' column and define dependent and independent variables - X and Y. We then fit the SVR model using these X and Y and predict on a test dataset to calculate the predictive performance of the model.

```
Xm = metadata_df.drop(columns=['Id', 'Pawpularity'])
Ym = metadata_df["Pawpularity"]
```

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
SVR = SVR()

#Fit the model on the training data
SVR.fit(Xm, Ym)

# Make predictions on the testing data
SVR_predictions = SVR.predict(x_test)

print(SVR_predictions)
```

Upon running, this model gave an RMSE of 21.08 which served as our benchmark to compare all future models against, as well as decide between the two approaches.



Submission and Description	Private Score	Public Score	Selected
SVRonmetadata-1 - Version 3 Succeeded (after deadline) · 1m ago · only metadata	21.08227	21.16876	☐

6.2 Model 2: CNN on images dataset

After making a move in Approach 1, now it is time to test the second approach.

CNN is relatively simple and straightforward in its approach and efficacy towards capturing hierarchical features and patterns in images when compared to the more complex pre-trained models like Resnet. It seemed prudent to gradually explore this realm with CNN before delving into the complexity of transfer learning models. Hence, we employ a CNN network on train and test image dataset as a starting point to extract meaningful features for image classification.

Firstly, we import the train and test images.

```
#get the data
train = pd.read_csv('/content/drive/My Drive/petfinder-pawpularity-score/train.csv')
test = pd.read_csv('/content/drive/My Drive/petfinder-pawpularity-score/test.csv')
```

Secondly, we define functions, train_id_to_path and test_id_to_path, to append the appropriate directory path and file extension to each identifier. These are used to transform image file identifiers (ID) into full image paths in a new column 'img_path'.

```
#Modify the Id such that each Id is the full image path. In the form
def train_id_to_path(x):
    return '/content/drive/My Drive/petfinder-pawpularity-score/train/' + x + ".jpg"
def test_id_to_path(x):
    return '/content/drive/My Drive/petfinder-pawpularity-score/test/' + x + ".jpg"

#Read in the data and drop unnecessary columns
train = pd.read_csv('/content/drive/My Drive/petfinder-pawpularity-score/train.csv')
train = train.drop(['Subject Focus', 'Eyes', 'Face', 'Near', 'Action', 'Accessory', 'Group', 'Collage', 'Human', 'Occlusion', 'Info', 'Blur'],axis=1)

test = pd.read_csv('/content/drive/My Drive/petfinder-pawpularity-score/test.csv')
test = test.drop(['Subject Focus', 'Eyes', 'Face', 'Near', 'Action', 'Accessory', 'Group', 'Collage', 'Human', 'Occlusion', 'Info', 'Blur'],axis=1)

#Add the .jpg extensions to the image file name ids
train["img_path"] = train["Id"].apply(train_id_to_path)
test["img_path"] = test["Id"].apply(test_id_to_path)
```

Since CNN is not inherently robust to outliers or non-linearities, we bin the dataset into 2, 4 and 10 bins based on quantile binning in order to capture non-linear relationships between the 'Pawpularity' values and the target variable more effectively. It provides a way to represent the data in a format that may better align with the underlying patterns.

```
#binning columns to test models
train['two_bin_pawp'] = pd.qcut(train['Pawpularity'], q=2, labels=False)
train = train.astype({"two_bin_pawp": str})

train['four_bin_pawp'] = pd.qcut(train['Pawpularity'], q=4, labels=False)
train = train.astype({"four_bin_pawp": str})

train['ten_bin_pawp'] = pd.qcut(train['Pawpularity'], q=10, labels=False)
train = train.astype({"ten_bin_pawp": str})
```

We convert images into eager tensors to prepare them for their computation using TensorFlow operations.

