Student Name: Bowen Li
Email: bli61001@usc.edu

Petfinder Pawpularity Score

Workflow for Predicting Petfinder Pawpularity Scores

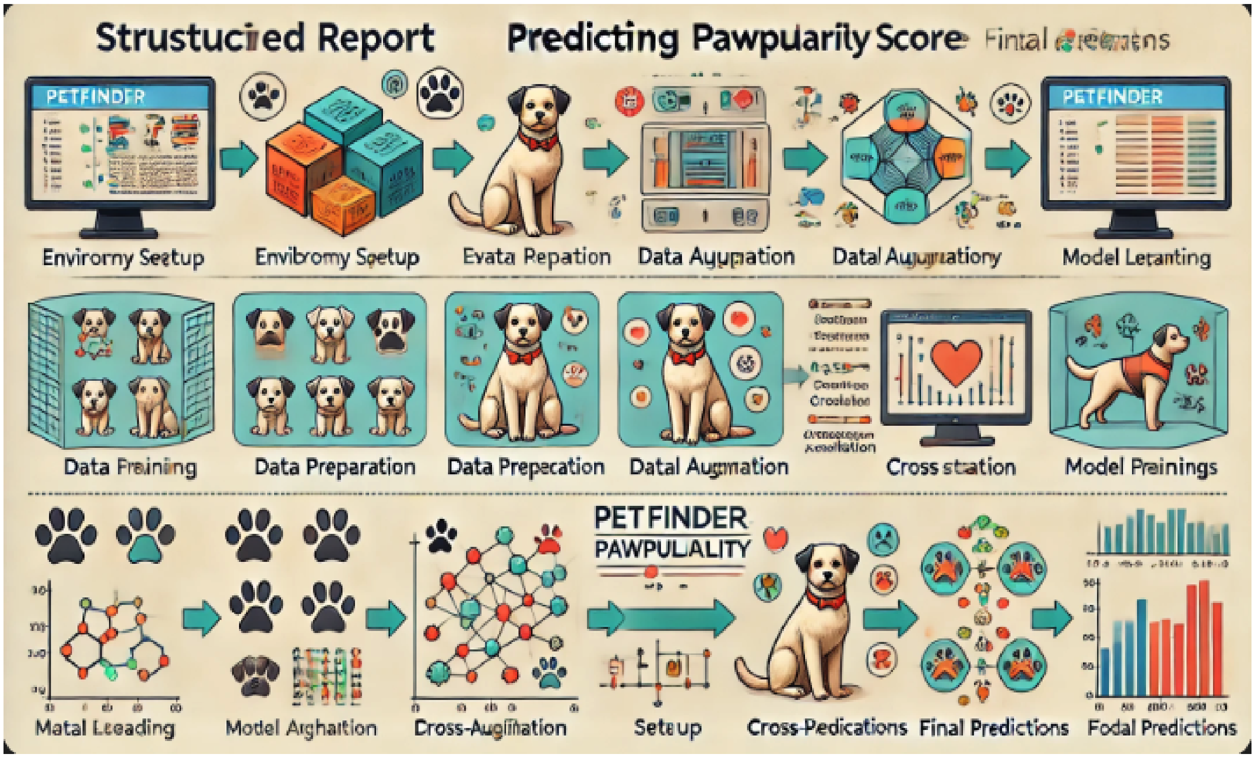

Figure 1: Workflow for Predicting Petfinder Pawpularity Scores

Environment Setup

This script sets up the necessary environment for training and running the machine learning models. It includes importing essential libraries, setting the system path for additional modules, ensuring reproducibility by setting random seeds, and preparing the directory for storing model checkpoints.

```python
# environment_setup.py
import sys
import os
import gc
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from pathlib import Path
from fastai.vision.all import *
from timm import create_model
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
import albumentations
from cuml.svm import SVR
import pickle
import tez
from tez.callbacks import EarlyStopping
from tqdm import tqdm
import math

sys.path.append(’../input/timm-pytorch-image-models/pytorch-image-models-master’)

myseed = 999
set_seed(myseed, reproducible=True)
torch.manual_seed(myseed)
torch.cuda.manual_seed(myseed)
torch.backends.cudnn.deterministic = True
torch.use_deterministic_algorithms(True)

if not os.path.exists(’/root/.cache/torch/hub/checkpoints/’):
    os.makedirs(’/root/.cache/torch/hub/checkpoints/’)
os.system("cp ’../input/swin-transformer/swin_large_patch4_window7_224_22kto1k.pth’ ’/root/.cache/torch/hub/checkpoints/swin_large_patch4_window7_224_22kto1k.pth’")
```

