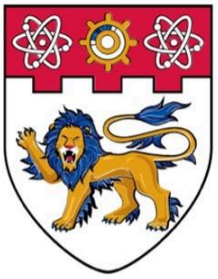

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4000/CZ4041 Machine Learning

PetFinder.my – Pawpularity Contest

Name	Matriculation Number	Contributions
Royce Teng	U2122063K	Everything
Oi Yeek Sheng	U2123653H	Everything
Teo Jia Yun	U2120392J	Everything

Table of Content

1. Introduction

A single photograph often determines a pet's future and a loving family. The PetFinder.my Pawpularity Contest on Kaggle was created to resolve this issue by utilising data figures to devise algorithms that can predict how appealing images of pets will be and, therefore, hopefully help with adoption strategies.

PetFinder.my is an animal welfare site in Malaysia that appreciates the role of data analytics in enhancing its adoption management system. Their existing "Cuteness Meter," while useful, is still in beta testing. This competition aims to improve this instrument by utilising a range of data analysis approaches to measure pet photos' appeal and provide recommendations on how the photos can be enhanced.

Our team participated in the contest with the goal of developing a model that would evaluate pet photos based on "Pawpularity" and be able to offer a popularity score. This report is intended to show how we carried out the assignment, prepared the data, chose the appropriate models, and creatively transformed the visual insights into something useful.

Next, we will discuss the methodology we developed and the decisions we made in constructing our model. We will also evaluate the results we have achieved, considering both the positive and the negative aspects.

1.1. Problem Statement

This paper is concerned with the problem presented by the PetFinder.my Contest available on Kaggle. The primary objective is to build a predictive machine learning model that will be able to efficiently and accurately forecast the pawpularity score. This score indicates the degree of how appealing the photograph of a pet offered for adoption on online platforms is.

In particular, the problem consists of:

1. Assessing the original pet images and the related metadata
2. Developing a model that will estimate the picture's popularity.
3. Making precise predictions that could help improve the quality of pet pictures.

2. Data Exploration and Analysis

2.1. Overview of Dataset

The dataset provided for this competition comprises 9,912 entries, each representing the metadata and characteristics associated with an image. Our final goal is to explore various factors that may influence the popularity of images, termed “Pawpularity” in the dataset. The dataset includes the following columns:

- Id: filename for the image
- Subject Focus: The pet stands out against an uncluttered background, not too close / far.
- Eyes: Both eyes face front, with at least one eye clear.
- Face: Denotes if the face is visible
- Near: Denotes that the pet fills a major portion of the image
- Action: Represents the pet is in the middle of an action
- Accessory: Denotes any physical or digital item accompanying the pet
- Group: Refers to two or more pets being present
- Collage: Identifies digitally edited images
- Human: Indicates the presence of a person
- Occlusion: Points out undesirable objects partially obscuring the pet
- Info: Represents added text or labels
- Blur: Describes if the image is out of focus

	Id	Subject Focus	Eyes	Face	Near	Action	Accessory	Group	Collage	Human	Occlusion	Info	Blur	Pawpularity
0	0007de18844b0dbbb5e1f607da0606e0	0	1	1	1	0	0	1	0	0	0	0	0	63
1	0009c66b9439883ba2750fb825e1d7db	0	1	1	0	0	0	0	0	0	0	0	0	42
2	0013fd999caf9a3efe1352ca1b0d937e	0	1	1	1	0	0	0	0	1	1	0	0	28
3	0018df346ac9c1d8413cfcc888ca8246	0	1	1	1	0	0	0	0	0	0	0	0	15
4	001dc955e10590d3ca4673f034feeef2	0	0	0	1	0	0	1	0	0	0	0	0	72
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9907	ffbfa0383c34dc513c95560d6e1fdb57	0	0	0	1	0	0	0	0	0	0	0	1	15
9908	ffcc8532d76436fc79e50eb2e5238e45	0	1	1	1	0	0	0	0	0	0	0	0	70
9909	ffdf2e8673a1da6fb80342fa3b119a20	0	1	1	1	0	0	0	0	1	1	0	0	20
9910	fff19e2ce11718548fa1c5d039a5192a	0	1	1	1	0	0	0	0	1	0	0	0	20
9911	fff8e47c766799c9e12f3cb3d66ad228	0	1	1	1	0	0	0	0	0	0	0	0	30