```
#Set the size image you want to use
image_height = 128
image_width = 128

#define a function that accepts an image url and outputs an eager tensor
def path_to_eagertensor(image_path):
    raw = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(raw, channels=3)
    image = tf.cast(image, tf.float32) / 255.0
    #image = tf.image.resize_with_pad(image, image_height, image_width) #optional with padding to retain original dimensions
    image = tf.image.resize(image, (image_height, image_width))
    return image
```

Now, we create a CNN model with the following specifications -

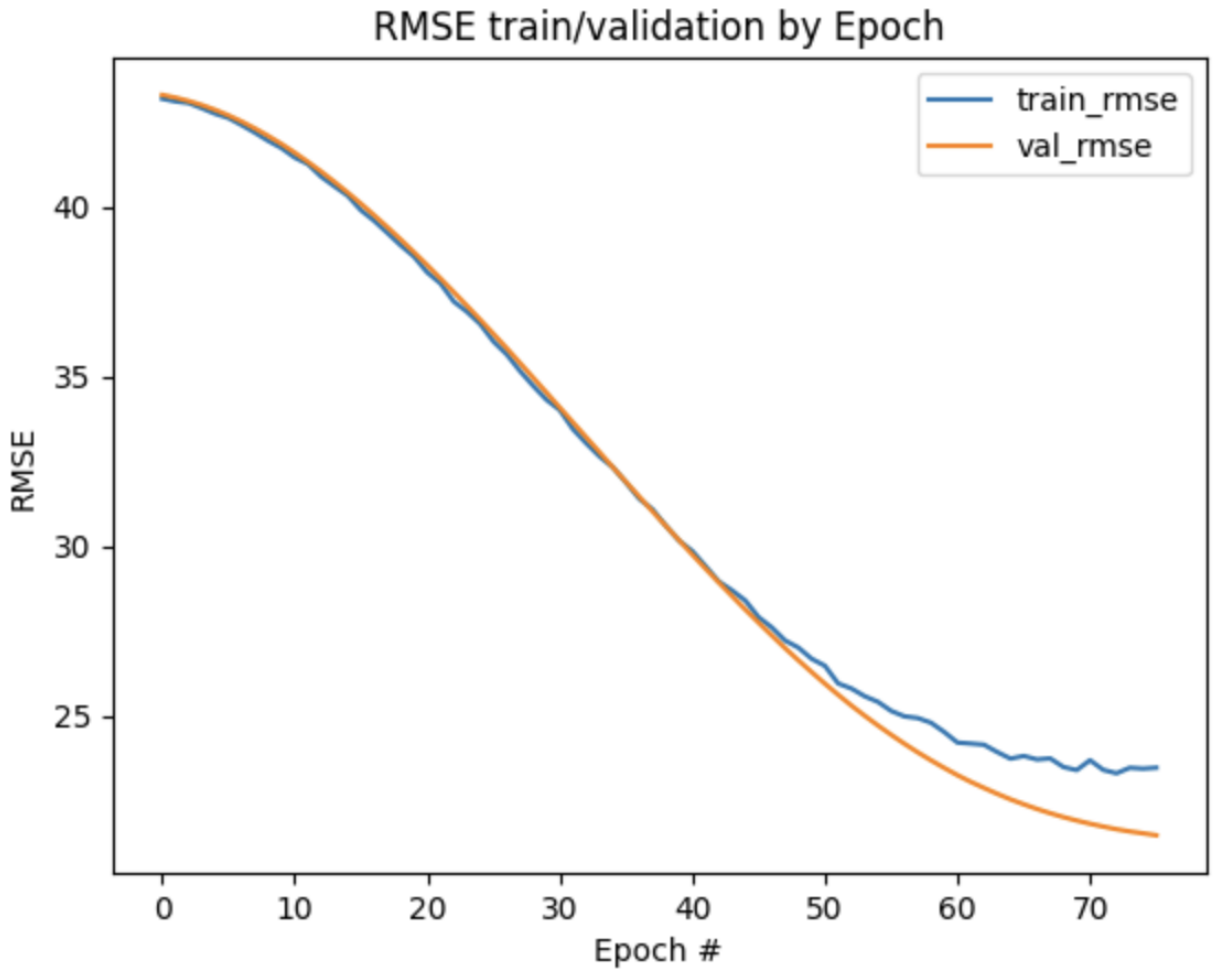
- 3 convolutional layers with 16, 32 and 64 filters and 2 MaxPooling layers
- 3 dense layers with 64, 1 and 1 neurons respectively
- ReLU is used for the activation function in convolutional and dense layers, while linear activation is used in the output layers
- Dropout layers with a dropout rate of 0.1 are included after the dense layers to prevent overfitting

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(16,(3,3),activation='relu',input_shape=(128,128,3)))
model.add(tf.keras.layers.MaxPool2D((2,2)))

model.add(tf.keras.layers.Conv2D(32,(3,3),activation='relu'))
model.add(tf.keras.layers.MaxPool2D((2,2)))

model.add(tf.keras.layers.Conv2D(64,(3,3),activation='relu'))
model.add(tf.keras.layers.MaxPool2D((2,2)))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64,activation='relu'))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.Dense(1,activation='linear'))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.Dense(1,activation='linear'))
```

We augment the images dataset using 'ImageDataGenerator' and implement 'EarlyStopping' before fitting the model with training data. This is done in order to mitigate overfitting and improve the generalizability of the model. 'ImageDataGenerator' generates augmented batches of images during training, applying specified transformations to each batch. This ensures that the model sees a diverse set of images in each epoch, enhancing its ability to learn robust and generalizable features.

```
data_augmentation =  ImageDataGenerator(
    rotation_range = 15,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    brightness_range=(0.1, 0.9),
    shear_range = 0.1,
    zoom_range = 0.15,
    horizontal_flip = True,
    fill_mode = "nearest")
# ImageDataGenerator(shear_range=0.2,
#                              zoom_range=0.2,
#                              horizontal_flip=True)
```

```
es_callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

history = model.fit(
    data_augmentation.flow(x_train,y_train,batch_size=128),
    validation_data = (x_test,y_test),
    steps_per_epoch = len(x_train) // 128,
    callbacks=[es_callback],
    epochs = 100
)
```



Upon running, CNN yielded an RMSE of 21.59 which is higher than the modeling with metadata. This comparative analysis helped us understand that metadata is in fact very important in understanding the patterns of image data, and hence needs to be incorporated into the modeling for superior results.

Submission and Description	Private Score	Public Score	Selected
CNN-final - Version 1 Succeeded (after deadline) · 6h ago · Customized CNN	21.58767	21.65214	☐

Therefore, we choose Approach 1 as the intended approach for our modeling further.

6.3 Model 3: Resnet (with Metadata)

We will load the metadata similar to previous mode, and transform the ID of the image into an image path using lambda functions.

```
train["file_path"] = train["Id"].apply(lambda identifier: "../input/petfinder-pawpularity-score/train/" + identifier + ".jpg")
test["file_path"] = test["Id"].apply(lambda identifier: "../input/petfinder-pawpularity-score/test/" + identifier + ".jpg")
```

In this first transfer learning model, we will load the twelve metadata attributes into a tabular form using a custom preprocess function that inputs the URL of the image. The preprocess function does image preprocessing including reading, decoding, normalization, cropping, and resizing. The tabular features are extracted from index 1 onwards, and the first element is treated as the label (cast to float32).

```
def preprocess(image_url, tabular):
    image_string = tf.io.read_file(image_url)
    image = tf.image.decode_jpeg(image_string, channels=3)
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.image.central_crop(image, 1.0)
    image = tf.image.resize(image, (image_size, image_size))
    return (image, tabular[1:]), tf.cast(tabular[0], tf.float32)
```

In the next step, we employ multimodal learning - create an Artificial Neural Network (ANN) for tabular data and function ‘block’ for creating a block of convolutional layers for the ResNet CNN.This is done primarily because data is of both tabular and image types. Tabular data typically contains features that are easy to represent in a structured form, such as numerical or categorical variables. Image data, on the other hand, requires specialized techniques for feature extraction due to its spatial nature.