Data Preparation

This script handles the loading and initial processing of the dataset. It reads the training data, generates image paths, shuffles the dataset, and normalizes the target variable (Pawpularity score).

```
# data_preparation.py
from pathlib import Path
import pandas as pd

dataset_path = Path(’../input/petfinder-pawpularity-score/’)
train_df = pd.read_csv(dataset_path/’train.csv’)
train_df[’path’] = train_df[’Id’].map(lambda x: str(dataset_path/’train’/x) + ’.jpg’)
train_df = train_df.drop(columns=[’Id’])
train_df = train_df.sample(frac=1).reset_index(drop=True)  # Shuffle DataFrame
len_df = len(train_df)
print(f"There are {len_df} images")
train_df[’norm_score’] = train_df[’Pawpularity’] / 100
```

data augmentation cv

This script applies data augmentation techniques to enhance the dataset and sets up stratified k-fold cross-validation to ensure balanced training and validation splits.

```
# data_augmentation_cv.py
import albumentations
import numpy as np
from sklearn.model_selection import StratifiedKFold
import pandas as pd
import matplotlib.pyplot as plt

# Data Augmentation
test_aug = albumentations.Compose(
    [
        albumentations.Resize(384, 384, p=1),
        albumentations.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.05, rotate_limit=15, p=0.5),
        albumentations.RandomBrightnessContrast(p=0.5),
        albumentations.Cutout(num_holes=8, max_h_size=8, max_w_size=8, fill_value=0, p=0.5),
        albumentations.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225], p=1.0),
    ],
    p=1.0,
)

# Stratified K-Fold Cross-Validation
num_bins = int(np.floor(1 + (3.3) * (np.log2(len(train_df)))))
train_df[’bins’] = pd.cut(train_df[’norm_score’], bins=num_bins, labels=False)
train_df[’fold’] = -1

N_FOLDS = 5
strat_kfold = StratifiedKFold(n_splits=N_FOLDS, random_state=999, shuffle=True)
for i, (_, train_index) in enumerate(strat_kfold.split(train_df.index, train_df[’bins’])):
    train_df.iloc[train_index, -1] = i

train_df[’fold’] = train_df[’fold’].astype(’int’)
train_df.fold.value_counts().plot.bar()
plt.savefig(’data_distribution.png’)
train_df.to_csv(’df_train_fold_struggle.csv’)
```

model definition training

This script defines a custom neural network model based on the swin_large_patch4_window7_224 architecture and sets up the data loaders and learner for training.

```
# model_definition_training.py
import torch.nn as nn
from fastai.vision.all import *
from timm import create_model

class cust_fastai_model(nn.Module):
    def __init__(self, model_name=’swin_large_patch4_window7_224’, ifpretrained=True):
        super().__init__()
        self.swin = create_model(model_name, pretrained=ifpretrained, num_classes=0)
        self.custom_head = nn.Linear(in_features=1536, out_features=1, bias=True)

    def forward(self, image):
        emb = self.swin(image).squeeze(-1).squeeze(-1)
        out = self.custom_head(emb)
        return out

def petfinder_rmse(input, target):
    return 100 * torch.sqrt(F.mse_loss(F.sigmoid(input.flatten()), target))

def get_data(fold):
    train_df_f = train_df.copy()
    train_df_f[’is_valid’] = (train_df_f[’fold’] == fold)
    dls = ImageDataLoaders.from_df(train_df_f, valid_col=’is_valid’, seed=999, fn_col=’path’,
                                   label_col=’norm_score’, y_block=RegressionBlock, bs=32,
                                   num_workers=8, item_tfms=Resize(224),
                                   batch_tfms=setup_aug_tfms([Brightness(), Contrast(), Hue(), Saturation()]))
    return dls

def get_learner(fold_num, model_name=’swin_large_patch4_window7_224’, ifpretrained=True, ifcut=False):
    data = get_data(fold_num)
    if ifcut:
        model = cust_fastai_model(model_name, ifpretrained)
    else:
        model = create_model(model_name, pretrained=ifpretrained, num_classes=data.c)
    learn = Learner(data, model, loss_func=BCEWithLogitsLossFlat(), metrics=petfinder_rmse).to_fp16()
    return learn
```