9912 rows × 14 columns

Figure 1: Dataset provided by Kaggle for the 'Pawpularity’ Contest

2.2 Data Cleaning

During the dataset analysis, we noticed that the problem of duplicate images with different ‘Pawpularity’ ratings exists. To solve this problem, we used image hashing to delete the duplicated images. The procedures followed include:

1. Image Hashing:

- We used the ‘image hash’ library to create an average hash for each image picture. This is useful for establishing similarities between specified images regarding their shapes and visual characteristics despite having different image titles.

2. Detection of Duplications:
    - The images with similar hashes were clustered to form the groups.
3. Verification and Visualization
    - To check for potential duplicates, we provided some of the detected duplicated images with their respective “Pawpularity” scores. This was useful for visually ascertaining whether the images were, in fact, duplicates.

Figure 2: Examples of duplicate images found in the dataset

4. Removal of Duplicates
    - All but the first occurrence were marked for deletion for each set of duplicates. Then, the original dataset was updated by removing entries associated with duplicate images, gaining a cleaned dataset without redundancy.

By implementing these steps, 27 images were discarded, and the quality of the overall dataset was enhanced, raising the validity of any future analysis or modelling based on this data.

2.3 Data Visualisation

The histogram gave an insight into the “Pawpularity’ score values in the available data. Here’s a brief overview:

- Shape
  - The right skew of the `Pawpularity' score histogram indicates that it is peaking between the two values, 20 and 40.
- Spread of Data:
  - The mean “Pawpularity” score is about 38.
  - The standard deviation is about 20.6.

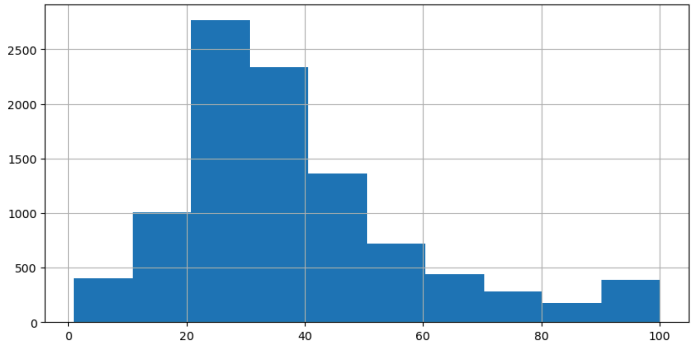

Figure 3: Distribution of 'Pawpularity' Scores

The analysis of the 12 binary features with respect to the Pawpularity scores unveils the fact that these features are not likely to be useful in making predictions in a machine-learning model. Here are the reasons:

1. Similar distributions:

   The box plots and histograms for each feature for the two categories (0 and 1) are very similar. This indicates that there’s little differentiation between the categories in terms of Pawpularity scores.

2. Consistent medians:

   The median lines in the box plots are at similar levels for both categories across all features. This indicates that the presence or absence of these features doesn’t consistently correspond to higher or lower Pawpularity scores.

3. No standout features:

   None of the 12 features shown appear to have a significantly different distribution that would make it a strong predictor.