Each modality (tabular and image) may capture different aspects of the Pawpularity score. By using a combination of both modalities, the model may be able to make more accurate predictions than if it were trained on either modality alone.

```
def build_tabular_model(inputs):

    x = keras.layers.Dense(12, activation='relu')(inputs)
    x = keras.layers.Dense(64, activation='relu')(x)
    x = keras.layers.Dropout(0.3)(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dense(64, activation='relu')(x)
    x = keras.layers.Concatenate()([x, inputs])
    return x
```

```
def block(x, filters, kernel_size, repetitions, pool_size=2, strides=2):
    for i in range(repetitions):
        x = tf.keras.layers.Conv2D(filters, kernel_size, activation='relu', padding='same')(x)
    x = tf.keras.layers.MaxPooling2D(pool_size, strides)(x)
    return x
```

Next, we define a function ‘get_model’ which is responsible for creating the overall architecture of the neural network model.

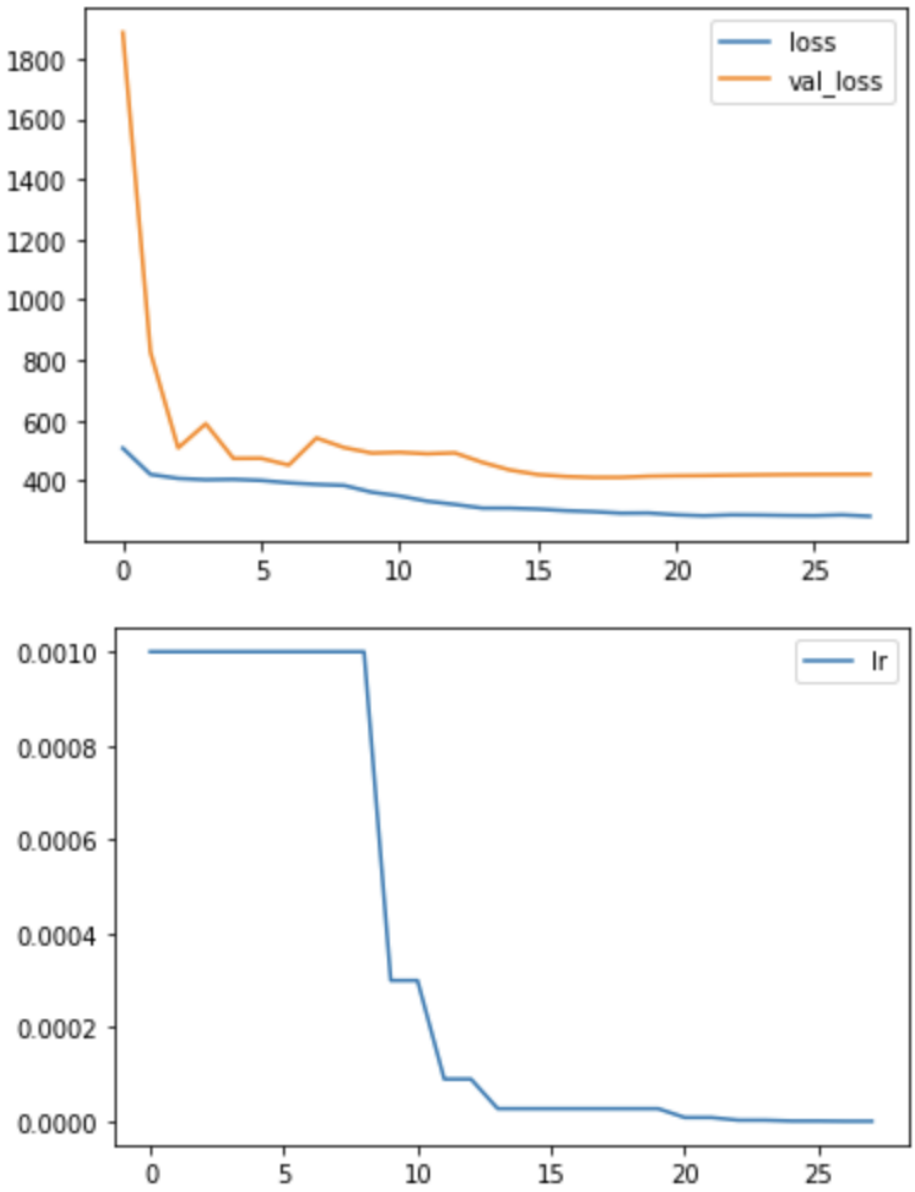
- Two input layers are defined: image_inputs for image data and tabular_inputs for tabular data.
- The model is loaded with pre-trained weights either from a local file (weights_path) or, if not found, using 'imagenet' weights.
- Image data is passed through data augmentation layers (RandomContrast and RandomRotation) before being processed by the ResNet50 model.
- The output from the ResNet50 model is then globally average-pooled.
- A dense layer with a single unit is added as the output layer.