model evaluation prediction

This script handles the evaluation of the model using cross-validation, generates predictions for the test set, and prepares the final submission file.

```
# model_evaluation_prediction.py
import gc
import torch
import numpy as np
from fastai.vision.all import *
from pathlib import Path

def test_cv(model_name, ifpretrained, image_size, model_path, n, beta, train_df, N_FOLDS=5, ifcut=False):
    all_preds = []
    train_df_f = train_df.copy()
    train_df_f[’pred’] = 1
    for i in range(N_FOLDS):
        learn = get_learner(fold_num=i, model_name=model_name, ifpretrained=ifpretrained, ifcut=ifcut)
        learn.model_dir = ’’
        learn.load(model_path + f’{i}.pkl’)
        dls = ImageDataLoaders.from_df(train_df, valid_pct=0.2, seed=999, fn_col=’path’,
                                       label_col=’norm_score’, y_block=RegressionBlock,
                                           bs=32,
                                       num_workers=8, item_tfms=Resize(image_size),
                                       batch_tfms=setup_aug_tfms([Brightness(), Contrast
                                           (), Hue(), Saturation(), RandomErasing(p=0.5,
                                           max_count=6)]))
        test_dl = dls.test_dl(train_df[train_df[’fold’] == i])
        preds, _ = learn.tta(dl=test_dl, n=n, beta=beta)
        preds = preds.view(preds.size(0),)
        print(f’Fold {i} results’, np.sqrt(((np.array(preds) - train_df[train_df[’fold’]
            == i][’norm_score’])**2).mean()))
        train_df_f.loc[train_df_f[’fold’] == i, ’pred’] = np.array(preds)
        del learn
        torch.cuda.empty_cache()
        gc.collect()
    print(np.sqrt(((train_df_f[’pred’] - train_df_f[’norm_score’]) ** 2).mean()))
    return train_df_f

def get_submit(model_name, ifpretrained, image_size, model_path, n, beta, N_FOLDS=5,
    ifcut=False):
    all_preds = []
    for i in range(N_FOLDS):
        print(f’Fold {i} results’)
        learn = get_learner(fold_num=i, model_name=model_name, ifpretrained=ifpretrained,
             ifcut=ifcut)
        learn.model_dir = ’’
        learn.load(model_path + f’{i}.pkl’)
        dls = ImageDataLoaders.from_df(train_df, valid_pct=0.2, seed=999, fn_col=’path’,
                                       label_col=’norm_score’, y_block=RegressionBlock,
                                           bs=32,
                                       num_workers=8, item_tfms=Resize(image_size),
                                       batch_tfms=setup_aug_tfms([Brightness(), Contrast
                                           (), Hue(), Saturation(), RandomErasing(p=0.5,
                                           max_count=6)]))
        test_dl = dls.test_dl(test_df)
        preds, _ = learn.tta(dl=test_dl, n=n, beta=beta)
        all_preds.append(preds)
        del learn
        torch.cuda.empty_cache()
        gc.collect()
    sample_df = pd.read_csv(dataset_path / ’sample_submission.csv’)
    preds = np.mean(np.stack(all_preds), axis=0)
    sample_df[’Pawpularity’] = preds * 100
    return sample_df

def main():
    sample_df1 = get_submit(’swin_large_patch4_window7_224’, False, 224, ’../input/swin-
        ting-model-embed-fastai/models/model_fold_’, 5, 0, 5, False)
    sample_svr1 = get_submit_svr(’swin_large_patch4_window7_224’, False, 224, ’../input/
        swin-ting-model-embed-fastai/models/model_fold_’, 1, 1, 5, False, svr_name=’../
        input/svrweight/svr_model_swin_tiny_224_vv1_’)
    sample_df2 = get_submit(’swin_large_patch4_window12_384_in22k’, False, 384, ’../input
        /pet-finder-new-model/swin_large_22k_v3_’, 1, 1, 5, False)

    weight = [0.7 * 0.8, 0.7 * 0.2, 0.15, 0.15]
    sample_df1[’Pawpularity’] = weight[0] * sample_df1[’Pawpularity’] + weight[1] * sample_svr1[’Pawpularity’] / 100 + weight[2] * sample_df2[’Pawpularity’] + weight[3] * df_test[’Pawpularity’]
    sample_df1.to_csv("submission.csv", index=False)
    sample_df1.head()

if __name__ == "__main__":
    main()
```