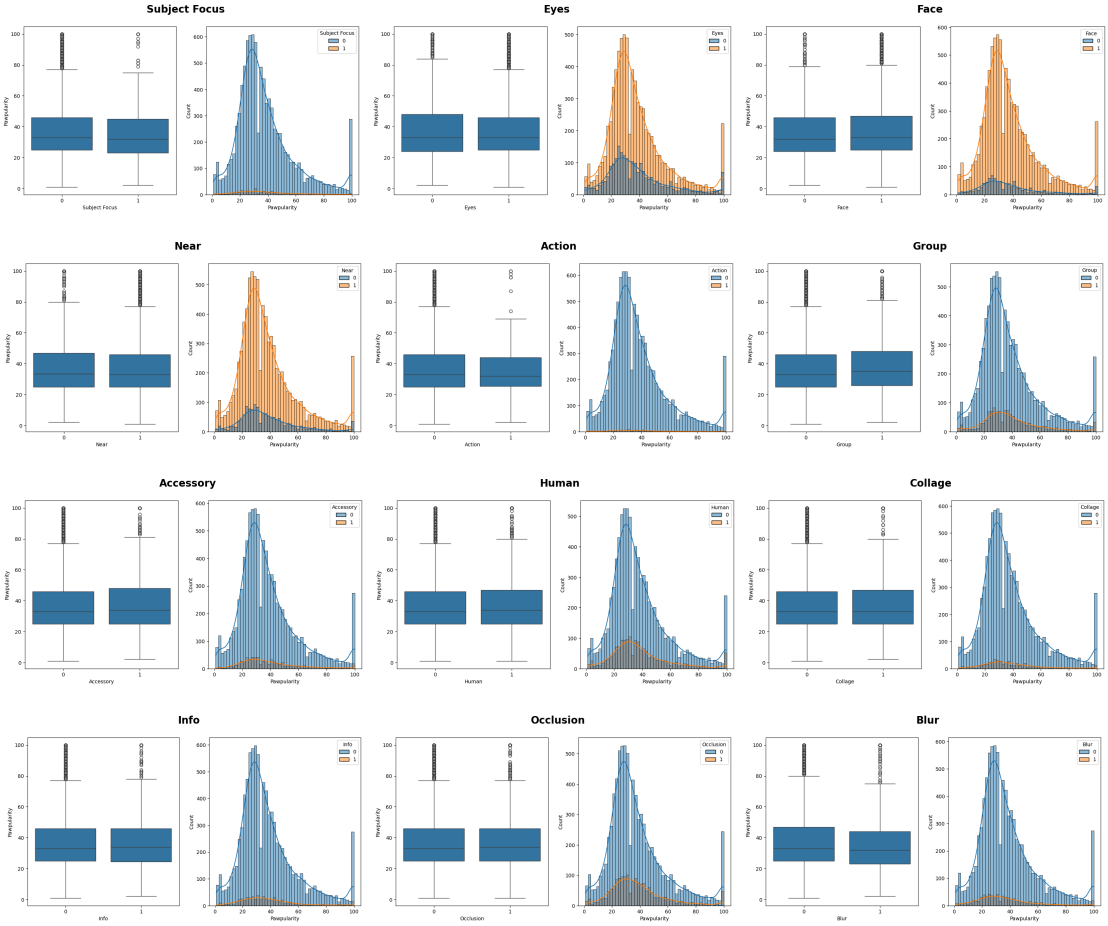

Figure 4: Histograms and Box Plots of 12 Binary Features in the Dataset

3. Methodology

3.1 Model Architecture

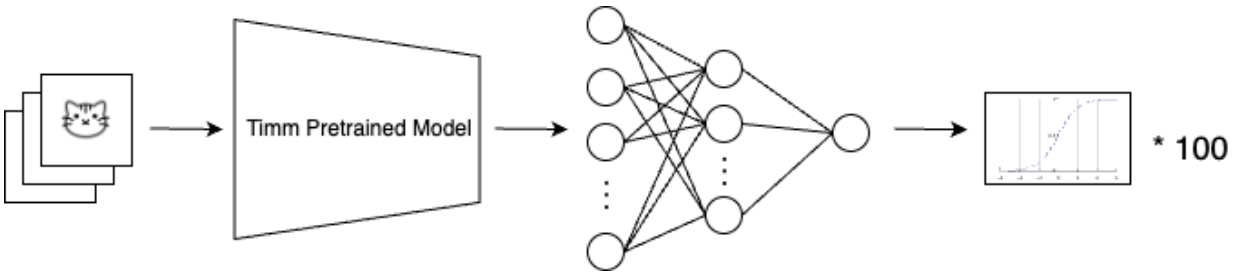

Fig 5: Architecture of Model

Our model design utilises the Timm library (Pytorch Image Models). Timm is a deep-learning library that provides a range of state-of-the-art computer vision models and resources to aid the training process.