The concatenate layer is used to combine the output from the ANN (tabular features) and CNN (image features). This allows the model to learn complex interactions between tabular and image features, enabling better representation learning.

```
import os
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import RandomContrast, RandomRotation

def get_model(image_size, columns):
    image_inputs = tf.keras.Input((image_size, image_size, 3))
    tabular_inputs = tf.keras.Input(len(columns))

    weights_path = '/kaggle/input/resnet50-weights-h5/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5'
    if os.path.exists(weights_path):
        resnet = ResNet50(include_top=False, weights=weights_path, pooling=None)
    else:
        print("Weights file not found. Using 'imagenet' weights.")
        resnet = ResNet50(include_top=False, weights='imagenet', pooling=None)

    image_x = resnet(RandomContrast(factor=0.1)(RandomRotation(factor=0.15)(image_inputs)))
    image_x = tf.keras.layers.GlobalAveragePooling2D()(image_x)

    # Assuming build_tabular_model is a function you have defined to handle tabular data
    tabular_x = build_tabular_model(tabular_inputs)

    x = tf.keras.layers.Concatenate(axis=1)([image_x, tabular_x])
    output = tf.keras.layers.Dense(1)(x)
    model = tf.keras.Model(inputs=[image_inputs, tabular_inputs], outputs=[output])
    return model
```

Using K-Fold cross-validation (K=5) , we define a training loop for training the model. Training hyperparameters include the number of epochs, early stopping, learning rate reduction, etc.
It trains the model for each fold, saves the best weights, and logs the training history. The training history is also visualized by plotting metrics such as loss, mean absolute error (MAE), mean squared error (RMSE), mean absolute percentage error (MAPE), and learning rate. The trained models and their corresponding histories are stored in the 'models' and 'historys' lists, respectively.

```
tf.keras.backend.clear_session()
models = []
historys = []
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
train_best_fold = True
best_fold = 0
for index, (train_indices, val_indices) in enumerate(kfold.split(train)):
    if train_best_fold and index != best_fold: continue
    x_train = train.loc[train_indices, "file_path"]
    tabular_train = train.loc[train_indices, ["Pawpularity"] + columns]
    x_val= train.loc[val_indices, "file_path"]
    tabular_val = train.loc[val_indices, ["Pawpularity"] + columns]
    checkpoint_path = "model_%d.h5"%(index)
    checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_path, save_best_only=True)
    early_stop = tf.keras.callbacks.EarlyStopping(min_delta=1e-4, patience=10)
    reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(factor=0.3,patience=2, min_lr=1e-7)
    callbacks = [early_stop, checkpoint, reduce_lr]
    optimizer = tf.keras.optimizers.Adam(1e-3)
    train_ds = tf.data.Dataset.from_tensor_slices((x_train, tabular_train)).map(preprocess).shuffle(512).batch(batch_size).cache().prefetch(2)
    val_ds = tf.data.Dataset.from_tensor_slices((x_val, tabular_val)).map(preprocess).batch(batch_size).cache().prefetch(2)
    model = get_model(image_size, columns)
    model.compile(loss = "mse", optimizer = optimizer, metrics = ["mae", rmse, "mape"])
    history = model.fit(train_ds, epochs=300, validation_data=val_ds, callbacks=callbacks, batch_size = 8)
    for metrics in [("loss", "val_loss"), ("mae", "val_mae"), ("mape", "val_mape"), ["lr"]]:
        pd.DataFrame(history.history, columns=metrics).plot()
        plt.show()
    model.load_weights(checkpoint_path)
    historys.append(history)
    models.append(model)
```



In the final steps, we define a function to preprocess the test data that is similar to the one for training, and use it to make predictions from the trained model.

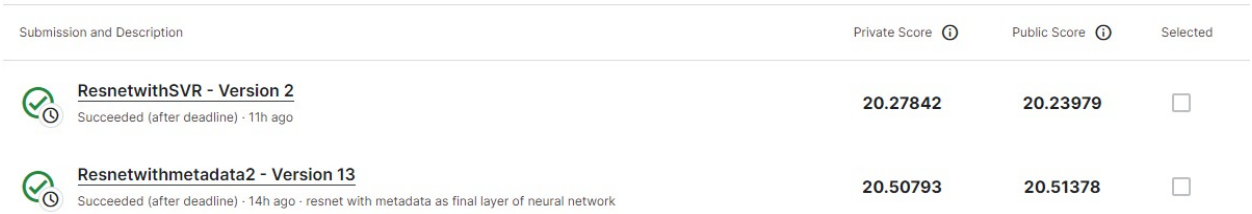
```
use_best_result = False
if use_best_result:
    if train_best_fold:
        best_model = models[0]
    else:
        best_fold = 0
        best_score = 10e8
        for fold, history in enumerate(historys):
            for val_rmse in history.history["val_rmse"]:
                if val_rmse < best_score:
                    best_score = val_rmse
                    best_fold = fold
        print("Best Score:%.2f Best Fold: %d"%(best_score, best_fold + 1))
        best_model = models[best_fold]
    results = best_model.predict(test_ds).reshape(-1)
else:
    total_results = []
    for model in models:
        total_results.append(model.predict(test_ds).reshape(-1))
    results = np.mean(total_results, axis=0).reshape(-1)
sample_submission["Pawpularity"] = results
sample_submission.to_csv("submission.csv", index=False)
```

Upon execution, the ResNet model demonstrated a notable improvement, yielding a Root Mean Squared Error (RMSE) of 20.51. This signifies a positive advancement compared to the performance of the previous two models. The outcomes from the ResNet model have not only showcased enhanced predictive accuracy but have also steered our strategy toward exploring additional transfer learning techniques. This success underscores the effectiveness of leveraging pre-trained models like ResNet for the given task, motivating further exploration of advanced transfer learning approaches in future model iterations.

6.4 Model 4: Improved ResNet (with Metadata)

In this model iteration, our objective was to explore the fine-tuning of different hyperparameters to enhance performance. Through systematic experimentation with various permutations and combinations, we identified that adjusting two key hyperparameters resulted in the most significant improvements.

