Generate training and testing sets in Google Colab

(I prepare the training and testing sets through Google Colab since I want to save the GPU in kaggle for model training.)

In [ ]:
```
# Easiest way to download kaggle data in Google Colab: https://www.kaggle.com/discussions/general/74235

# 1. Go to your account, Scroll to API section and Click Expire API Token to remove previous tokens
# 2. Click on Create New API Token - It will download kaggle.json file on your machine
# 3. Go to your Google Colab project file and run the following commands
```

In [1]:
```
# ! pip install -q kaggle
from google.colab import files
files.upload() # need to choose the file you've downloaded from
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[1]:
```
{'kaggle.json': b'{"username":"weichunchang2000","key":"773179abc6899133f0e9962470ce127f"}'}
```

In [2]:
```
# make directory named kaggle and copy kaggle.json file there
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/

# change the permissions of the file
! chmod 600 ~/.kaggle/kaggle.json

# list the dataset
! kaggle datasets list
```

```
ref                                               title                                                 size  lastUpdated
downloadCount  voteCount  usabilityRating
------------------------------------------------  ----------------------------------------------------  -----  ------------
-------  -------------  ---------  ---------------
thedrcat/daigt-v2-train-dataset                   DAIGT V2 Train Dataset                                 29MB  2023-11-16 0
1:38:36           1220         134  1.0
muhammadbinimran/housing-price-prediction-data    Housing Price Prediction Data                         763KB  2023-11-21 1
7:56:32           4736          89  1.0
```

```
carlmcbrideellis/llm-7-prompt-training-dataset  LLM: 7 prompt training dataset                    41MB  2023-11-15 0
7:32:56            1504         115  1.0
thedrcat/daigt-proper-train-dataset             DAIGT Proper Train Dataset                       119MB  2023-11-05 1
4:03:25            1459         134  1.0
joebeachcapital/30000-spotify-songs             30000 Spotify Songs                                3MB  2023-11-01 0
6:06:43            9888         211  1.0
jacksondivakarr/laptop-price-prediction-dataset  Laptop Price Prediction Dataset                 119KB  2023-11-30 1
6:23:34            813           29  1.0
ddosad/auto-sales-data                          Automobile Sales data                             79KB  2023-11-18 1
2:36:41            3860          69  1.0
julnazz/diabetes-health-indicators-dataset      Diabetes Health Indicators Dataset                 5MB  2023-11-27 0
7:10:53            853           21  1.0
nelgiriyewithana/world-educational-data         World Educational Data                             9KB  2023-11-04 0
6:10:17            7852         163  1.0
thedevastator/bank-term-deposit-predictions     Bank Term Deposit Predictions                    541KB  2023-11-30 1
4:37:39            849           29  1.0
sujaykapadnis/products-datasets                 Detailed Products Datasets                       100KB  2023-11-24 0
3:25:10            1070          26  1.0
maso0dahmed/video-games-data                    Video Games Data                                   5MB  2023-11-25 1
9:08:46            1214          36  1.0
alejopaullier/daigt-external-dataset            DAIGT | External Dataset                           3MB  2023-10-31 1
9:11:35            1004         122  0.7647059
nelgiriyewithana/australian-vehicle-prices      Australian Vehicle Prices                        582KB  2023-11-27 0
4:51:30            1126          44  1.0
prasad22/healthcare-dataset                     🩺Healthcare Dataset 🧪                               483KB  2023-10-31
11:30:58            7027         109  1.0
adampq/linkedin-jobs-machine-learning-data-set  LinkedIn Job Postings - Machine Learning Data Set   38MB  2023-11-28 1
7:18:04            437           25  1.0
jacksondivakarr/online-shopping-dataset         🛒 Online Shopping Dataset 📊📉📈                        5MB  2023-11-
12 12:35:58            4083          76  1.0
asimislam/30-yrs-stock-market-data              30 yrs Stock Market Data                         882KB  2023-11-29 2
0:18:02            1081          27  1.0
imtkaggleteam/life-expectancy                   Life Expectancy                                  730KB  2023-11-30 1
2:22:23            621           33  0.9411765
muhammadbinimran/covid-19-pandemic-data         COVID-19 Pandemic Data                            457B  2023-11-07 2
0:42:55            1012          24  0.9411765
```

In [3]:

```
# ! kaggle competitions download -c 'name-of-competition', you will find this in each competition
! kaggle competitions download -c petfinder-pawpularity-score
```

```
Downloading petfinder-pawpularity-score.zip to /content
100% 983M/983M [00:51<00:00, 25.9MB/s]
100% 983M/983M [00:51<00:00, 20.1MB/s]
```

In [ ]:
```
! rm -r pawpularitydataset # remove the directory if needed to rerun

! mkdir pawpularitydataset
! unzip petfinder-pawpularity-score.zip -d pawpularitydataset
```

In [ ]:
```
! ls pawpularitydataset
```

```
sample_submission.csv  test  test.csv  train  train.csv
```

In [ ]:
```
! ls pawpularitydataset/train | head -n 10
```

```
0007de18844b0dbbb5e1f607da0606e0.jpg
0009c66b9439883ba2750fb825e1d7db.jpg
0013fd999caf9a3efe1352ca1b0d937e.jpg
0018df346ac9c1d8413cfcc888ca8246.jpg
001dc955e10590d3ca4673f034feeef2.jpg
001dd4f6fafb890610b1635f967ea081.jpg
0023b8a3abc93c712edd6120867deb53.jpg
0031d6a9ef7340f898c3e05f92c7bb04.jpg
0042bc5bada6d1cf8951f8f9f0d399fa.jpg
0049cb81313c94fa007286e9039af910.jpg
```

In [ ]:
```
import numpy as np
import pandas as pd
import cv2
import os
import matplotlib.pyplot as plt
%matplotlib inline
import gc # garbage collector for cleaning deleted data from memory

pd.options.display.max_columns = None
pd.options.display.max_rows = None
```

We don't deal with the test set since we will submit the notebook for grading and will load the test set then

In [ ]:
```
train_imgs = [] # just to initialize in case I need to rerun
```

```
train_dir = 'pawpularitydataset/train'
train_imgs = ['pawpularitydataset/train/{}'.format(i) for i in os.listdir(train_dir)] # get train images
```

In [ ]:

```
train_imgs[:10]
```

Out[ ]:

```
['pawpularitydataset/train/15c681c62392f2ee73ee0087f37ddeaf.jpg',
 'pawpularitydataset/train/4130c0acf816e5b857a7217805da7f13.jpg',
 'pawpularitydataset/train/db26ad9754421faec035456f15269f52.jpg',
 'pawpularitydataset/train/f5f53baf396fee9ee0d51cf0ca5701cf.jpg',
 'pawpularitydataset/train/fc00c2d6b03a78ddd12cde5716c5b0ab.jpg',
 'pawpularitydataset/train/dc978e94fb761b9ee01b0595a2e3b9c8.jpg',
 'pawpularitydataset/train/1c8284661c5c710cd1bd517d5c3e0f63.jpg',
 'pawpularitydataset/train/d3df7802063d5cd7df72a873824015b2.jpg',
 'pawpularitydataset/train/2da1d3fb0dff907c26e11af77f056203.jpg',
 'pawpularitydataset/train/2d589fe856f7487989ac558e65cc213b.jpg']
```

In [ ]:

```
len(train_imgs)
```

Out[ ]:

```
9912
```

In [ ]:

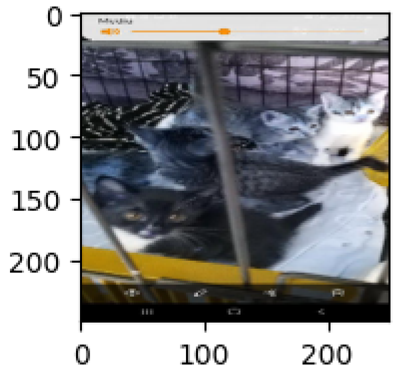
```
# declare our image dimensions using color images

img_size = 250
channels = 3  # change to 1 if need to use grayscale image

# define function to read and process the images to an acceptable format for our model
train_score = pd.read_csv('pawpularitydataset/train.csv') # the pawpularity score is in this csv file

def read_and_process_image(list_of_images):
    X = [] # an array of resized images
    y = [] # an array of score

    for i, image in enumerate(list_of_images):
        X.append(cv2.resize(cv2.imread(image, cv2.IMREAD_COLOR), (img_size, img_size), interpolation=cv2.INTER_CUBIC))
        y.append(train_score.Pawpularity[i]) # get the score

    return X, y
```

In [ ]:

```
# get the whole train and label data
X, y = read_and_process_image(train_imgs)
```

In [ ]:
```
len(y)
```
Out[ ]:
```
9912
```

In [ ]:
```
# randomly check one image and show to check
plt.figure(figsize=(5, 2))
plt.imshow(X[2])
```
Out[ ]:
```
<matplotlib.image.AxesImage at 0x7c6e63617730>
```



In [ ]:
```
# convert list to numpy array
X = np.array(X)
y = np.array(y)
```

Mount my Google drive to save the processed training arrays and labels

So that I only need to upload the array.npy and label.npy to kaggle to train the model instead of redo data preprocessing everytime.

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```
```
Mounted at /content/drive
```

In [ ]:
```
# save to google drive so that don't need to load the image each time
np.save("/content/drive/MyDrive/Colab Notebooks/group250/training_X.npy", X)
np.save("/content/drive/MyDrive/Colab Notebooks/group250/training_y.npy", y)
```

Done preprocessing images into arrays and save in Google drive

Let's switch to the kaggle notebook for model training

Only need to execute this notebook in kaggle

We trained the model, generated the prediction, then saved the output as submission.csv for scoring

In [37]:
```
import numpy as np
import pandas as pd
import cv2
import torchvision.transforms as transforms
import time
import os
import random
import matplotlib.pyplot as plt
%matplotlib inline
import gc # aarbage collector to clean uesless data from memory

pd.options.display.max_columns = None
pd.options.display.max_rows = None

# have checked that we can appropriately ignore warnings
import warnings
warnings.filterwarnings('ignore') # ignore warnings
```

Set up the array generating function for later usage

In [4]:
```
# for test set
# declare our image dimensions using color images

img_size = 250
channels = 3  # change to 1 if need to use grayscale image

# define function to read and process the images to an acceptable format for our model
def read_and_process_image_test(list_of_images):
    X = [] # an array of resized images
    for i, image in enumerate(list_of_images):
        X.append(cv2.resize(cv2.imread(image, cv2.IMREAD_COLOR), (img_size, img_size), interpolation=cv2.INTER_CUBIC))

    return X
```

Load in the training array and label

We have to up load them to the input section of kaggle notebook first, so we can load the preprocessed training set array:

1. image size = 250 pixel * 250 pixels
2. The categorical label has already been preprocessed using one hot encoding

In [38]:

```
X_TRAIN = np.load("/kaggle/input/training250/training_X.npy", allow_pickle = True)
y_TRAIN = np.load("/kaggle/input/training250/training_y.npy", allow_pickle = True)

print("shape of train images:", X_TRAIN.shape)
print("shape of labels:", y_TRAIN.shape)
```

```
shape of train images: (9912, 250, 250, 3)
shape of labels: (9912,)
```

In [39]:

```
# split the data into train and validation set
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X_TRAIN, y_TRAIN, test_size=0.20, random_state=2)

print("shape of train images:", X_train.shape)
print("shape of validation images:", X_val.shape)
print("shape of labels:", y_train.shape)
print("shape of labels:", y_val.shape)
```