We use models from the Timm library to identify features in the image. Our process starts by feeding a batch of 3-channel images into the Timm model. These images undergo processing to capture embeddings that accurately portray the image characteristics. The acquired embeddings are then passed through two linear layers that gradually decrease the dimensions to generate a numerical output. The output then uses sigmoid activation and is scaled by 100 to derive a Pawpularity score between 0 and 100. Figure 5 shows the architecture of our model.

```
class PetNet(nn.Module):
    def __init__(self, model_name, out_features,inp_channels, pretrained):
        super().__init__()
        self.model = timm.create_model(model_name, pretrained=pretrained, in_chans=3, num_classes=0)
        self.fc1 = nn.Linear(self.model.num_features, 128)
        self.dropout = nn.Dropout(0.1)
        self.fc2 = nn.Linear(128, 1)

    def get_image_embedding(self, image):
        return self.model(image)

    def forward(self, image):
        output = self.model(image)
        x= self.fc1(output)
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

Fig 6: Implementation of PetNet Class in Pytorch

3.2 Evaluation Metric

Our project utilised two evaluation metrics, one for training loss and another for test loss.

Training Loss

During training, we used Binary Cross Entropy with Logits Loss (BCEWithLogitsLoss) as our training loss. This loss function is particularly well-suited for binary classification tasks and is applied directly to the raw model outputs (logits). This ensures compatibility with the model's architecture while still providing a robust measure for optimising our prediction through backpropagation.

The calculation of BCEWithLogitsLoss for a single data point is given by the following:

$$BCEWithLogitsLoss = -[y \cdot log(\sigma(x)) + (1 - y) \cdot log(1 - \sigma(x))]$$

Where:

- y : Ground truth label (0 or 1).
- x : Model's raw output (logit).
- $\sigma(x)$: Sigmoid function applied to x, defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Test Loss

Our model effectiveness is assessed using the root mean square error (RMSE) indicated in the competition guidelines. To determine the loss incurred by our model performance evaluation process involves utilising the RMSE calculation formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

Where:

- $\hat{y}_i$: Model's predicted value
- y_i : Ground truth value
- n : Number of samples

This metric quantifies the deviation between the model's predictions and the actual values, providing a comprehensive measure of error. By minimising RMSE, we aim to enhance the precision of our model, ensuring its prediction closely aligns with the data points.

3.3 5-Fold Cross Validation Model Training

In this project, we use 5-fold Cross-Validation to train and validate models. The dataset is divided into 5 subsets, labelled Fold1, Fold2, ..., Fold5. Using these subsets, we train 5 models of the same architecture, each time using a unique fold as the validation set and the remaining 4 as the training set. For example, Model 1 is trained using Folds 2–5 data and validated on Fold 1. This process is repeated until each fold has served as the validation set exactly once. Figure 7 illustrates the 5-fold Cross-Validation approach to model training.

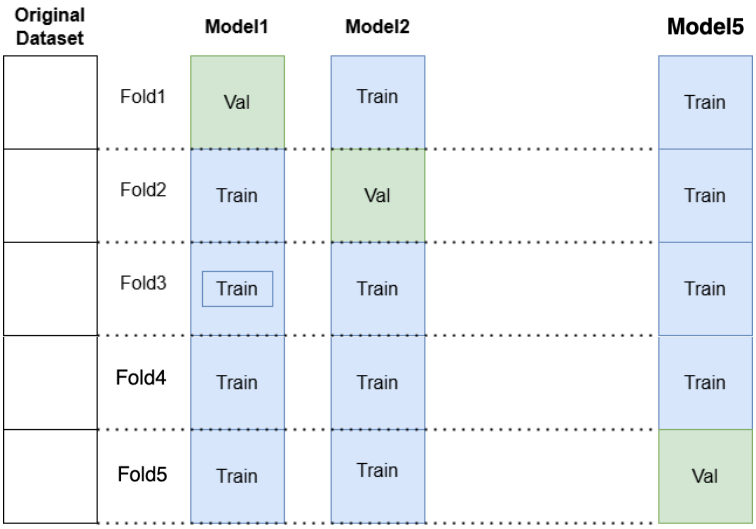

Fig 7: Model training using 5-fold Cross-Validation

The folds are generated using **Stratified K-Fold Cross-Validation**, which ensures that each fold represents the overall dataset distribution. Figure 8 shows the data distribution of training images across each fold.

kfold	mean	std	count
0	37.981846	20.593212	1983
1	38.056984	20.565744	1983
2	38.069122	20.641669	1982
3	38.031786	20.570571	1982
4	38.055499	20.609343	1982

Fig 8: Data distribution of training images across each fold

After training all 5 folds, we obtain predictions for each training image across these models. Collectively, these predictions are referred to as **Out-of-Fold (OOF) predictions**. By computing the RMSE of each fold and averaging them, we can get the overall performance of the model architecture. This approach resembles ensemble learning, which helps to improve the model's robustness and accuracy.

Furthermore, since each model is trained on only 80% of the data, this method helps prevent overfitting on the complete training dataset while improving the model's generalisation ability on unseen test data. The models are also trained on random subsets, which reduces the risk of overfitting on any particular fold, thus enhancing the model's capacity to perform well on the test set.

3.4 Ensemble Learning

Ensemble Learning is a powerful machine learning technique that combines multiple models' predictions to create a more robust final prediction. Integrating different machine learning models reduces the impact of individual model errors and enhances generalisation.