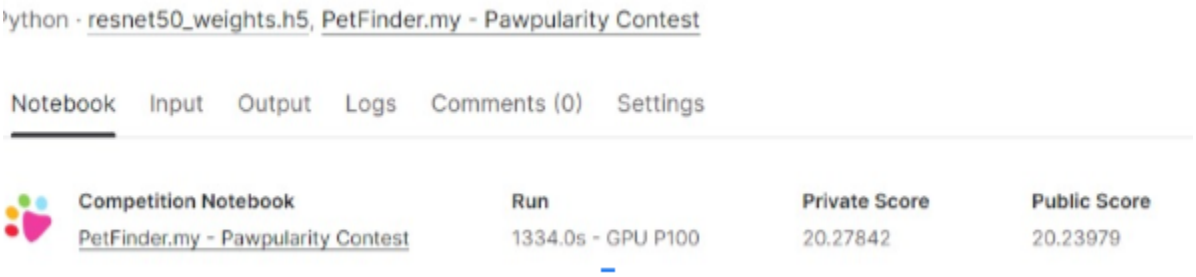
Specifically, increasing the number of neurons from 64 to 128 and elevating the dropout rate from 0.1 to 0.3 led to an impressive reduction in the Root Mean Squared Error (RMSE) to 20.28.



6.5 Model 5: EfficientNet (with Metadata)

In the next model iteration, we experimented with a highly effective image classification algorithm - EfficientNet. EfficientNet is designed to be highly efficient in terms of computational resources. It introduces a compound scaling method that uniformly scales the network's depth, width, and resolution. This approach balances model size and accuracy. It achieves state-of-the-art performance on various image classification tasks, and outranks ResNet in terms of accuracy.

Upon execution, EfficientNet yielded an RMSE of 20.23.



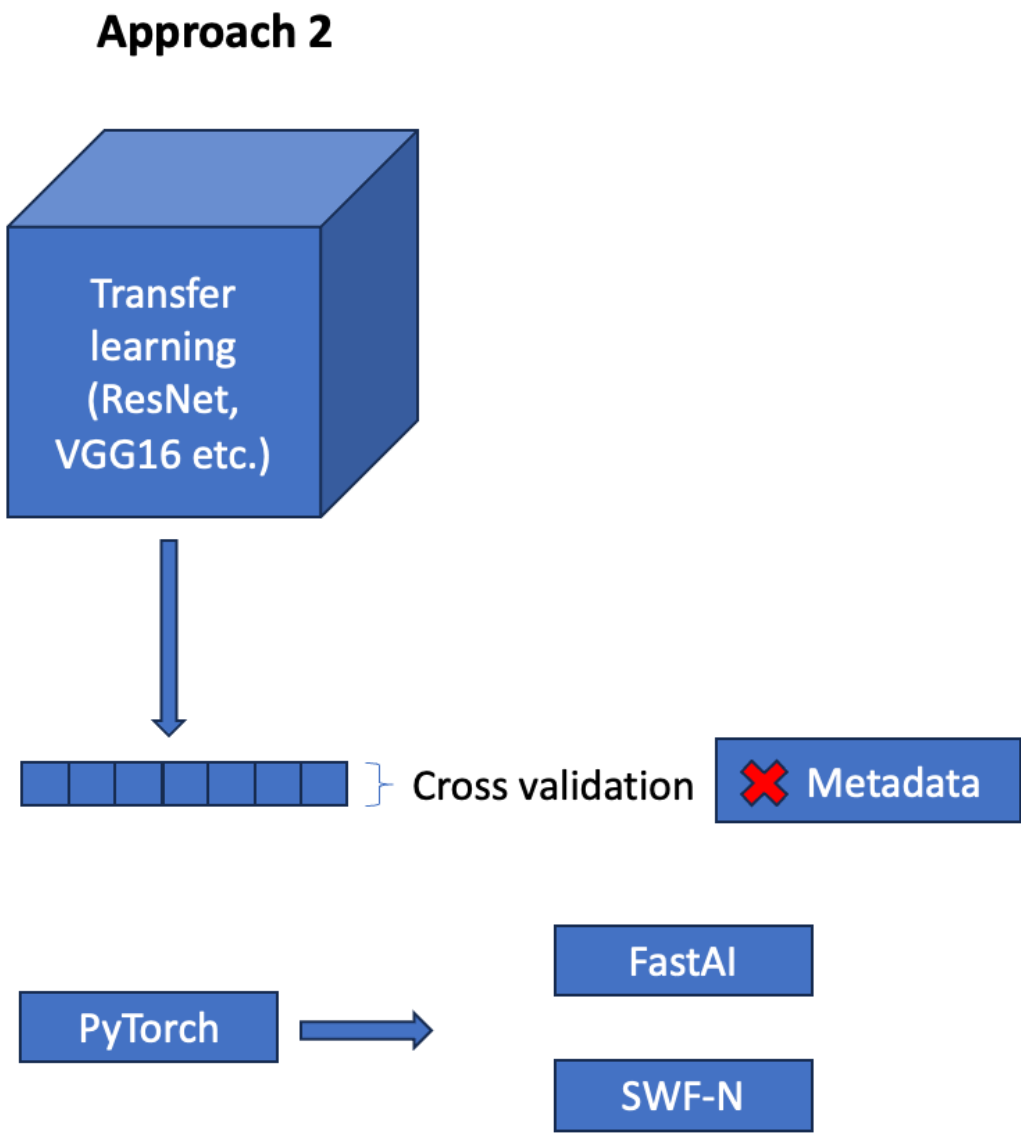
6.6 Model 6: ResNet + EfficientNet + SVR

Until now, our exploration has involved using SVR, CNN, ResNet, and EfficientNet as standalone image classification algorithms. Through this process, we've come to recognize that relying solely on a single image classification algorithm may not yield the desired operational prediction efficiency. The next logical progression in this journey involves combining the strengths of two transfer learning techniques to extract features, and augment this result metadata as input for a final prediction model.

In the final model, we use PyTorch and deep learning such as timm (PyTorch Image Models) and CLIP (Contrastive Language-Image Pre-training) to create an advanced deep learning framework.