```
shape of train images: (7929, 250, 250, 3)
shape of validation images: (1983, 250, 250, 3)
shape of labels: (7929,)
shape of labels: (1983,)
```

In [7]:

```
# set up small batch to avoid run out of memory when allocating
batch_size = 32
```

In [8]:

```
# clear memory
del X_TRAIN
del y_TRAIN
gc.collect()
```

Out[8]: 0

Image augmentation

In [10]:

```
# this would helps prevent overfitting, since we are using a small dataset
train_datagen = ImageDataGenerator(rescale = 1./255,   # scale the image between 0 and 1
                                   rotation_range = 60,
                                   width_shift_range = 1.0,
                                   height_shift_range = 1.0,
                                   shear_range = 0.4,
                                   zoom_range = [0.1, 2],
                                   horizontal_flip = True,
                                   vertical_flip = True,
                                   fill_mode='nearest')


val_datagen = ImageDataGenerator(rescale = 1./255)  # do not augment validation data. we only perform rescale

# create the image generators
train_generator = train_datagen.flow(X_train, y_train, batch_size=batch_size)
val_generator = val_datagen.flow(X_val, y_val, batch_size=batch_size)
```

Import modules for model training

In [9]:

```
import keras
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPooling2D, Dropout, Flatten, Dense, Activation
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
```

Self-built CNN

In [41]:

```
# define the function to create CNN model
```

```
def creat_model():

    model = keras.models.Sequential()

    # data_format='channels_last': so the channels(1 for grayscale/3 for RGB) will be the last dimension in input_shape
    # X_train should be: (batch_size, height, width, channels)
    # i.e., (training_data.shape[0], img_size, img_size, 1) since we have 25000 data

    # convolutional layer 1
    model.add(Conv2D(filters=32, kernel_size=3, data_format='channels_last', input_shape=(img_size, img_size, 3), paddi
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    # convolutional layer 2
    # after the 1st layer, don't need to specify the size of the input
    model.add(Conv2D(filters=64, kernel_size=3))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    # convolutional layer 3
    # after the 1st layer, don't need to specify the size of the input
    model.add(Conv2D(filters=128, kernel_size=3))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    # convolutional layer 4
    # after the 1st layer, don't need to specify the size of the input
    model.add(Conv2D(filters=256, kernel_size=3))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    # convolutional layer 5
    # after the 1st layer, don't need to specify the size of the input
    model.add(Conv2D(filters=512, kernel_size=3))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
```

```
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    # flatten layer
    model.add(Flatten())

    # dense layer 1
    model.add(Dense(units=512))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(Dropout(0.25))

    # dense layer 2
    model.add(Dense(units=256))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(Dropout(0.25))

    # dense layer 3
    model.add(Dense(units=128))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(Dropout(0.25))

    # dense layer 4
    model.add(Dense(units=64))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(Dropout(0.5))

    # dense layer 5
    model.add(Dense(units=32))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(Dropout(0.5))

    # dense layer 6, i.e. output layer (size=1 for regression)
    model.add(Dense(units=1, activation='relu'))

    # compile
    model.compile(optimizer='adam', loss='mse', metrics=[tf.keras.metrics.RootMeanSquaredError()])

    return model
```

In [ ]:
```
# define the early stopping callback
early_stopping = EarlyStopping(monitor='val_loss',
                               patience=4,
                               restore_best_weights=True)
```

In the model training phase, we only extract epochs from 8-30 and set early stop to avoid overfitting

We did this by setting initial_epoch to get more stable outcome

In [42]:
```
# creat_model() will return a cnn model initial structure
model = creat_model()

# train the model
history = model.fit(train_generator,
                    steps_per_epoch = len(X_train) // batch_size,
                    epochs=30,
                    initial_epoch = 7,
                    validation_data=val_generator,
                    validation_steps = len(X_val) // batch_size,
                    callbacks=[early_stopping])
```

```
Epoch 8/30
2023-12-05 05:41:51.776630: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout failed: INVALID_ARGUMEN
T: Size of values 0 does not match size of permutation 4 @ fanin shape insequential_1/dropout_24/dropout/SelectV2-2-Tra
nsposeNHWCToNCHW-LayoutOptimizer
247/247 [==============================] - 135s 507ms/step - loss: 1657.9318 - root_mean_squared_error: 40.7177 - val_l
oss: 1548.3010 - val_root_mean_squared_error: 39.3485
Epoch 9/30
247/247 [==============================] - 124s 503ms/step - loss: 1191.9634 - root_mean_squared_error: 34.5248 - val_l
oss: 1158.0403 - val_root_mean_squared_error: 34.0300
Epoch 10/30
247/247 [==============================] - 125s 505ms/step - loss: 750.4642 - root_mean_squared_error: 27.3946 - val_lo
ss: 666.8702 - val_root_mean_squared_error: 25.8238
Epoch 11/30
247/247 [==============================] - 125s 508ms/step - loss: 563.9544 - root_mean_squared_error: 23.7477 - val_lo
ss: 492.6856 - val_root_mean_squared_error: 22.1965
Epoch 12/30
247/247 [==============================] - 127s 513ms/step - loss: 511.8406 - root_mean_squared_error: 22.6239 - val_lo
ss: 469.2993 - val_root_mean_squared_error: 21.6633
Epoch 13/30
247/247 [==============================] - 125s 504ms/step - loss: 513.8691 - root_mean_squared_error: 22.6687 - val_lo
ss: 461.5994 - val_root_mean_squared_error: 21.4849
```

```
Epoch 14/30
247/247 [==============================] - 125s 503ms/step - loss: 516.8012 - root_mean_squared_error: 22.7333 - val_lo
ss: 456.0887 - val_root_mean_squared_error: 21.3562
Epoch 15/30
247/247 [==============================] - 126s 511ms/step - loss: 504.9437 - root_mean_squared_error: 22.4710 - val_lo
ss: 452.8730 - val_root_mean_squared_error: 21.2808
Epoch 16/30
247/247 [==============================] - 126s 510ms/step - loss: 503.6738 - root_mean_squared_error: 22.4427 - val_lo
ss: 444.9966 - val_root_mean_squared_error: 21.0949
Epoch 17/30
247/247 [==============================] - 127s 513ms/step - loss: 499.0487 - root_mean_squared_error: 22.3394 - val_lo
ss: 437.3275 - val_root_mean_squared_error: 20.9124
Epoch 18/30
247/247 [==============================] - 127s 513ms/step - loss: 504.1559 - root_mean_squared_error: 22.4534 - val_lo
ss: 440.0754 - val_root_mean_squared_error: 20.9780
Epoch 19/30
247/247 [==============================] - 127s 514ms/step - loss: 494.2424 - root_mean_squared_error: 22.2316 - val_lo
ss: 464.9687 - val_root_mean_squared_error: 21.5631
Epoch 20/30
247/247 [==============================] - 126s 509ms/step - loss: 494.3918 - root_mean_squared_error: 22.2349 - val_lo
ss: 443.1474 - val_root_mean_squared_error: 21.0511
Epoch 21/30
247/247 [==============================] - 126s 508ms/step - loss: 494.9007 - root_mean_squared_error: 22.2464 - val_lo
ss: 445.7154 - val_root_mean_squared_error: 21.1120
```

Save the model for future useage

In [ ]:

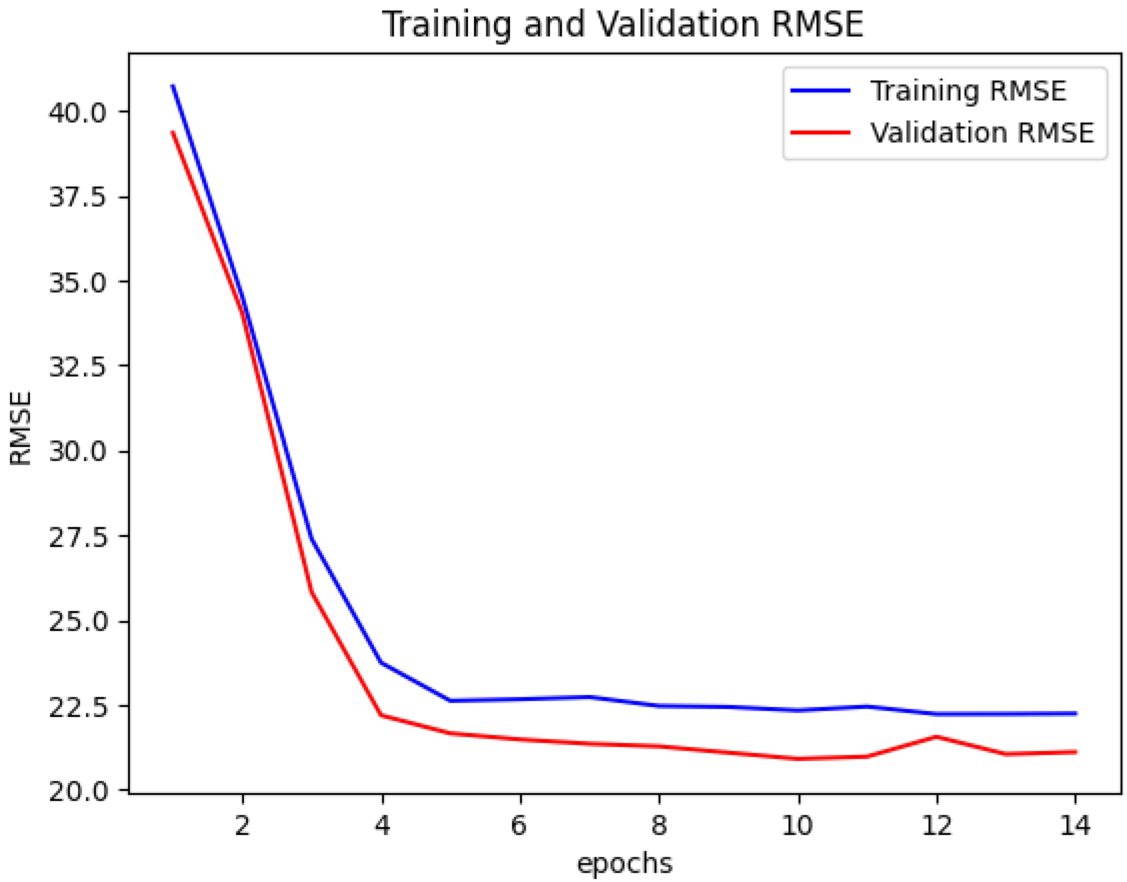
```
# save the entire model as a `.keras` zip archive
model.save('1204_my_model_8-30_epoch.keras')
```

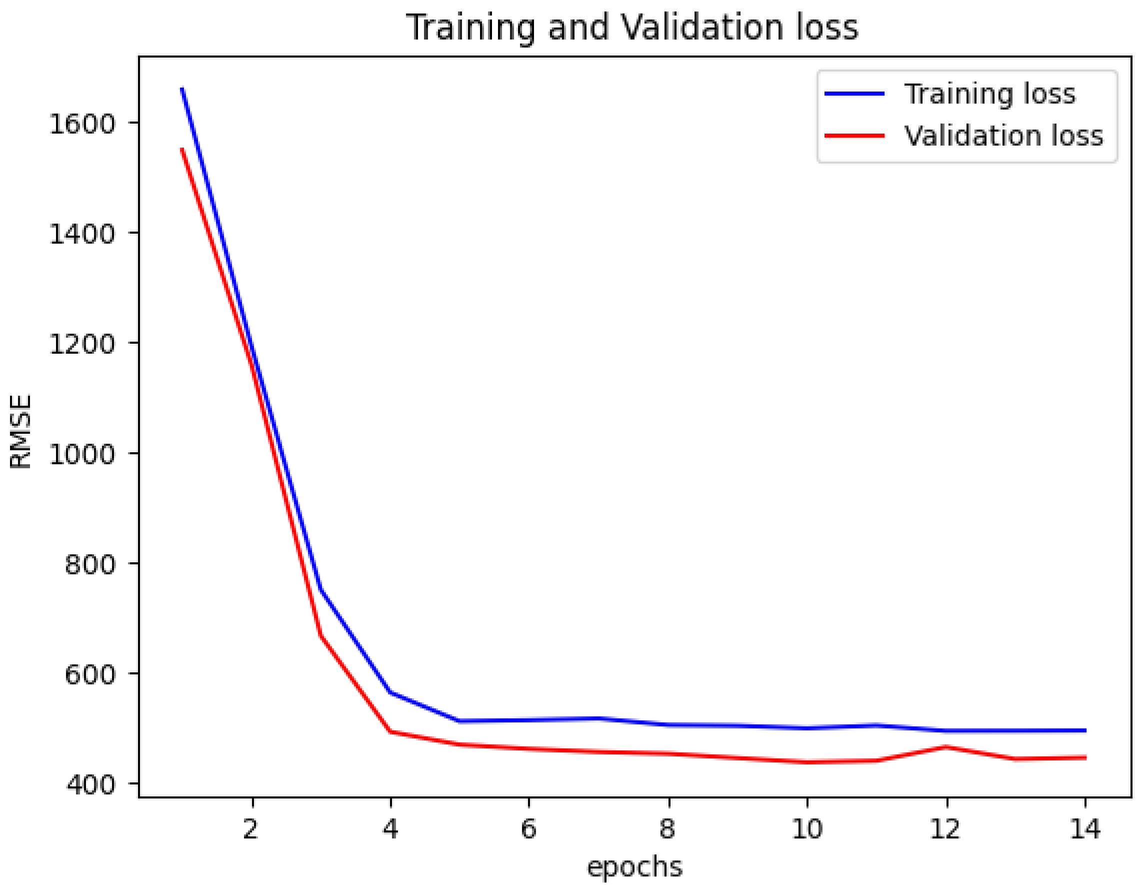
Visualize training and validation loss

In [43]:

```
# plot the train and val curve

rmse = history.history['root_mean_squared_error']
val_rmse = history.history['val_root_mean_squared_error']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(rmse) + 1)

#Train and validation accuracy
plt.plot(epochs, rmse, 'b', label='Training RMSE')
```

```
plt.plot(epochs, val_rmse, 'r', label='Validation RMSE')
plt.title('Training and Validation RMSE')
plt.xlabel('epochs')
plt.ylabel('RMSE')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('epochs')
plt.ylabel('RMSE')
plt.legend()

plt.show()
```


Training and Validation RMSE
Training RMSE
Validation RMSE
RMSE
epochs
40.0
37.5
35.0
32.5
30.0
27.5
25.0
22.5
20.0
2
4
6
8
10
12
14



Summary for model training

1. Loss in train set and validation set decreased gradually in similar patterns as training epochs increase.
2. The training process was interrupted by early stopping after 21 epochs, so we got 8-21 epochs.
3. We can see that validation loss is around 21, which might not be good enough, so we want to try the pretrainined model under Keras structure.

Pretrained Keras model: EfficientNetB4 (version for regression)

Among all the model we've tried (listed in the appendix), this model gave us the best result in test set

Since the submission of the notebook can't connect to the internet, we can't use the code like:
" pretrained_model = EfficientNetB4(include_top = False, weights = 'imagenet', input_shape=(img_size, img_size, 3)) "
since this will need to connect to ImageNet to load the weight

Get the model path under kaggle notebook

In [14]:
```
model_name = "efficientnetv2-b4"

model_handle_map = {'efficientnetv2-b4': '/kaggle/input/efficientnet/tensorflow2/b4-feature-vector/1'}

model_image_size_map = {
  "efficientnetv2-b4": img_size,
}

model_handle = model_handle_map.get(model_name)
pixels = model_image_size_map.get(model_name, img_size)

print(f"selected model path: {model_name} : {model_handle}")

IMAGE_SIZE = (pixels, pixels)
print(f"input size {IMAGE_SIZE}")
```

```
selected model path: efficientnetv2-b4 : /kaggle/input/efficientnet/tensorflow2/b4-feature-vector/1
input size (250, 250)
```

In [15]:
```
# check the imput shape
IMAGE_SIZE + (3,)
```

Out[15]:
```
(250, 250, 3)
```

In [17]:
```
import tensorflow_hub as hub

model = tf.keras.Sequential([
    # explicitly define the input shape: (250, 250, 3)
    tf.keras.layers.InputLayer(input_shape=IMAGE_SIZE + (3,)),

    # set trainable = True for fine tune
    hub.KerasLayer(model_handle, trainable = True),

    # dense layer 1
```

```
    tf.keras.layers.Dense(units=32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 2
    tf.keras.layers.Dense(units=64),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 3
    tf.keras.layers.Dense(units=128),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("leaky_relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 4
    tf.keras.layers.Dense(units=256),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 5
    tf.keras.layers.Dense(units=512),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("leaky_relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 6
    tf.keras.layers.Dense(units=256),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 7
    tf.keras.layers.Dense(units=128),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 8
    tf.keras.layers.Dense(units=64),
    tf.keras.layers.BatchNormalization(),
```

```
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 9
    tf.keras.layers.Dense(units=32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 10
    tf.keras.layers.Dense(units=512),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("leaky_relu"),
    tf.keras.layers.Dropout(0.5),

    # dense layer 11
    tf.keras.layers.Dense(units=256),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("leaky_relu"),
    tf.keras.layers.Dropout(0.6),

    # dense layer 12
    tf.keras.layers.Dense(units=128),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("leaky_relu"),
    tf.keras.layers.Dropout(0.7),

    # dense layer 13
    tf.keras.layers.Dense(units=64),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.Dropout(0.25),

    # dense layer 14
    tf.keras.layers.Dense(units=32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.Dropout(0.25),

    # dense layer 15, output layer
    tf.keras.layers.Dense(units=1, activation='relu') # dimension for output is 1 for regression problem
])
model.build((None,)+IMAGE_SIZE+(3,))
```

```python
# check the model structure
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 keras_layer_1 (KerasLayer)  (None, 1792)              17673816  
                                                                 
 dense_11 (Dense)            (None, 32)                57376     
                                                                 
 batch_normalization_10 (Ba  (None, 32)                128       
 tchNormalization)                                               
                                                                 
 activation_10 (Activation)  (None, 32)                0         
                                                                 
 dropout_10 (Dropout)        (None, 32)                0         
                                                                 
 dense_12 (Dense)            (None, 64)                2112      
                                                                 
 batch_normalization_11 (Ba  (None, 64)                256       
 tchNormalization)                                               
                                                                 
 activation_11 (Activation)  (None, 64)                0         
                                                                 
 dropout_11 (Dropout)        (None, 64)                0         
                                                                 
 dense_13 (Dense)            (None, 128)               8320      
                                                                 
 batch_normalization_12 (Ba  (None, 128)               512       
 tchNormalization)                                               
                                                                 
 activation_12 (Activation)  (None, 128)               0         
                                                                 
 dropout_12 (Dropout)        (None, 128)               0         
                                                                 
 dense_14 (Dense)            (None, 256)               33024     
                                                                 
 batch_normalization_13 (Ba  (None, 256)               1024      
 tchNormalization)                                               
                                                                 
 activation_13 (Activation)  (None, 256)               0         
                                                                 
 dropout_13 (Dropout)        (None, 256)               0         
                                                                 
 dense_15 (Dense)            (None, 512)               131584    
                                                                 
 batch_normalization_14 (Ba  (None, 512)               2048      
```

```
 tchNormalization)

 activation_14 (Activation)  (None, 512)               0

 dropout_14 (Dropout)        (None, 512)               0

 dense_16 (Dense)            (None, 256)               131328

 batch_normalization_15 (Ba  (None, 256)               1024
 tchNormalization)

 activation_15 (Activation)  (None, 256)               0

 dropout_15 (Dropout)        (None, 256)               0

 dense_17 (Dense)            (None, 128)               32896

 batch_normalization_16 (Ba  (None, 128)               512
 tchNormalization)

 activation_16 (Activation)  (None, 128)               0

 dropout_16 (Dropout)        (None, 128)               0

 dense_18 (Dense)            (None, 64)                8256

 batch_normalization_17 (Ba  (None, 64)                256
 tchNormalization)

 activation_17 (Activation)  (None, 64)                0

 dropout_17 (Dropout)        (None, 64)                0

 dense_19 (Dense)            (None, 32)                2080

 batch_normalization_18 (Ba  (None, 32)                128
 tchNormalization)

 activation_18 (Activation)  (None, 32)                0

 dropout_18 (Dropout)        (None, 32)                0

 dense_20 (Dense)            (None, 512)               16896

 batch_normalization_19 (Ba  (None, 512)               2048
 tchNormalization)
```

```
 activation_19 (Activation)  (None, 512)               0

 dropout_19 (Dropout)        (None, 512)               0

 dense_21 (Dense)            (None, 256)               131328

 batch_normalization_20 (Ba  (None, 256)               1024
 tchNormalization)

 activation_20 (Activation)  (None, 256)               0

 dropout_20 (Dropout)        (None, 256)               0

 dense_22 (Dense)            (None, 128)               32896

 batch_normalization_21 (Ba  (None, 128)               512
 tchNormalization)

 activation_21 (Activation)  (None, 128)               0

 dropout_21 (Dropout)        (None, 128)               0

 dense_23 (Dense)            (None, 64)                8256

 batch_normalization_22 (Ba  (None, 64)                256
 tchNormalization)

 activation_22 (Activation)  (None, 64)                0

 dropout_22 (Dropout)        (None, 64)                0

 dense_24 (Dense)            (None, 32)                2080

 batch_normalization_23 (Ba  (None, 32)                128
 tchNormalization)

 activation_23 (Activation)  (None, 32)                0

 dropout_23 (Dropout)        (None, 32)                0

 dense_25 (Dense)            (None, 1)                 33

=================================================================
Total params: 18282137 (69.74 MB)
Trainable params: 18152009 (69.24 MB)
Non-trainable params: 130128 (508.31 KB)
_________________________________________________________________
```

In [18]:

```
# compile the model, using 'mse' as loss function
# since this version is dealing with regression problem
model.compile(optimizer='nadam',
              loss='mse',
              metrics=[tf.keras.metrics.RootMeanSquaredError()])
```

In the model training phase, we only extract epochs from 8-30 and set early stop to avoid overfitting

We did this by setting initial_epoch to get more stable outcome

In [20]:

```
# train the model
# we still use the early stopping as defined previously
history = model.fit(
    train_generator,
    steps_per_epoch = len(X_train) // batch_size,
    epochs=30,
    initial_epoch = 7,
    validation_data=val_generator,
    validation_steps = len(X_val) // batch_size,
    callbacks=[early_stopping])
```