```
models = {
    'beit_large_patch16_512': {
        'model_path': '/kaggle/input/beit_large__512_5fold/transformers/default/1',
        'im_size': 512,
        'pred':[],
        'weight':0.4
    },
    'swin_large_patch4_window7_224': {
        'model_path': '/kaggle/input/swin_5folds/transformers/default/1',
        'im_size':224,
        'pred':[],
        'weight': 0.4
    },
    'vit_large_r50_s32_384': {
        'model_path': '/kaggle/input/vit-large-5-folds/pytorch/default/1',
        'im_size': 384,
        'pred':[],
        'weight': 0.2
    },
}
```

Fig 9: Models used in Ensemble Learning

3.5 Forward Selection

Forward selection is a type of feature selection technique used primarily in statistical modelling and machine learning. The common use case is to select a subset of relevant features for model construction, improving model performance by reducing overfitting and enhancing generalisation.

Process

Forward selection begins with an empty model containing no predictor variables. The process is as follows:

1. Initial Feature Evaluation:
    - Each feature is added individually to the model, and the model's performance is evaluated using the selected evaluation metric.
    - The feature that has the best evaluation metric score is selected.
2. Iterative (Stepwise) Selection:
    - Additional features are added in a stepwise fashion, where each iteration the combination of features will be evaluated to identify the best subset.
    - This process is repeated iteratively until no new feature significantly improves the model's performance or until a predefined number of features is reached.

The forward selection algorithm is suitable for this Kaggle competition as pre-trained models provide good and reasonable estimation. The idea of adopting forward selection is to stack embeddings extracted from pre-trained models, which would provide a better estimation fitting in SVR. At the same time, this approach also helps us identify the best set of pre-trained models for this project.

The advantages of forward selection lie in its simplicity, which allows for easy understanding and implementation. At the same time, this approach is less computationally intensive compared to exhaustive search methods. However, forward selection still poses some limitations. The most prominent is the risk of overfitting, as the model may not generalise well to unseen data without proper validation.

3.6 Support Vector Regression (SVR)

Support Vector Regression (SVR) is a machine learning technique developed based on Support Vector Machine (SVM). The main function of SVR is to perform regression by locating hyperplanes in high-dimensional space that best fit the data while providing a wider margin of error from the hyperplane to improve the accuracy of predictions.