- PyTorch is used as the primary deep learning framework. It provides a flexible and dynamic computational graph that offers a variety of pre-built layers and modules which simplify the process of building complex architectures.
- The timm library is utilized for image feature extraction. Transfer learning is employed, where pre-trained models (e.g., tf_efficientnet_l2_ns_475, ViT-B-16) are loaded. These models have been pre-trained on large datasets for general image understanding and can be fine-tuned for specific tasks.
- CLIP is used for advanced image understanding. It combines vision and language pre-training to enable models to understand images in a broader context, including relationships with textual descriptions.

To reduce RMSE further, we revised our approach to include advanced model versions from FastAI.



6.6.1 Data Preprocessing

We create a custom class 'PetfinderDataSet' to load and process images from the PetFinder dataset.

Constructor __init__ : Apply specified transformations on the input images

Constructor __len__ : Returns the size of total number of samples in the dataset

Constructor __getitem__ method is responsible for loading and processing an individual sample given its index (idx).

```
class PetfinderDataSet(Dataset):
    def __init__(self, img_id_lst, transform, base_path='./petfinder-pawpularity-score/test/', new_size = 0):
        self.img_id_lst = img_id_lst.copy()
        self.base_path = base_path
        self.new_size = new_size
        self.transform = transform

    def __len__(self):
        return len(self.img_id_lst)

    def __getitem__(self, idx):
        img = Image.open(self.base_path + self.img_id_lst[idx] + '.jpg').convert('RGB')

        resized_img = self.transform(img)

        return resized_img
```

The 'PetCropDataset_CenterCrop' dataset is designed for a pet image classification task. It provides functionality for cropping and resizing pet images based on center coordinates. It accommodates cases where the detection of a cat or dog fails. The dataset prepares images for training deep learning models by applying relevant transformations, facilitating improved model generalization and performance.

```
# To be modified
class PetCropDataset_CenterCrop(Dataset):
    def __init__(self, df, end_trans, base_path='./petfinder-pawpularity-score/test/', new_size = 0):
        self.img_id_lst = df["Id"].tolist().copy()
        self.df = df
        self.base_path = base_path
        self.new_size = new_size
        self.end_trans = end_trans

    def __len__(self):
        return len(self.img_id_lst)

    def __getitem__(self, idx):

        img = Image.open(self.base_path + self.img_id_lst[idx] + '.jpg').convert('RGB')
        width, height = img.size

        center_chosen = [self.df.iloc[idx, -4], self.df.iloc[idx, -3]] # the pet's center ratio coordinate

        if self.df.iloc[idx, -2] == 0: # failed to detect cat or dog
            cropped_img = T.CenterCrop(size=min(width, height))(img)
        else:
            if width > height:
                (top_left, top_right, bottom_left, bottom_right, center) = T.FiveCrop(size=(height, height))(img)
                margin = (height/2)/width
                if center_chosen[0] < margin: # crop to the left
                    cropped_img = top_left
                elif center_chosen[0] > (1-margin): # crop to the right
                    cropped_img = top_right
                else:
                    left = int( width * (center_chosen[0]-margin) )
                    cropped_img = T.functional.crop(img=img, top=0, left=left, height=height, width=height)
            else:
                (top_left, top_right, bottom_left, bottom_right, center) = T.FiveCrop(size=(width, width))(img)
                margin = (width/2)/height
                if center_chosen[1] < margin: # crop to the top
                    cropped_img = top_left
                elif center_chosen[1] > (1-margin): # crop to the bottom
                    cropped_img = bottom_left
                else:
                    top = int( height * (center_chosen[0]-margin) )
                    cropped_img = T.functional.crop(img=img, top=0, left=0, height=width, width=width)

        resized_img = T.Resize(size=self.new_size)(cropped_img)
        resized_img = self.end_trans(resized_img)

        return resized_img
```

The CLIPDataset class represents a dataset for the CLIP (Contrastive Language-Image Pre-training) model. It takes a list of image IDs, a base path for image files, and a preprocessing function as input. The __len__ method returns the dataset's length, and __getitem__ loads and preprocesses an image based on the provided image ID, returning the processed image for training or evaluation.

```
class CLIPDataset(Dataset):
    def __init__(self, img_id_lst, base_path='../input/petfinder-pawpularity-score/test/', preprocess=None):

        self.img_id_lst = img_id_lst.copy()
        self.base_path = base_path
        self.preprocess = preprocess

    def __len__(self):
        return len(self.img_id_lst)

    def __getitem__(self, idx):
        img = Image.open(self.base_path + self.img_id_lst[idx] + '.jpg').convert('RGB')
        img = self.preprocess(img)
        return img
```

Next, we perform feature extraction using both Timm and CLIP models. The following are the pre-trained models for each of the above two, serialized using pickle -

Timm - [tf_efficientnet_l2_ns_475, beitv2_large_patch16_224]

CLIPP - [RN50x16", "RN50x4", "ViT-B-16", "ViT-B-32]

Feature extraction using Timm models

```
# timm_model_lst = ["../input/timm-pretrained-models/beitv2_large_patch16_224.pkl"]
timm_model_lst = ["../input/timm-pretrained-models/tf_efficientnet_l2_ns_475.pkl",
                  "../input/timm-pretrained-models/beitv2_large_patch16_224.pkl"]
#"../input/timm-pretrained-models/swin_base_patch4_window7_224.pkl"
                  # "../input/timm-pretrained-models/vit_large_r50_s32_384.pkl",
                  # "../input/timm-pretrained-models/vit_large_patch16_384.pkl"]
for m in timm_model_lst:
    print("Extracing features using", m)
    pretrained_model = pickle.load( open( m, "rb" ) ).to('cuda')
    trans_config = resolve_data_config({}, model=pretrained_model)
    default_trans = create_transform(**trans_config)

    test_dataset = PetfinderDataSet(img_id_lst=test_id_lst, transform=default_trans, base_path=test_data_base_path)
    test_data_loader = DataLoader(test_dataset, batch_size=batch_size, num_workers=2, shuffle=False)

    with torch.no_grad():
        emb = [pretrained_model(data.to('cuda')).cpu().numpy() for data in test_data_loader]
    EMB_full = np.concatenate(emb, 0)

    model_name = m.split("/")[-1].split(".")[0]
    pickle.dump( EMB_full, open( "./testset_features/"+model_name+"_features.pkl", "wb" ) )

    del pretrained_model, EMB_full
    torch.cuda.empty_cache()
    gc.collect()
    print("Done!\n")
```

```
Extracing features using ../input/timm-pretrained-models/tf_efficientnet_l2_ns_475.pkl
Done!

Extracing features using ../input/timm-pretrained-models/beitv2_large_patch16_224.pkl
Done!
```