```
Epoch 8/30
247/247 [==============================] - 314s 590ms/step - loss: 1669.7002 - root_mean_squared_error: 40.8588 - val_l
oss: 1848.7576 - val_root_mean_squared_error: 42.9941
Epoch 9/30
247/247 [==============================] - 141s 567ms/step - loss: 1192.0747 - root_mean_squared_error: 34.5226 - val_l
oss: 80995.6250 - val_root_mean_squared_error: 284.5968
Epoch 10/30
247/247 [==============================] - 141s 571ms/step - loss: 717.9319 - root_mean_squared_error: 26.7892 - val_lo
ss: 583.4852 - val_root_mean_squared_error: 24.1498
Epoch 11/30
247/247 [==============================] - 141s 569ms/step - loss: 506.7095 - root_mean_squared_error: 22.5041 - val_lo
ss: 426.6828 - val_root_mean_squared_error: 20.6496
Epoch 12/30
247/247 [==============================] - 139s 560ms/step - loss: 468.0057 - root_mean_squared_error: 21.6270 - val_lo
ss: 427.3836 - val_root_mean_squared_error: 20.6665
Epoch 13/30
247/247 [==============================] - 140s 566ms/step - loss: 470.0680 - root_mean_squared_error: 21.6746 - val_lo
ss: 426.0973 - val_root_mean_squared_error: 20.6353
Epoch 14/30
247/247 [==============================] - 140s 566ms/step - loss: 458.6443 - root_mean_squared_error: 21.4094 - val_lo
ss: 438.4631 - val_root_mean_squared_error: 20.9327
```

```
Epoch 15/30
247/247 [==============================] - 141s 571ms/step - loss: 461.9837 - root_mean_squared_error: 21.4871 - val_lo
ss: 428.4690 - val_root_mean_squared_error: 20.6925
Epoch 16/30
247/247 [==============================] - 141s 570ms/step - loss: 468.0606 - root_mean_squared_error: 21.6281 - val_lo
ss: 427.2888 - val_root_mean_squared_error: 20.6640
Epoch 17/30
247/247 [==============================] - 145s 586ms/step - loss: 457.7862 - root_mean_squared_error: 21.3892 - val_lo
ss: 587.8098 - val_root_mean_squared_error: 24.2388
```

Save the model for future useage

In [24]:

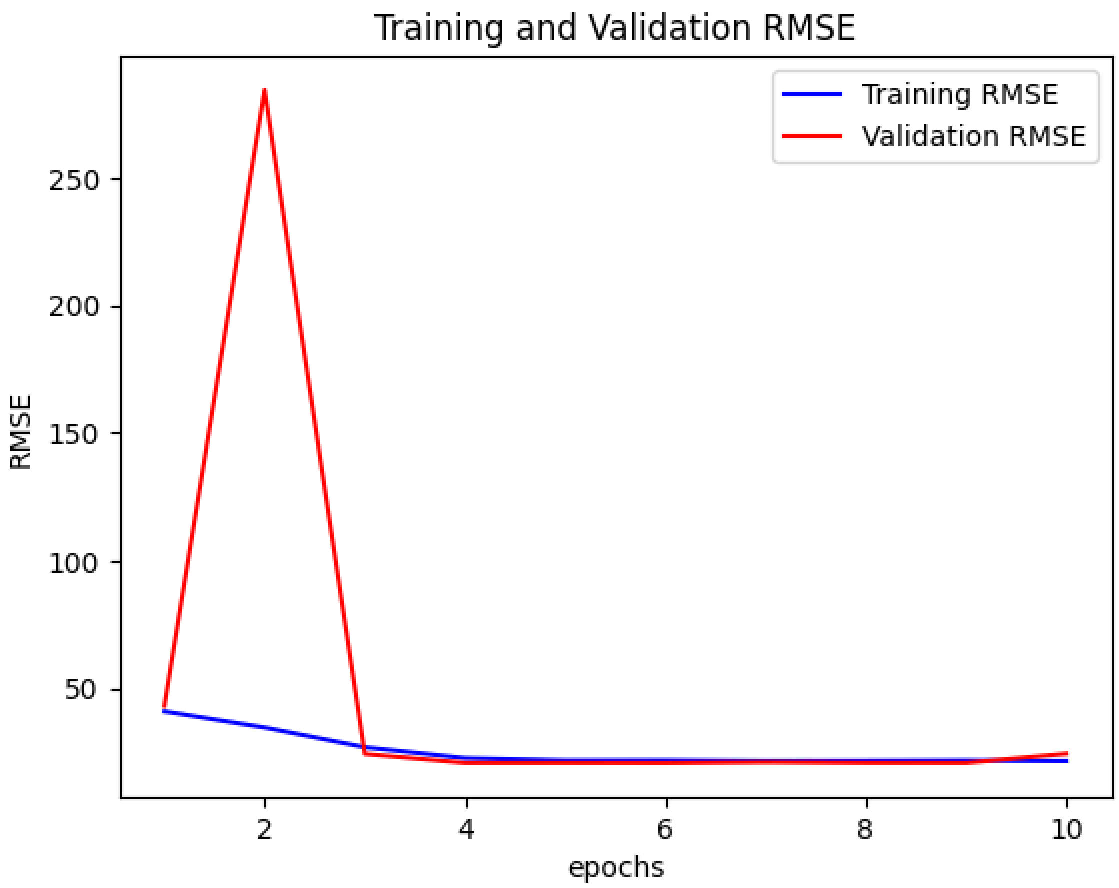
```
# save the entire model as a `.keras` zip archive
model.save('/kaggle/working/1204_efficientnetv2-b4_8-30_epochs_early_stop.keras')
```

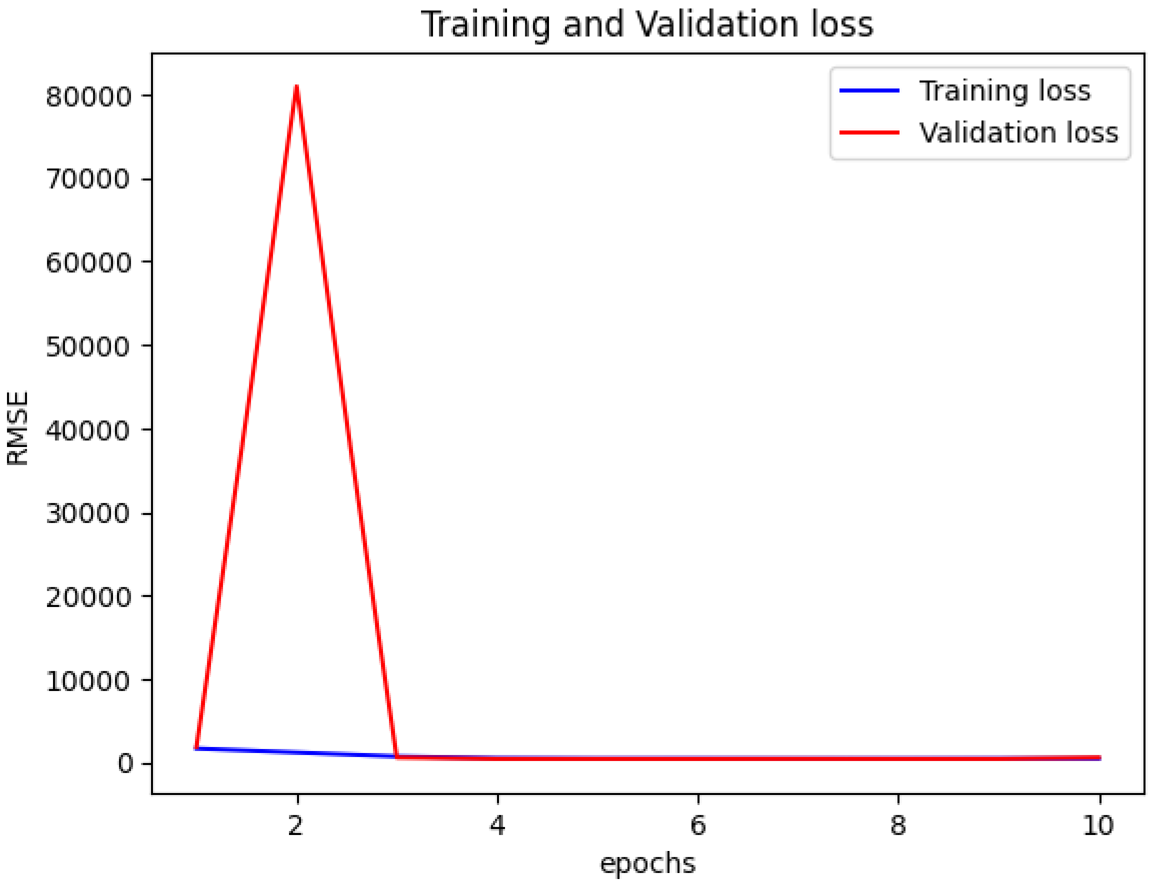
Visualize training and validation loss

In [23]:

```
# plot the train and val curve

rmse = history.history['root_mean_squared_error']
val_rmse = history.history['val_root_mean_squared_error']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(rmse) + 1)

#Train and validation accuracy
plt.plot(epochs, rmse, 'b', label='Training RMSE')
plt.plot(epochs, val_rmse, 'r', label='Validation RMSE')
plt.title('Training and Validation RMSE')
plt.xlabel('epochs')
plt.ylabel('RMSE')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('epochs')
plt.ylabel('RMSE')
plt.legend()
```

```
plt.show()
```


Training and Validation RMSE
Training RMSE
Validation RMSE
RMSE
250
200
150
100
50
2
4
6
8
10
epochs



Summary for model training

1. Loss in training set and validation set follows similar patterns, both skyrocket in the second epoch and falls back. That's why we don't want to use the first several epochs; they are unstable.
2. The training process was interrupted by early stopping after 21 epochs, so we got 8-21 epochs.
3. The performance in RMSE is better than self-built CNN since they are around 20 (just interrupt when rising to 24). So we decided to use this model for prediction.

Do prediciton on testing set

In [25]:
```
# load the model for next time
reloaded_model = tf.keras.models.load_model('/kaggle/working/1204_efficientnetv2-b4_8-30_epochs_early_stop.keras')
```

Read in testing images

In [26]:
```
test_dir = '/kaggle/input/petfinder-pawpularity-score/test'
test_imgs = ['/kaggle/input/petfinder-pawpularity-score/test/{}'.format(i) for i in os.listdir(test_dir)] # get test im
```

In [27]:
```
# process the test set
X_test = read_and_process_image_test(test_imgs)

# convert list to numpy array
X_test = np.array(X_test)

# augmentation
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow(X_test) # after rescaling for the colors
```

Set up Id for DataFrame building

In [28]:
```
Id = []

for i in range(len(test_imgs)):
    id = test_imgs[i].split('test/')[1].split('.')[0]
    Id.append(id)
```

In [29]:
```
Id
```

Out[29]:
```
['c978013571258ed6d4637f6e8cc9d6a3',
 '4e429cead1848a298432a0acad014c9d',
 '43a2262d7738e3d420d453815151079e',
 '8f49844c382931444e68dffbe20228f4',
 '4128bae22183829d2b5fea10effdb0c3',
 '80bc3ccafcc51b66303c2c263aa38486',
 'e0de453c1bffc20c22b072b34b54e50f',
 'b03f7041962238a7c9d6537e22f9b017']
```

Predict on test set

In [30]:
```
outcome = reloaded_model.predict(test_generator)
```