In this project, SVR is employed by training models corresponding to each image embedding taken from each of the trained fold models. This introduces diversity into the ensemble, combining two distinct learning tasks. The transformer model head uses classification loss as a

learning criterion for improving predictions, while the SVR head uses regression loss as the objective.

This combination enables the ensemble to capture a broader range of patterns in the data, ultimately improving predictive performance. Figure 10 provides an overview of the architecture of SVR and Transformers heads.

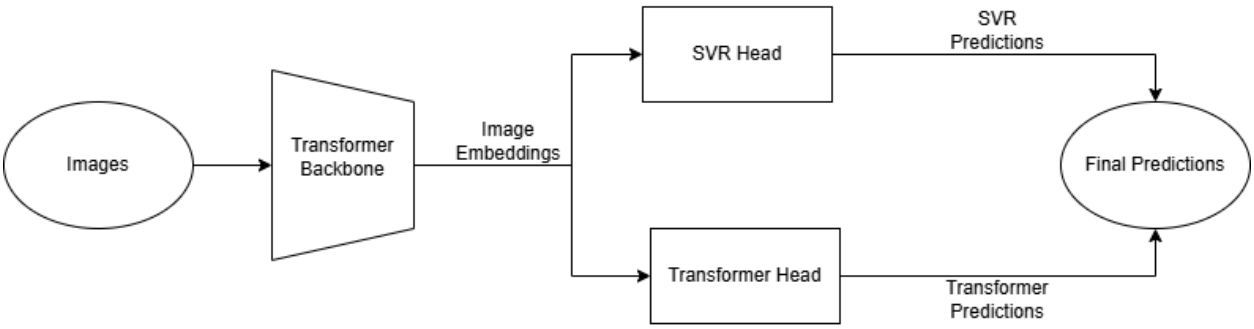

Fig 10: Architecture of SVR

4. Experimental Studies

4.1 Forward Selection Results

In the forward selection experiment, we explored a search space comprising some of the most popular and widely recognised pre-trained transformer models available in the Timm library. From an initial pool of 10 shortlisted models, the forward selection process was applied iteratively to identify the best combination of models that optimised performance.

The best-performing combination was achieved with 5 models. Adding additional models beyond this point not only failed to improve performance but also led to a slight decline. This was likely due to the introduction of redundant features that impacted model generalisation. Figure 11 shows the validation loss when adding models through forward selection. Refer to Table 1 for the performance of the 10 pretrained models.

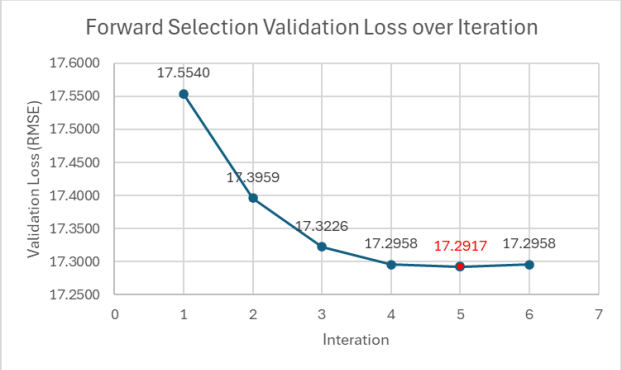

Fig 11: Validation Loss when adding models through Forward Selection

PERFORMANCE OF INDIVIDUAL PRETRAINED MODEL

Pretrained Model	Validation Loss
beit_large_patch16_512	17.61049
deit_base_distilled_patch16_384	17.65580
maxvit_xlarge_tf_512	17.71198
tf_efficientnet_tf_512	**17.55403**
vit_huge_patch14_clip_336	17.72994
swin_large_patch4_window7_224	**17.55403**
swin_large_patch4_window12_384	17.79772
resnet50x4_clip.openai	17.85158
resnet101_clip.openai	17.95928
resnext101_32x8d	18.27009

Table 1: Performance of Individual Pretrained Model

4.2 SVR vs Linear Layer

We found that when we fit the image embeddings that were extracted from the Swin transformer and fit into SVR, the prediction score was much worse at 20.74163. If we were to use the linear layer to make predictions, we would be able to achieve a much better score of 17.94179. Hence, we decided to make use of the linear layers for all the models to make predictions. Figure 12 shows the prediction score between the Swin transformer and the Swin transformer with SVR attached to it.

Fig 12: Prediction score between Swin and SVR

4.3 Weighted Voting

To further enhance the model's performance, we used **weighted voting**. Instead of relying on a single model, predictions from all trained fold models are aggregated, and a weighted voting strategy is applied to determine the final output. This method introduces diversity into the final prediction process as it leverages the strengths of different model architectures.