Feature extraction using CLIP models

```
# CLIP_model_lst = ["../input/clip-full-package/CLIP_full_package/CLIP_models/ViT-B-16.pkl"]
CLIP_model_lst = ["../input/clip-full-package/CLIP_full_package/CLIP_models/RN50x16.pkl",
                  "../input/clip-full-package/CLIP_full_package/CLIP_models/RN50x4.pkl",
                  "../input/clip-full-package/CLIP_full_package/CLIP_models/ViT-B-16.pkl",
                  "../input/clip-full-package/CLIP_full_package/CLIP_models/ViT-B-32.pkl"]
for m in CLIP_model_lst:
    print("Extracing features using", m)
    clip_model, preprocess = pickle.load( open( m, "rb" ) )
    clip_model.to('cuda')

    full_dataset = CLIPDataset(img_id_lst=test_id_lst, base_path=test_data_base_path, preprocess=preprocess)
    full_data_loader = DataLoader(full_dataset, batch_size=batch_size, num_workers=2, shuffle=False)

    with torch.no_grad():
        emb = [clip_model.encode_image(data.to('cuda')).cpu().numpy() for data in full_data_loader]
    EMB_full = np.concatenate(emb, 0)

    model_name = m.split("/")[-1].split(".")[0]
    pickle.dump( EMB_full, open( "./testset_features/"+model_name+"_features.pkl", "wb" ) )

    del clip_model, EMB_full
    torch.cuda.empty_cache()
    gc.collect()
    print("Done!\n")
```

```
Extracing features using ../input/clip-full-package/CLIP_full_package/CLIP_models/RN50x16.pkl
Done!

Extracing features using ../input/clip-full-package/CLIP_full_package/CLIP_models/RN50x4.pkl
Done!

Extracing features using ../input/clip-full-package/CLIP_full_package/CLIP_models/ViT-B-16.pkl
Done!

Extracing features using ../input/clip-full-package/CLIP_full_package/CLIP_models/ViT-B-32.pkl
Done!
```

In this crucial step, the features obtained from Timm and CLIP are combined for the test set.

6.6.2 Overview of models -

- models: A baseline configuration with five different models.
- model_old_hill_1 (and) model_old_hill_2: Two sets of models with a scaling factor applied to one of the models.
- models_man_2 (and) models_man_5: Configurations with varying model combinations and a scaling factor applied.

6.6.3 Feature Aggregation Loop -

- The code initializes EMB_test to None. Then, it iterates through the models specified in the selected configuration (models), loading the corresponding features using pickle.
- For the first model (i == 0), the features are directly assigned to EMB_test. For subsequent models, features are concatenated along the second axis (axis=1) to combine them.
- After the loop, EMB_test contains the combined features from all specified models for the test set.

The final result is a combined feature set (EMB_test) that incorporates features from multiple pre-trained models. This aggregated feature set along with metadata is used downstream for linear prediction.

Combine features

```
# Fixed setting
models = ["RN50x16",
          "RN50x4",
          "ViT-B-16",
          "ViT-B-32",
          "tf_efficientnet_l2_ns_475"]
model_old_hill_1 = ["RN50x16",
                    "beitv2_large_patch16_224",
                    "beitv2_large_patch16_224"] # * 1.032
model_old_hill_2 = ["ViT-B-16",
                    "beitv2_large_patch16_224"] # * 1.032
models_man_2 = ["RN50x16",
                "RN50x4",
                "ViT-B-16",
                "ViT-B-32",
                "tf_efficientnet_l2_ns_475",
                "beitv2_large_patch16_224"] # loss: 17.296  * 1.030
models_man_5 = ["RN50x16",
          "RN50x4",
          "ViT-B-16",
          "ViT-B-32",
          "beitv2_large_patch16_224",
          "swin_base_patch4_window7_224",
          "tf_efficientnet_l2_ns"] # loss: 17.257  * 1.029
# "beitv2_large_patch16_224"
#           "vit_large_patch16_384",
#           "vit_large_r50_s32_384"
# models = ["beitv2_large_patch16_224",
#           "RN50x16"]
models = models_man_2
EMB_test = None
for i in range(len(models)):
    if i == 0:
        EMB_test = pickle.load( open( "./testset_features/"+models[0]+"_features.pkl", "rb" ) )
    else:
        EMB_test = np.concatenate((EMB_test, pickle.load( open( "./testset_features/"+models[i]+"_features.pkl", "rb" ) )), axis=1)
```

In the final step, we load a SVR and a scaler, and apply the loaded scaler to transform the combined features (EMB_test). We then use this SVR model to make predictions on the test dataset. An important note here is that we applied a scaling factor of 1.03 to the raw predictions and clipped the predictions to be within the range of 1 to 70 in order to fine-tune and constrain the model's outputs for better performance.

```
# load svr model
svr_model, scaler = pickle.load(open("../input/cumlsvr-full/cumlSVR_man_2.pkl", 'rb'))
EMB_test = scaler.transform(EMB_test)
# get prediction
test_pred = np.clip(svr_model.predict(EMB_test)*1.03, 1, 70)
# test_pred = svr_model.predict(EMB_test)*1.03
```

Finally, we create a submission file and submit it for evaluation.

```
df_test[['Id','Pawpularity']].to_csv('submission.csv', index=False)
```

Upon execution, this model yielded a highly impressive RMSE of 17.09.