```
1/1 [==============================] - 3s 3s/step
```

Construct the Pawpularity array to record the scores

In [31]:
```
Pawpularity = []

for i in range(len(test_imgs)):
    pawpularity = outcome[i].item()
    Pawpularity.append(pawpularity)
```

In [32]:
```
Pawpularity
```

Out[32]:
```
[38.98554611206055,
 38.98340606689453,
 38.982887268066406,
 38.971656799316406,
 38.984466552734375,
 38.98414611816406,
 38.97465133666992,
 38.986270904541016]
```

Now let's build the dataframe to save as a csv file

In [35]:
```
dic = {'Id': Id, 'Pawpularity': Pawpularity}
result = pd.DataFrame(dic)
result.head(10)
```

Out[35]:

	Id	Pawpularity
0	c978013571258ed6d4637f6e8cc9d6a3	38.985546
1	4e429cead1848a298432a0acad014c9d	38.983406
2	43a2262d7738e3d420d453815151079e	38.982887

	Id	Pawpularity
3	8f49844c382931444e68dffbe20228f4	38.971657
4	4128bae22183829d2b5fea10effdb0c3	38.984467
5	80bc3ccafcc51b66303c2c263aa38486	38.984146
6	e0de453c1bffc20c22b072b34b54e50f	38.974651
7	b03f7041962238a7c9d6537e22f9b017	38.986271

pawpularity - Version 49

Succeeded (after deadline) · 13h ago · Notebook pawpularity | Version 49

20.51024 20.51161

The grade we get in the end, the best among all other models we've tried

In [36]:

```
# save the DataFrame to a csv file for submission
result.to_csv('submission.csv', index=False)
```

This is the end of the notebook

Generate training and testing sets in Google Colab

(I prepare the training and testing sets through Google Colab since I want to save the GPU in kaggle for model training.)

In [ ]:
```
# Easiest way to download kaggle data in Google Colab: https://www.kaggle.com/discussions/general/74235

# 1. Go to your account, Scroll to API section and Click Expire API Token to remove previous tokens
# 2. Click on Create New API Token - It will download kaggle.json file on your machine
# 3. Go to your Google Colab project file and run the following commands
```

In [1]:
```
# ! pip install -q kaggle
from google.colab import files
files.upload() # need to choose the file you've downloaded from
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[1]:
{'kaggle.json': b'{"username":"weichunchang2000","key":"773179abc6899133f0e9962470ce127f"}'}

In [2]:
```
# make directory named kaggle and copy kaggle.json file there
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/

# change the permissions of the file
! chmod 600 ~/.kaggle/kaggle.json

# list the dataset
! kaggle datasets list
```

```
ref                                              title                                                 size  lastUpdated
downloadCount  voteCount  usabilityRating
-----------------------------------------------  ---------------------------------------------------  -----  ------------
-------  -------------  ---------  ---------------
thedrcat/daigt-v2-train-dataset                  DAIGT V2 Train Dataset                                29MB  2023-11-16 0
1:38:36           1220         134  1.0
muhammadbinimran/housing-price-prediction-data   Housing Price Prediction Data                        763KB  2023-11-21 1
7:56:32           4736          89  1.0
```

```
carlmcbrideellis/llm-7-prompt-training-dataset  LLM: 7 prompt training dataset                        41MB  2023-11-15 0
7:32:56            1504         115  1.0
thedrcat/daigt-proper-train-dataset             DAIGT Proper Train Dataset                           119MB  2023-11-05 1
4:03:25            1459         134  1.0
joebeachcapital/30000-spotify-songs             30000 Spotify Songs                                    3MB  2023-11-01 0
6:06:43            9888         211  1.0
jacksondivakarr/laptop-price-prediction-dataset Laptop Price Prediction Dataset                      119KB  2023-11-30 1
6:23:34             813          29  1.0
ddosad/auto-sales-data                          Automobile Sales data                                 79KB  2023-11-18 1
2:36:41            3860          69  1.0
julnazz/diabetes-health-indicators-dataset      Diabetes Health Indicators Dataset                     5MB  2023-11-27 0
7:10:53             853          21  1.0
nelgiriyewithana/world-educational-data         World Educational Data                                 9KB  2023-11-04 0
6:10:17            7852         163  1.0
thedevastator/bank-term-deposit-predictions     Bank Term Deposit Predictions                        541KB  2023-11-30 1
4:37:39             849          29  1.0
sujaykapadnis/products-datasets                 Detailed Products Datasets                           100KB  2023-11-24 0
3:25:10            1070          26  1.0
maso0dahmed/video-games-data                    Video Games Data                                       5MB  2023-11-25 1
9:08:46            1214          36  1.0
alejopaullier/daigt-external-dataset            DAIGT | External Dataset                               3MB  2023-10-31 1
9:11:35            1004         122  0.7647059
nelgiriyewithana/australian-vehicle-prices      Australian Vehicle Prices                            582KB  2023-11-27 0
4:51:30            1126          44  1.0
prasad22/healthcare-dataset                     🩺Healthcare Dataset 🧪                                  483KB  2023-10-31
11:30:58            7027         109  1.0
adampq/linkedin-jobs-machine-learning-data-set  LinkedIn Job Postings - Machine Learning Data Set     38MB  2023-11-28 1
7:18:04             437          25  1.0
jacksondivakarr/online-shopping-dataset         🛒 Online Shopping Dataset 📊📉📈                          5MB  2023-11-
12 12:35:58            4083          76  1.0
asimislam/30-yrs-stock-market-data              30 yrs Stock Market Data                             882KB  2023-11-29 2
0:18:02            1081          27  1.0
imtkaggleteam/life-expectancy                   Life Expectancy                                      730KB  2023-11-30 1
2:22:23             621          33  0.9411765
muhammadbinimran/covid-19-pandemic-data         COVID-19 Pandemic Data                                457B  2023-11-07 2
0:42:55            1012          24  0.9411765
```

In [3]:

```
# ! kaggle competitions download -c 'name-of-competition', you will find this in each competition
! kaggle competitions download -c petfinder-pawpularity-score
```

```
Downloading petfinder-pawpularity-score.zip to /content
100% 983M/983M [00:51<00:00, 25.9MB/s]
100% 983M/983M [00:51<00:00, 20.1MB/s]
```

In [ ]:
```
! rm -r pawpularitydataset # remove the directory if needed to rerun

! mkdir pawpularitydataset
! unzip petfinder-pawpularity-score.zip -d pawpularitydataset
```

In [ ]:
```
! ls pawpularitydataset
```

```
sample_submission.csv  test  test.csv  train  train.csv
```

In [ ]:
```
! ls pawpularitydataset/train | head -n 10
```

```
0007de18844b0dbbb5e1f607da0606e0.jpg
0009c66b9439883ba2750fb825e1d7db.jpg
0013fd999caf9a3efe1352ca1b0d937e.jpg
0018df346ac9c1d8413cfcc888ca8246.jpg
001dc955e10590d3ca4673f034feeef2.jpg
001dd4f6fafb890610b1635f967ea081.jpg
0023b8a3abc93c712edd6120867deb53.jpg
0031d6a9ef7340f898c3e05f92c7bb04.jpg
0042bc5bada6d1cf8951f8f9f0d399fa.jpg
0049cb81313c94fa007286e9039af910.jpg
```

In [ ]:
```
import numpy as np
import pandas as pd
import cv2
import os
import matplotlib.pyplot as plt
%matplotlib inline
import gc # garbage collector for cleaning deleted data from memory

pd.options.display.max_columns = None
pd.options.display.max_rows = None
```

We don't deal with the test set since we will submit the notebook for grading and will load the test set then

In [ ]:
```
train_imgs = [] # just to initialize in case I need to rerun
```

```
train_dir = 'pawpularitydataset/train'
train_imgs = ['pawpularitydataset/train/{}'.format(i) for i in os.listdir(train_dir)] # get train images
```

In [ ]:
```
train_imgs[:10]
```

Out[ ]:
```
['pawpularitydataset/train/15c681c62392f2ee73ee0087f37ddeaf.jpg',
 'pawpularitydataset/train/4130c0acf816e5b857a7217805da7f13.jpg',
 'pawpularitydataset/train/db26ad9754421faec035456f15269f52.jpg',
 'pawpularitydataset/train/f5f53baf396fee9ee0d51cf0ca5701cf.jpg',
 'pawpularitydataset/train/fc00c2d6b03a78ddd12cde5716c5b0ab.jpg',
 'pawpularitydataset/train/dc978e94fb761b9ee01b0595a2e3b9c8.jpg',
 'pawpularitydataset/train/1c8284661c5c710cd1bd517d5c3e0f63.jpg',
 'pawpularitydataset/train/d3df7802063d5cd7df72a873824015b2.jpg',
 'pawpularitydataset/train/2da1d3fb0dff907c26e11af77f056203.jpg',
 'pawpularitydataset/train/2d589fe856f7487989ac558e65cc213b.jpg']
```

In [ ]:
```
len(train_imgs)
```

Out[ ]:
```
9912
```

In [ ]:
```
# declare our image dimensions using color images

img_size = 250
channels = 3  # change to 1 if need to use grayscale image

# define function to read and process the images to an acceptable format for our model
train_score = pd.read_csv('pawpularitydataset/train.csv') # the pawpularity score is in this csv file

def read_and_process_image(list_of_images):
    X = [] # an array of resized images
    y = [] # an array of score

    for i, image in enumerate(list_of_images):
        X.append(cv2.resize(cv2.imread(image, cv2.IMREAD_COLOR), (img_size, img_size), interpolation=cv2.INTER_CUBIC))
        y.append(train_score.Pawpularity[i]) # get the score

    return X, y
```

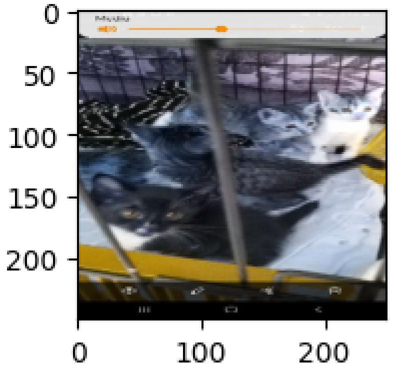
In [ ]:
```
# get the whole train and label data
X, y = read_and_process_image(train_imgs)
```

In [ ]:
```
len(y)
```

Out[ ]: 9912

In [ ]:
```
# randomly check one image and show to check
plt.figure(figsize=(5, 2))
plt.imshow(X[2])
```

Out[ ]: <matplotlib.image.AxesImage at 0x7c6e63617730>



In [ ]:
```
# convert list to numpy array
X = np.array(X)
y = np.array(y)
```

Mount my Google drive to save the processed training arrays and labels

So that I only need to upload the array.npy and label.npy to kaggle to train the model instead of redo data preprocessing everytime.