The formula for calculating the weighted sum of the predictions from individual models is as follows:

$$\widehat{y} = \sum_{i=1}^{n} w_i \cdot \widehat{y_i}$$

Where:

$\widehat{y}$: Final ensemble prediction

n : Total number of models in ensemble

w_i : Weight assigned to the i^{th} model. These weights represent the relative importance of each model, and the sum amounts to 1

$\widehat{y_i}$: Prediction of the i^{th} model

Model Beit achieved the best score, so it was given the most weight. Swin and Vit, which performed worse than Beit, will be given lesser weights. Figure 13 shows the prediction scores of individual models.

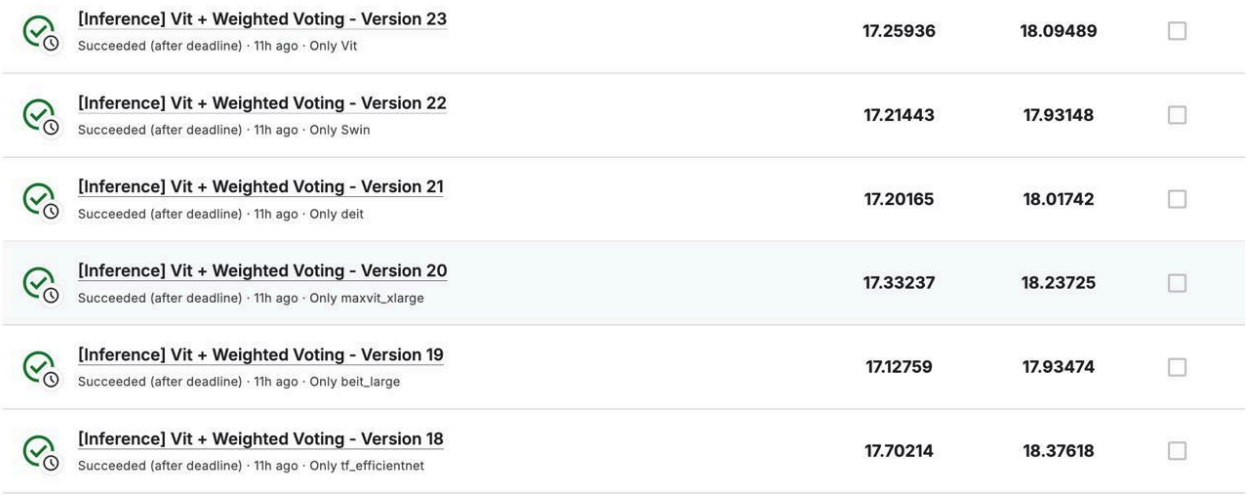

Fig 13: Prediction score of Individual Models

Through trial and error, we achieved the best score using

$$(0.5 * Beit) + (0.15 * Vit) + (0.35 * Swin)$$

5. Novel Solutions

In this Kaggle competition, we used a strategy that focuses on employing a linear layer in a more novel way. After conducting explorations and trials, we discovered a technique that provided superior outcomes than other commonly incorporated practices.

Key Components of Our Solution:

1. Enhanced Architecture with Two Linear Layers
    - Before fine-tuning, we incorporated two new linear layers into the pretrained model. The first layer performed dimensional reduction to maintain the image embeddings' most prominent features while getting rid of irrelevant information or noise. Then, the second layer was used to make predictions about the output. This design also took advantage of the neural network's capacity to model complex and nonlinear relationships, which was more advantageous than traditional machine learning methods such as SVR.
2. BCEWithLogitsLoss as training loss
    - The linear layer's result passes through a sigmoid function. As a sigmoid transforms a number in the range of 0 to 1, this can ensure our model output will always be mapped to 0 to 100 after multiplying by 100.
    - We chose to use BCEWithLogitsLoss instead of RMSE for training loss. RMSE treats all deviations linearly, so it is less sensitive to errors, even those near its target. On the other hand, BCEWithLogitsLoss penalises outputs heavily when they deviate significantly from the target and less when they are near the target, making the model more sensitive to errors. This helps the model better learn to predict the regression score.
3. Weighted Voting
    - We implemented a weighted voting mechanism to obtain the maximum strength of the different models.
    - The models selected by forward selection are evaluated by submitting to the Kaggle platform using their private dataset.
    - Based on these evaluations, we assigned specific weights to each model, reflecting their relative importance and predictive power.
    - This approach allows us to utilise different models' diverse strengths while mitigating their weaknesses.