****Kindly ignore the notebook names, as we were fine tuning the same notebook and submitting****

7. Learnings

Post completion of this project, we list the following findings as our key learnings:

Positive Contribution of Metadata: The inclusion of metadata features has proven to be valuable and positively impactful in the image classification task, when used with a stand-alone transfer learning model.

CNN Limitations: While Convolutional Neural Networks (CNNs) are commonly used for image classification, their limitations become apparent in this task. CNN is more suited for smaller image classification tasks with sufficient amounts of labeled data, and may not be suited for deep architectures where the complexity of the prediction is more.

Transfer Learning Superiority: Transfer Learning (TL) emerges as a superior approach - leveraging models pre-trained on extensive datasets is a sure-shot way to improve the performance of the prediction model and reduce the training and computation time required.

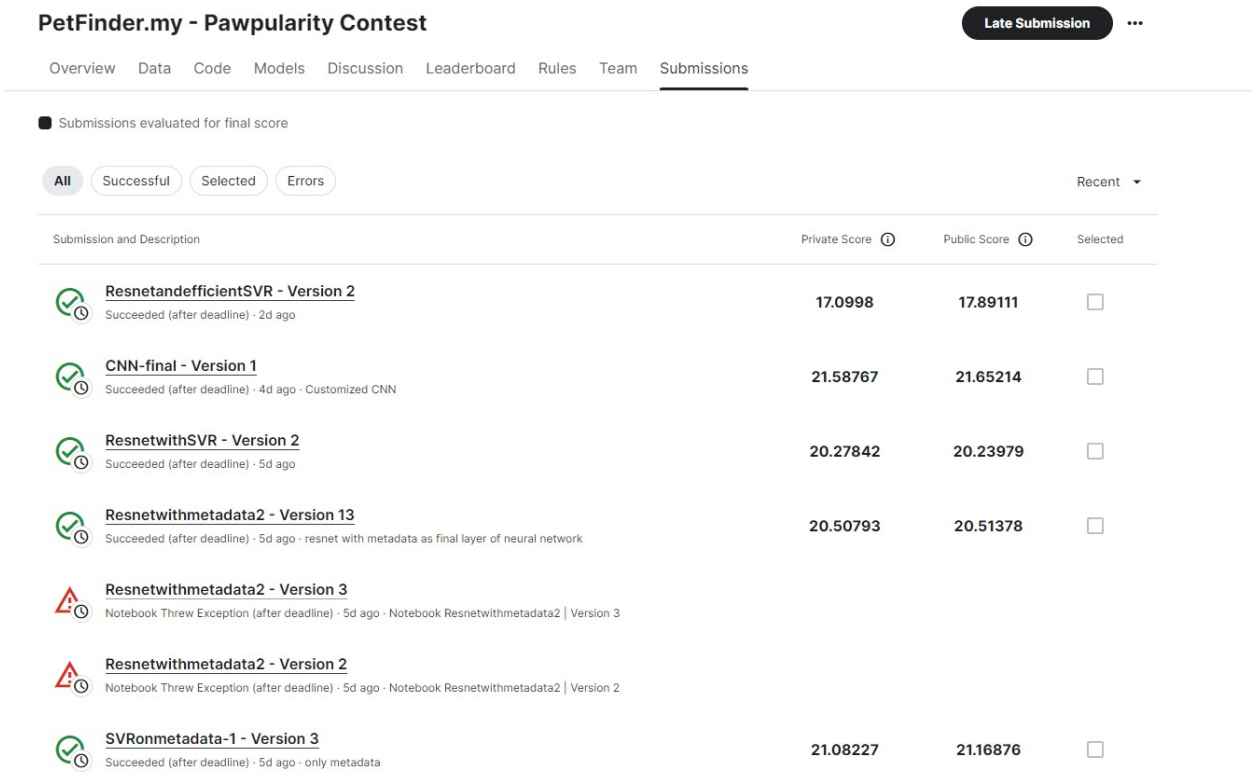
Multiple Transfer Learning Models: A single Transfer Learning model is generally insufficient to deliver best results in cases of complex image classification tasks. For optimal performance, it is recommended to use a combination of at least two Transfer Learning models in order to achieve desired performance.

Effectiveness of Hyperparameter Tuning: Fine-tuning hyperparameters, such as increasing the number of neurons and adjusting the dropout layer rate, proves more effective than unfreezing layers and then fine tuning a pre-trained transfer learning model. This emphasizes the critical role of feature extraction quality in image classification.

Significance of Data Augmentation: Utilizing Data Augmentation, specifically through ImageDataGenerator, emerges as a vital prerequisite. This technique significantly contributes to

improved segmentation of both the training and testing image datasets, further enhancing model robustness.

8. Final Kaggle Leaderboard



9. APPENDIX

1.**ResNets: Why Do They Perform Better than Classic ConvNets? (Conceptual Analysis)**

https://towardsdatascience.com/resnets-why-do-they-perform-better-than-classic-convnets-conceptual-analysis-6a9c82e06e53

2.**EfficientNet and its Performance Comparison with Other Transfer Learning Networks:**https://wisdomml.in/efficientnet-and-its-performance-comparison-with-other-transfer-learning-networks/

3.**Improved image super-resolution by Support Vector Regression**
https://ieeexplore.ieee.org/document/6033289

4. **Data augmentation for improving deep learning in image classification problem**
https://ieeexplore.ieee.org/abstract/document/8388338

5. **Top 4 pre-trained models For Image Classification:**
https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/

6. **Pytorch Image models**: https://timm.fast.ai/

7. **EfficientNet and its Performance Comparison with Other Transfer Learning Networks**

https://wisdomml.in/efficientnet-and-its-performance-comparison-with-other-transfer-learning-networks/