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
# save to google drive so that don't need to load the image each time
np.save("/content/drive/MyDrive/Colab Notebooks/group250/training_X.npy", X)
np.save("/content/drive/MyDrive/Colab Notebooks/group250/training_y.npy", y)
```

Done preprocessing images into arrays and save in Google drive

Let's switch to the kaggle notebook for model training

This notebook is doing one hot encoding on Pawpularity score for classification using train.csv

To save time for final auto-grading in the system, we did the one hot encoding for scores in advance in this notebook

This will be used in classification version of CNN

In [1]:
```
import numpy as np
import pandas as pd
```

In [15]:
```
df = pd.read_csv('train.csv')
df.head(5)
```

Out[15]:

	Id	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur	Pawpularit
0	0007de18844b0dbbb5e1f607da0606e0	0	1	1	1	0	0	1	0	0	0	0	0	6
1	0009c66b9439883ba2750fb825e1d7db	0	1	1	0	0	0	0	0	0	0	0	0	4
2	0013fd999caf9a3efe1352ca1b0d937e	0	1	1	1	0	0	0	0	1	1	0	0	2
3	0018df346ac9c1d8413cfcc888ca8246	0	1	1	1	0	0	0	0	0	0	0	0	1
4	001dc955e10590d3ca4673f034feeef2	0	0	0	1	0	0	1	0	0	0	0	0	7

In [16]:
```
# convert the last numerical column to categorical
df['Pawpularity'] = pd.Categorical(df['Pawpularity'])
df.head(5)
```

Out[16]:

	Id	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur	Pawpularit
0	0007de18844b0dbbb5e1f607da0606e0	0	1	1	1	0	0	1	0	0	0	0	0	6
1	0009c66b9439883ba2750fb825e1d7db	0	1	1	0	0	0	0	0	0	0	0	0	4
2	0013fd999caf9a3efe1352ca1b0d937e	0	1	1	1	0	0	0	0	1	1	0	0	2

	Id	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur	Pawpularit
3	0018df346ac9c1d8413cfcc888ca8246	0	1	1	1	0	0	0	0	0	0	0	0	1
4	001dc955e10590d3ca4673f034feeef2	0	0	0	1	0	0	1	0	0	0	0	0	7

Do one hot encoding on the Pawpularity scores

In [17]:

```
# apply one-hot encoding
df_encoded = pd.get_dummies(df, columns=['Pawpularity'], prefix='score')
df_encoded.head(5)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\__init__.py:138: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is
required for this version of SciPy (detected version 1.24.4)
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required for this version of "
```

Out[17]:

	Id	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur	score_1	s
0	0007de18844b0dbbb5e1f607da0606e0	0	1	1	1	0	0	1	0	0	0	0	0	False	
1	0009c66b9439883ba2750fb825e1d7db	0	1	1	0	0	0	0	0	0	0	0	0	False	
2	0013fd999caf9a3efe1352ca1b0d937e	0	1	1	1	0	0	0	0	1	1	0	0	False	
3	0018df346ac9c1d8413cfcc888ca8246	0	1	1	1	0	0	0	0	0	0	0	0	False	
4	001dc955e10590d3ca4673f034feeef2	0	0	0	1	0	0	1	0	0	0	0	0	False	

Then use (0, 1) expression rather than (False, True)

In [19]:

```
df_encoded.replace({False: 0, True: 1}, inplace=True)
df_encoded.head(5)
```

Out[19]:

	Id	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur	score_1	s
0	0007de18844b0dbbb5e1f607da0606e0	0	1	1	1	0	0	1	0	0	0	0	0	0	
1	0009c66b9439883ba2750fb825e1d7db	0	1	1	0	0	0	0	0	0	0	0	0	0	

	Id	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur	score_1	s
2	0013fd999caf9a3efe1352ca1b0d937e	0	1	1	1	0	0	0	0	1	1	0	0	0	
3	0018df346ac9c1d8413cfcc888ca8246	0	1	1	1	0	0	0	0	0	0	0	0	0	
4	001dc955e10590d3ca4673f034feeef2	0	0	0	1	0	0	1	0	0	0	0	0	0	

Save the file for further usage

In [20]:
```
df_encoded.to_csv('with_category_score.csv', index=True)
```

This is the end of the notebook (result should be used in other notebooks)

This notebook is doing regression on Pawpularity score using train.csv

To save time for final auto-grading in the system, we did the GridSearch in advance in this notebook

We regress Pawpularity scores on feature data and train the optimized SVR model

In [ ]:

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.model_selection import GridSearchCV
```

Load in the train.csv

In [13]:

```
df = pd.read_csv('train.csv')
df = df.drop(columns = 'Id')

# extract the dependent and target variables
X_train_SVR = df.iloc[:, 0:12]
y_train_SVR = df.iloc[:, 12]

# training-validation split
X_train_svr, X_val_svr, y_train_svr, y_val_svr = train_test_split(X_train_SVR, y_train_SVR, test_size=0.2, random_state
```

Define the hyperparameter grid and SVR

In [ ]:

```
# define the hyperparameter grid
svr_grid = {'kernel': ['rbf'], 'C': [10, 1, 0.1], 'epsilon': [10, 1, 0.1], 'gamma': [10, 1, 0.1, 0.01]}
```

In [15]:

```
# define the svm regressor: SVR
svr = svm.SVR() # for svr, y is expected to have floating point values instead of integer values
```

Do GridSearch

In [16]:
```
svr_clf = GridSearchCV(estimator = svr, param_grid = svr_grid, scoring = 'neg_root_mean_squared_error', cv = 10, refit
svr_clf.fit(X_train_svr, y_train_svr)
print("Best hyperparameters settings: ", svr_clf.best_params_)
print('RMSE: ', -svr_clf.best_score_)
```

```
Best hyperparameters settings:  {'C': 0.1, 'epsilon': 10, 'gamma': 1, 'kernel': 'rbf'}
RMSE:  20.644052638806244
```

This is the end of the notebook (result should be used in other notebooks)

Only need to execute this notebook in kaggle

We trained the model, generated the prediction, then saved the output as submission.csv for scoring

In [2]:

```
import numpy as np
import pandas as pd
import cv2
import torchvision.transforms as transforms
import time
import os
import random
import matplotlib.pyplot as plt
%matplotlib inline
import gc # aarbage collector to clean uesless data from memory

pd.options.display.max_columns = None
pd.options.display.max_rows = None

# have checked that we can appropriately ignore warnings
import warnings
warnings.filterwarnings('ignore') # ignore warnings
```

Set up the array generating function for later usage

In [3]:

```
# for test set only
# declare our image dimensions using color images

img_size = 250
channels = 3  # change to 1 if need to use grayscale image

# define function to read and process the images to an acceptable format for our model
def read_and_process_image_test(list_of_images):
    X = [] # an array of resized images
    for i, image in enumerate(list_of_images):
        X.append(cv2.resize(cv2.imread(image, cv2.IMREAD_COLOR), (img_size, img_size), interpolation=cv2.INTER_CUBIC))

    return X
```

Load in the training array and label

We have to up load them to the input section of kaggle notebook first, so we can load the preprocessed training set array:

1. image size = 250 pixel * 250 pixels
2. The categorical label has already been preprocessed using one hot encoding

In [4]:

```
X_TRAIN = np.load("/kaggle/input/training250/training_X.npy", allow_pickle = True)
y_TRAIN = pd.read_csv("/kaggle/input/categorical-score-no-metadata/with_category_score.csv")

print("shape of train images:", X_TRAIN.shape)
print("shape of labels:", y_TRAIN.shape)
```

```
shape of train images: (9912, 250, 250, 3)
shape of labels: (9912, 100)
```

In [5]:

```
# split the data into train and validation set
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X_TRAIN, y_TRAIN, test_size=0.20, random_state=42)

print("shape of train images:", X_train.shape)
print("shape of validation images:", X_val.shape)
print("shape of labels:", y_train.shape)
print("shape of labels:", y_val.shape)
```

```
shape of train images: (7929, 250, 250, 3)
shape of validation images: (1983, 250, 250, 3)
shape of labels: (7929, 100)
shape of labels: (1983, 100)
```

In [5]:

```
# set up small batch to avoid run out of memory when allocating
batch_size = 32
```

In [6]:

```
# clear useless variables to save memory
del X_TRAIN
del y_TRAIN
gc.collect()
```

Out[6]: 4

Image augmentation

In [8]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# this would helps prevent overfitting, since we are using a small dataset
train_datagen = ImageDataGenerator(rescale = 1./255,   # scale the image between 0 and 1
                                   rotation_range = 60,
                                   width_shift_range = 1.0,
                                   height_shift_range = 1.0,
                                   shear_range = 0.4,
                                   zoom_range = [0.1, 2],
                                   horizontal_flip = True,
                                   vertical_flip = True,
                                   fill_mode='nearest')

val_datagen = ImageDataGenerator(rescale = 1./255)  # do not augment validation data. we only perform rescale

# create the image generators
train_generator = train_datagen.flow(X_train, y_train, batch_size=batch_size)
val_generator = val_datagen.flow(X_val, y_val, batch_size=batch_size)
```

Import modules for model training

In [17]:

```
import keras
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import BatchNormalization, Dropout, Flatten, Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
```

Self-built CNN

(already be demonstrated in the regression version so skip here)

Pretrained Keras model: EfficientNetB4 (version for classification)

Among all the model we've tried (listed in the appendix), this model gave us the best result in test set

Since the submission of the notebook can't connect to the internet, we can't use the code like:
" pretrained_model = EfficientNetB4(include_top = False, weights = 'imagenet', input_shape=(img_size, img_size, 3)) "
since this will need to connect to ImageNet to load the weight

In [10]:

```
model_name = "efficientnetv2-b4"
model_handle_map = {'efficientnetv2-b4': '/kaggle/input/efficientnet/tensorflow2/b4-classification/1'}
model_image_size_map = {"efficientnetv2-b4": img_size}

model_handle = model_handle_map.get(model_name)
pixels = model_image_size_map.get(model_name, img_size)

print(f"selected model path: {model_name} : {model_handle}")

IMAGE_SIZE = (pixels, pixels)
print('input size:', IMAGE_SIZE)
```

```
selected model path: efficientnetv2-b4 : /kaggle/input/efficientnet/tensorflow2/b4-classification/1
input size (250, 250)
```

(The units and droupout rate is randomly assigned to the layers)

In [11]:

```
import tensorflow_hub as hub

model = tf.keras.Sequential([
    # explicitly define the input shape: (250, 250, 3)
    tf.keras.layers.InputLayer(input_shape=IMAGE_SIZE + (3,)),

    # set trainable = True for fine tune
    hub.KerasLayer(model_handle, trainable = True),

    # pooling and flaten layers are skipped in this version of pretrained model loading structure

    # dense layer 1
    tf.keras.layers.Dense(units=512),
```

```
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation("relu"),
        tf.keras.layers.Dropout(0.5),

        # dense layer 2
        tf.keras.layers.Dense(units=256),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation("relu"),
        tf.keras.layers.Dropout(0.5),

        # dense layer 3
        tf.keras.layers.Dense(units=128),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation("relu"),
        tf.keras.layers.Dropout(0.5),

        # dense layer 4
        tf.keras.layers.Dense(units=64),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation("relu"),
        tf.keras.layers.Dropout(0.5),

        # dense layer 5
        tf.keras.layers.Dense(units=32),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation("relu"),
        tf.keras.layers.Dropout(0.5),

        # dense layer 6
        tf.keras.layers.Dense(units=512),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation("leaky_relu"),
        tf.keras.layers.Dropout(0.5),

        # dense layer 7
        tf.keras.layers.Dense(units=256),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation("leaky_relu"),
        tf.keras.layers.Dropout(0.6),

        # dense layer 8
        tf.keras.layers.Dense(units=128),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation("leaky_relu"),
```

```
    tf.keras.layers.Dropout(0.7),

    # dense layer 9
    tf.keras.layers.Dense(units=64),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("leaky_relu"),
    tf.keras.layers.Dropout(0.25),