6. Results

Using 0.50*Beit + 0.15*Vit + 0.35*Swin as the weights for weighted voting, our Ensemble model achieved 17.80479 RMSE on the public leaderboard. This result slots between 186th and 187th place, which is 187/3538=0.0529, top 5.29%. Figure 14 shows the leaderboard results for our ensemble mode.

Fig 14: Leaderboard Results for Ensemble model

7. Conclusion

In this project, we explored a series of strategies to enhance the predictive performance of the Kaggle competition. Using state-of-the-art pre-trained transformer models, forward selection, 5-fold cross-validation training, weighted voting, and fine-tuned linear layers allowed us to reach a respectable public RMSE score of 17.80479. We were then placed in the uppermost 5.29% of the public leaderboard.

This competition challenged us to think creatively and experiment with various approaches. Not all the tips and techniques turned out to be effective in improving the model, but some others helped significantly improve the performance of our model. The key insights we gained from these experiments include:

Feature Selection and Model Generalization: Through forward selection, we learned to identify subsets of models that enhance generalisation while avoiding redundancy, thus optimising our results.

Linear Layers vs. SVR: This project demonstrated that fine-tuned Linear Layers have the superior capability of neural networks in capturing complex and nonlinear relationships compared to traditional machine learning techniques.

Ensemble Learning and Weighted Voting: This project also taught us how to amplify the strengths of individual models while mitigating their weaknesses using ensemble learning and weighted voting

To conclude, this project allowed us to explore and understand machine learning and deep learning techniques used for tasks such as computer vision regression. The experience and knowledge acquired from this project will be invaluable for future endeavours in machine learning.

References

1. Bao, H., Dong, L., Piao, S., & Wei, F. (2021). Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*.
2. Dosovitskiy, A. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.
3. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 10012-10022).
4. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
5. Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1492-1500).
6. Tu, Z., Talebi, H., Zhang, H., Yang, F., Milanfar, P., Bovik, A., & Li, Y. (2022, October). Maxvit: Multi-axis vision transformer. In *European conference on computer vision* (pp. 459-479). Cham: Springer Nature Switzerland.
7. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2021, July). Training data-efficient image transformers & distillation through attention. In *International conference on machine learning* (pp. 10347-10357). PMLR.
8. Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105-6114). PMLR.