    # dense layer 10
    tf.keras.layers.Dense(units=32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("leaky_relu"),
    tf.keras.layers.Dropout(0.25),

    # dense layer 6, output layer
    tf.keras.layers.Dense(units=100, activation='softmax') # we view the 100 Pawpularity score as 100 categories
])
model.build((None,)+IMAGE_SIZE+(3,))

# check the model structure
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 keras_layer (KerasLayer)    (None, 1000)              19466816  
                                                                 
 dense (Dense)               (None, 512)               512512    
                                                                 
 batch_normalization (Batch  (None, 512)               2048      
 Normalization)                                                  
                                                                 
 activation (Activation)     (None, 512)               0         
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 256)               131328    
                                                                 
 batch_normalization_1 (Bat  (None, 256)               1024      
 chNormalization)                                                
                                                                 
 activation_1 (Activation)   (None, 256)               0         
                                                                 
 dropout_1 (Dropout)         (None, 256)               0         
```

```
dense_2 (Dense)             (None, 128)               32896

batch_normalization_2 (Bat  (None, 128)               512
chNormalization)

activation_2 (Activation)   (None, 128)               0

dropout_2 (Dropout)         (None, 128)               0

dense_3 (Dense)             (None, 64)                8256

batch_normalization_3 (Bat  (None, 64)                256
chNormalization)

activation_3 (Activation)   (None, 64)                0

dropout_3 (Dropout)         (None, 64)                0

dense_4 (Dense)             (None, 32)                2080

batch_normalization_4 (Bat  (None, 32)                128
chNormalization)

activation_4 (Activation)   (None, 32)                0

dropout_4 (Dropout)         (None, 32)                0

dense_5 (Dense)             (None, 512)               16896

batch_normalization_5 (Bat  (None, 512)               2048
chNormalization)

activation_5 (Activation)   (None, 512)               0

dropout_5 (Dropout)         (None, 512)               0

dense_6 (Dense)             (None, 256)               131328

batch_normalization_6 (Bat  (None, 256)               1024
chNormalization)

activation_6 (Activation)   (None, 256)               0

dropout_6 (Dropout)         (None, 256)               0

dense_7 (Dense)             (None, 128)               32896
```

```
 batch_normalization_7 (Bat  (None, 128)              512
 chNormalization)

 activation_7 (Activation)   (None, 128)              0

 dropout_7 (Dropout)         (None, 128)              0

 dense_8 (Dense)             (None, 64)               8256

 batch_normalization_8 (Bat  (None, 64)               256
 chNormalization)

 activation_8 (Activation)   (None, 64)               0

 dropout_8 (Dropout)         (None, 64)               0

 dense_9 (Dense)             (None, 32)               2080

 batch_normalization_9 (Bat  (None, 32)               128
 chNormalization)

 activation_9 (Activation)   (None, 32)               0

 dropout_9 (Dropout)         (None, 32)               0

 dense_10 (Dense)            (None, 100)              3300

=================================================================
Total params: 20356580 (77.65 MB)
Trainable params: 20227412 (77.16 MB)
Non-trainable params: 129168 (504.56 KB)
_________________________________________________________________
```

In [12]:

```
# compile the model, using 'categorical_crossentropy' as loss function
# since this version is dealing with classification problem in multi outcomes
model.compile(optimizer='nadam',
              loss='categorical_crossentropy',
              metrics=[keras.metrics.CategoricalCrossentropy()])
```

In [13]:

```
# define the early stopping callback
early_stopping = EarlyStopping(monitor='val_loss',
                               patience=5,
                               restore_best_weights=True)
```

In the model training phase, we only extract epochs from 8-30 and set early stop to avoid overfitting

We did this by setting initial_epoch to get more stable outcome

In [14]:
```
# train the model
model.fit(train_generator,
          steps_per_epoch = len(X_train) // batch_size,
          epochs=30,
          initial_epoch = 7,
          validation_data=val_generator,
          validation_steps = len(X_val) // batch_size,
          callbacks=[early_stopping])
```

```
Epoch 8/30
247/247 [==============================] - 312s 582ms/step - loss: 4.8170 - categorical_crossentropy: 4.5373 - val_los
s: 4.6187 - val_categorical_crossentropy: 4.3359
Epoch 9/30
247/247 [==============================] - 140s 567ms/step - loss: 4.6332 - categorical_crossentropy: 4.3491 - val_los
s: 4.4987 - val_categorical_crossentropy: 4.2134
Epoch 10/30
247/247 [==============================] - 141s 570ms/step - loss: 4.5680 - categorical_crossentropy: 4.2817 - val_los
s: 4.4888 - val_categorical_crossentropy: 4.2017
Epoch 11/30
247/247 [==============================] - 141s 571ms/step - loss: 4.5461 - categorical_crossentropy: 4.2588 - val_los
s: 4.4888 - val_categorical_crossentropy: 4.2014
Epoch 12/30
247/247 [==============================] - 141s 571ms/step - loss: 4.5316 - categorical_crossentropy: 4.2436 - val_los
s: 4.4867 - val_categorical_crossentropy: 4.1991
Epoch 13/30
247/247 [==============================] - 141s 571ms/step - loss: 4.5162 - categorical_crossentropy: 4.2287 - val_los
s: 4.4962 - val_categorical_crossentropy: 4.2092
Epoch 14/30
247/247 [==============================] - 142s 572ms/step - loss: 4.5186 - categorical_crossentropy: 4.2318 - val_los
s: 4.4904 - val_categorical_crossentropy: 4.2046
Epoch 15/30
247/247 [==============================] - 141s 568ms/step - loss: 4.5082 - categorical_crossentropy: 4.2243 - val_los
s: 4.4837 - val_categorical_crossentropy: 4.2010
Epoch 16/30
247/247 [==============================] - 142s 572ms/step - loss: 4.5034 - categorical_crossentropy: 4.2226 - val_los
s: 4.4795 - val_categorical_crossentropy: 4.2010
Epoch 17/30
247/247 [==============================] - 142s 572ms/step - loss: 4.4911 - categorical_crossentropy: 4.2149 - val_los
s: 4.4729 - val_categorical_crossentropy: 4.1988
Epoch 18/30
```

```
247/247 [==============================] - 142s 571ms/step - loss: 4.4884 - categorical_crossentropy: 4.2164 - val_los
s: 4.4689 - val_categorical_crossentropy: 4.1987
Epoch 19/30
247/247 [==============================] - 141s 569ms/step - loss: 4.4778 - categorical_crossentropy: 4.2103 - val_los
s: 4.4635 - val_categorical_crossentropy: 4.1986
Epoch 20/30
247/247 [==============================] - 141s 569ms/step - loss: 4.4743 - categorical_crossentropy: 4.2121 - val_los
s: 4.4616 - val_categorical_crossentropy: 4.2022
Epoch 21/30
247/247 [==============================] - 141s 567ms/step - loss: 4.4646 - categorical_crossentropy: 4.2081 - val_los
s: 4.4539 - val_categorical_crossentropy: 4.2003
Epoch 22/30
247/247 [==============================] - 140s 567ms/step - loss: 4.4613 - categorical_crossentropy: 4.2110 - val_los
s: 4.4484 - val_categorical_crossentropy: 4.2013
Epoch 23/30
247/247 [==============================] - 140s 565ms/step - loss: 4.4464 - categorical_crossentropy: 4.2025 - val_los
s: 4.4410 - val_categorical_crossentropy: 4.2007
Epoch 24/30
247/247 [==============================] - 144s 581ms/step - loss: 4.4424 - categorical_crossentropy: 4.2055 - val_los
s: 4.4356 - val_categorical_crossentropy: 4.2020
Epoch 25/30
247/247 [==============================] - 142s 573ms/step - loss: 4.4366 - categorical_crossentropy: 4.2070 - val_los
s: 4.4237 - val_categorical_crossentropy: 4.1982
Epoch 26/30
247/247 [==============================] - 142s 571ms/step - loss: 4.4269 - categorical_crossentropy: 4.2054 - val_los
s: 4.4173 - val_categorical_crossentropy: 4.2001
Epoch 27/30
247/247 [==============================] - 142s 572ms/step - loss: 4.4205 - categorical_crossentropy: 4.2074 - val_los
s: 4.4060 - val_categorical_crossentropy: 4.1970
Epoch 28/30
247/247 [==============================] - 141s 568ms/step - loss: 4.4112 - categorical_crossentropy: 4.2051 - val_los
s: 4.4085 - val_categorical_crossentropy: 4.2062
Epoch 29/30
247/247 [==============================] - 142s 575ms/step - loss: 4.4054 - categorical_crossentropy: 4.2066 - val_los
s: 4.3917 - val_categorical_crossentropy: 4.1968
Epoch 30/30
247/247 [==============================] - 141s 568ms/step - loss: 4.3940 - categorical_crossentropy: 4.2029 - val_los
s: 4.3835 - val_categorical_crossentropy: 4.1966
```

Save the model for future useage

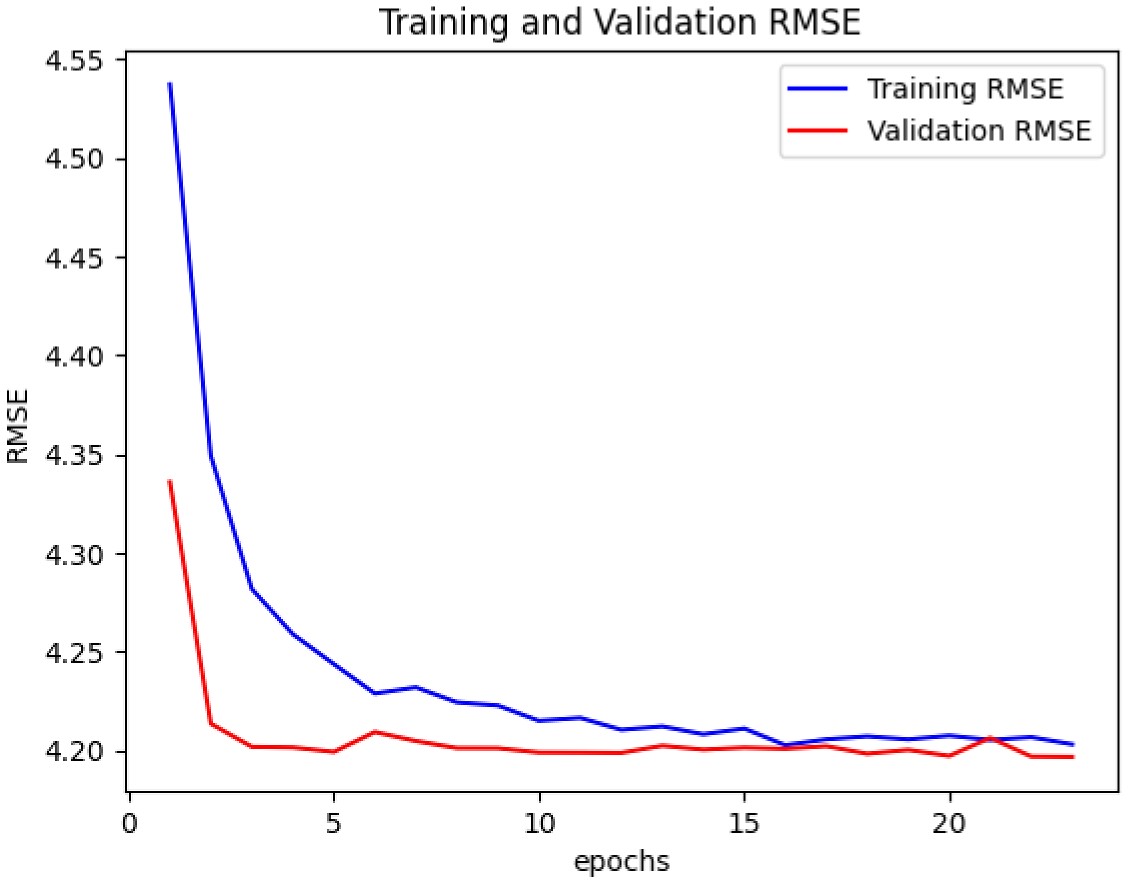
In [15]:

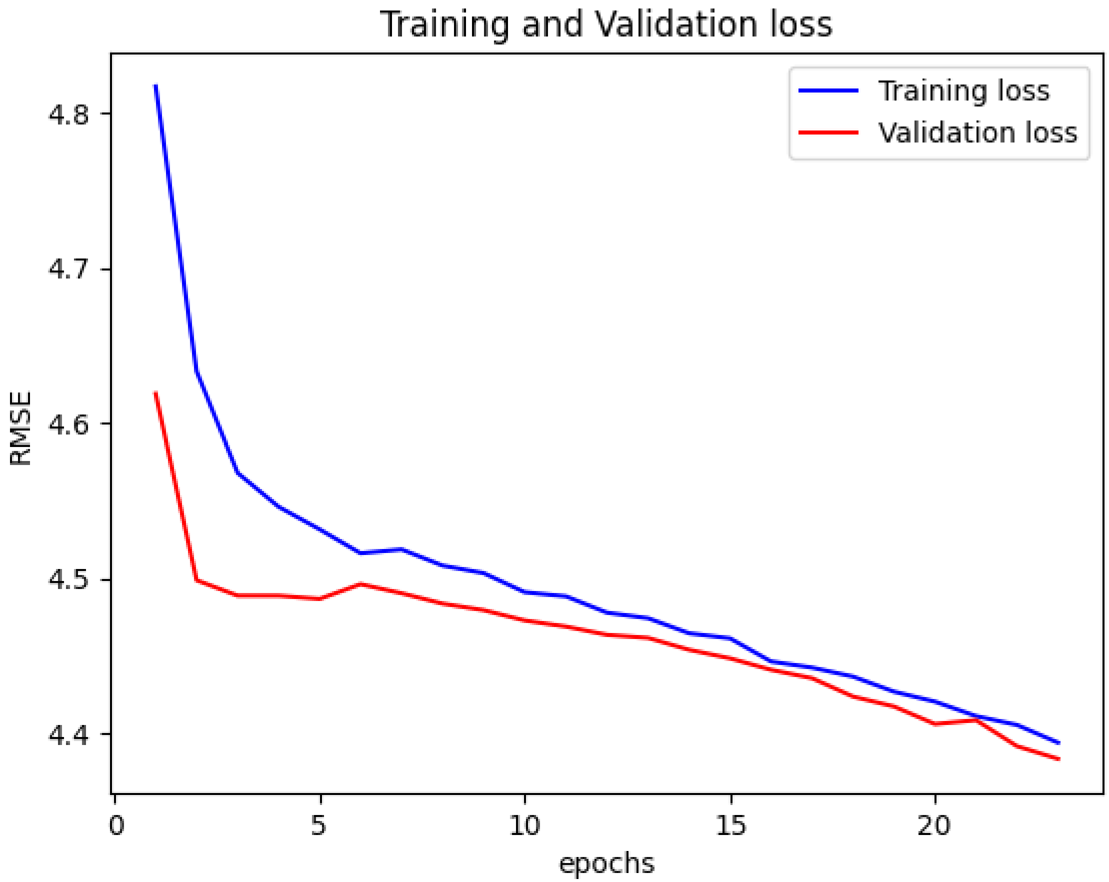
```
# save the entire model as a `.keras` zip archive
model.save('/kaggle/working/1204_efficientnetv2-b4_8-30_epochs_early_stop_classification.keras')
```

Visualize training and validation loss

In [16]:

```
# plot the train and val curve

rmse = history.history['categorical_crossentropy']
val_rmse = history.history['val_categorical_crossentropy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(rmse) + 1)

#Train and validation accuracy
plt.plot(epochs, rmse, 'b', label='Training RMSE')
plt.plot(epochs, val_rmse, 'r', label='Validation RMSE')
plt.title('Training and Validation RMSE')
plt.xlabel('epochs')
plt.ylabel('RMSE')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('epochs')
plt.ylabel('RMSE')
plt.legend()

plt.show()
```


Training and Validation RMSE
Training RMSE
Validation RMSE
RMSE
epochs
4.55
4.50
4.45
4.40
4.35
4.30
4.25
4.20
0
5
10
15
20



Summary for model training

Loss in training set and validation set decreased gradually in the similar patterns as training epochs increase, and it wasn't interrupted by early stopping.

Do prediciton on testing set

In [ ]:
```
# load the model for next time
reloaded_model = tf.keras.models.load_model('/kaggle/working/1204_efficientnetv2-b4_8-30_epochs_early_stop_classificati
```

Read in testing images

In [15]:
```
test_dir = '/kaggle/input/petfinder-pawpularity-score/test'
test_imgs = ['/kaggle/input/petfinder-pawpularity-score/test/{}'.format(i) for i in os.listdir(test_dir)] # get test im
```

In [18]:
```
# process the test set
X_test = read_and_process_image_test(test_imgs)

# convert list to numpy array
X_test = np.array(X_test)

# augmentation
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow(X_test) # after rescaling for the colors
```

Set up Id for DataFrame building

In [19]:
```
Id = []

for i in range(len(test_imgs)):
    id = test_imgs[i].split('test/')[1].split('.')[0]
    Id.append(id)
```

In [20]:
```
Id
```

Out[20]:
```
['c978013571258ed6d4637f6e8cc9d6a3',
 '4e429cead1848a298432a0acad014c9d',
 '43a2262d7738e3d420d453815151079e',
 '8f49844c382931444e68dffbe20228f4',
 '4128bae22183829d2b5fea10effdb0c3',
 '80bc3ccafcc51b66303c2c263aa38486',
 'e0de453c1bffc20c22b072b34b54e50f',
 'b03f7041962238a7c9d6537e22f9b017']
```

Predict on test set

In [22]:
```
outcome = reloaded_model.predict(test_generator)
```

```
1/1 [==============================] - 3s 3s/step
```

There're 8 images in test set, and for each image we generated the probability for it to be scored as 1 to 100

In [40]:
```
len(outcome) # we only have 8 images in test set
```

Out[40]:
```
8
```

In [42]:
```
len(outcome[0]) # and for each image, we generate the probability for it to be scored as 1 to 100, thus having length =
```

Out[42]:
```
100
```

Construct the Pawpularity array to record the scores

In [43]:
```
Pawpularity = []

for i in range(len(test_imgs)):
    pawpularity = 0
    for j in range(100):
        pawpularity = pawpularity + outcome[i][j]*(j+1)
    Pawpularity.append(pawpularity)
```

In [44]:
```
Pawpularity
```

Out[44]:
```
[38.141670293029165,
 38.141670293029165,
 38.1416702727729,
 38.141670293029165,
 38.1416702727729,
 38.1416702727729,
 38.141670293029165,
 38.1416702727729]
```

Metadata

We also combined the metadata to see if the performance could be improved

We regress Pawpularity scores on features in the table to train an SVR model then do prediction on test set, then we combine these scores with the Pawpularity scores generated from CNN by taking average

In [8]:
```
df = pd.read_csv('/kaggle/input/petfinder-pawpularity-score/train.csv')
df = df.drop(columns = 'Id')

X_train_SVR = df.iloc[:, 0:12]
y_train_SVR = df.iloc[:, 12]

from sklearn.model_selection import train_test_split
X_train_svr, X_val_svr, y_train_svr, y_val_svr = train_test_split(X_train_SVR, y_train_SVR, test_size=0.2, random_state
```

(This part was done in another notebook)

In [ ]:
```
'''svr_grid = {'kernel': ['rbf'], 'C': [10, 1, 0.1], 'epsilon': [10, 1, 0.1], 'gamma': [10, 1, 0.1, 0.01]}
svr = svm.SVR() # for svr, y is expected to have floating point values instead of integer values

svr_clf = GridSearchCV(estimator = svr, param_grid = svr_grid, scoring = 'neg_root_mean_squared_error', cv = 10, refit
svr_clf.fit(X_train_svr, y_train_svr)
print("Best hyperparameters settings: ", svr_clf.best_params_)
print('RMSE: ', -svr_clf.best_score_)'''
```

From the GridSearch in advance, we got the optimozed hyperparameters

To save time for final auto-grading in the system, we did the GridSearch in advance in another notebook (attached in appendix)

And we got the following optimized hyperparameters:
{'C': 0.1, 'epsilon': 10, 'gamma': 1, 'kernel': 'rbf'}
RMSE: 20.644052638806244

In [9]:
```
from sklearn import svm
from sklearn.model_selection import GridSearchCV

svr_clf = svm.SVR(C = 0.1, epsilon = 10, gamma = 1, kernel = 'rbf') # for svr, y is expected to have floating point val
svr_clf.fit(X_train_svr, y_train_svr)
```

Out[9]:

▾ SVR

SVR(C=0.1, epsilon=10, gamma=1)

In [10]:

```
df_test = pd.read_csv('/kaggle/input/petfinder-pawpularity-score/test.csv')
df_test = df_test.drop(columns = 'Id')

# extract independent variables
X_test_svr = df_test.iloc[:, 0:12]

# generate prediction
y_pred = svr_clf.predict(X_test_svr)
y_pred
```

Out[10]:

```
array([35.56403228, 35.89369024, 35.52965552, 35.4705902 , 35.51811519,
       35.5632086 , 35.52966413, 35.38661314])
```

Take mean of the 2 scores generated to get the final Pawpularity scores

In [13]:

```
score = (y_pred + Pawpularity)/2
score
```

Out[13]:

```
array([36.85285129, 37.01768026, 36.8356629 , 36.80613025, 36.82989273,
       36.85243943, 36.83566721, 36.76414171])
```

Now let's build the dataframe to save as a csv file

In [21]:

```
dic = {'Id': Id, 'Pawpularity': score}
result = pd.DataFrame(dic)
result.head(10)
```

Out[21]:

	Id	Pawpularity
0	c978013571258ed6d4637f6e8cc9d6a3	36.852851
1	4e429cead1848a298432a0acad014c9d	37.017680
2	43a2262d7738e3d420d453815151079e	36.835663

	Id	Pawpularity
3	8f49844c382931444e68dffbe20228f4	36.806130
4	4128bae22183829d2b5fea10effdb0c3	36.829893
5	80bc3ccafcc51b66303c2c263aa38486	36.852439
6	e0de453c1bffc20c22b072b34b54e50f	36.835667
7	b03f7041962238a7c9d6537e22f9b017	36.764142

pawpularity - Version 51

Succeeded (after deadline) · 7h ago · Notebook pawpularity | Version 51

20.51293 20.5241

The grade we get in the end, already much better than other models we've tried

In [22]:

```
# save the DataFrame to a CSV file for submission
result.to_csv('submission.csv', index=False)
```

This is the end of the notebook