COMP90086 Computer Vision Final Project Report

Man-Hua Chu
1403798

Pei-Hsuan Hsu
1424430

I. Introduction

A key application of computer vision is finding the physical relationships between objects in the real world. For example, self-driving cars depends on visual data to gauge the location and trajectory of nearby objects to navigate. In this project, we aim to develop an algorithm to predict the stability of a stack of blocks from a single image.

The given dataset is a subset of the ShapeStacks [1]. Each image shows a vertical stack of blocks, composited by 2-6 blocks with diverse colours and shapes. Our goal is to predict the Stable Height of the stack, namely the number of blocks that will not fall down from the top. In the following sections, we will demonstrate how to fine tune a pre-trained model to solve this task.

II. Design Choices

We have explored several pre-trained Convolution Neural Networks (CNN) from lightweights models to deep residual networks for the task. The following paragraphs illustrate the model comparison and the justifications of our final choice.

Most models we tuned are pre-trained on the ILSVRC 2012 [2] dataset, a subset of ImageNet, known as ImageNet-1k, containing 1000 object classes with 1,281,167 training images, 50,000 validation images and 100,000 test images[1].

Given the limited computation resources, we started from simple and efficient models, such as the MobileNets, which can be applied in diverse circumstances and perform well on tasks like real-time image classification and object detection on mobile devices. We tried *MobileNetV2* [3] from Keras while the result shows that it overfitted the training set, despite of applying techniques such as early stopping, dropout, and class weights. Then we moved on to more advanced MobileNetV3 [4]. The *Tf_mobilenetv3_large_minimuml_100.in1k* and and *Mobilenetv3_large_100.ra_in1k* model from HuggingFace ended up having an accuracy of 40% and 45%, respectively. They performed only marginally better than the baseline that has an accuracy of 40%, so we shifted to heavier-weights CNNs that may be more suitable for this task.

Next, we trained deeper neural networks such as *ResNet18*, *ResNet50*, and *BiT* [5] models. They have over 10 millions of parameters (see Table. I), leading to relatively time-consuming training and a higher complexity. For the *ResNet*, its residual connections mitigate the vanishing gradient problem in deeper networks. After fine-tuning, *ResNet* [6] demonstrated that deeper networks perform better with the advantage of more layers capturing more features, with accuracies of 47% on *ResNet18* and 50% on *ResNet50*. For the *BiT* model, we did not find suitable hyperparameters and it suffered from overfitting after we tested for several trials. Because *BiT* is a transfer learning model, it requires more training and tuning time, we ended up choosing alternative models.

Considering the trade-off between efficiency and accuracy, we employed *ResNet50*, which has deeper but simpler architecture. *ResNet50*'s architecture is elaborated below.

TABLE I
Model Design Comparison

Model	No. of Layers	No. of Parameters	Size of Weights
MobileNetV2	53	3,504,872	13.4MB
Tf_mobilenetv3_large_minimuml_100.in1k	41	3,504,872	15.8MB
Mobilenetv3_large_100.ra_in1k	54	5,483,032	22.1MB
timm/resnet18.a1_in1k	18	11,689,512	46.8MB
timm/resnet50.a1_in1k	50	25,557,032	102.0MB
BiT	50	23,510,400	89.57MB

III. Methodology

Our objective is to predict the Stable Height in each scene. We frame the problem as a classification task, standardising the training workflow by treating images as the input, and Stable Height, labelled from 0 to 5, as the output. The dataset was split into training and validation sets at the ratio of 8:2 with stratification. We will delve into the architecture of *ResNet50* and the fine-tuning methods.

A. Model Architecture

The selected model, *ResNet50*, has been one of the most popular and widely used architectures in the *ResNet* family. By improving the VGGNet model, *ResNet* introduces the concept of residual blocks, which uses fewer filters and consequently reduce the model complexity without sacrificing performance.

ResNet50 uses bottleneck residual blocks with 3 layers instead of 2 (Fig. 1) with identity shortcut, which makes it keep the efficiency as building blocks (Fig. 1 left) and remain similar time complexity. The bottleneck block is constructed in the order of a 1×1 layer for reducing the number of channels,

[1]Refer to https://huggingface.co/datasets/ILSVRC/imagenet-1k

then a 3×3 layer to extract features, and ends with a 1×1 layer expanding the original channel size.

The structure of *ResNet50* is initialised by 7×7 convolution with 64 filters and stride of 2, followed by a max-pooling layer to reduce dimensions. Next, it is composed by residual blocks using bottleneck architecture with four different filter sizes. Then, there is a global average pooling layer to prevent overfitting by reducing the feature map size, and a final layer producing classification predictions.

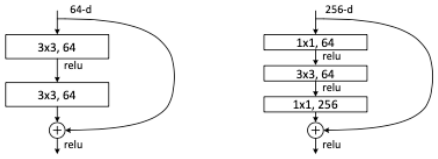

Fig. 1. Bottleneck Architecture[2]

ResNet50 is a powerful CNN model for image classification which solves the problem of vanishing gradient and allows deeper networks while keeping a relative high accuracy.

B. Experiment Steps

Step 1 Modifying the Last Layer The pre-trained *ResNet50* is designed for classification tasks and outputs 1000 classes. Our target Stable Height, has six possible classes, hence we modify the final fully connected layer to output six logits.

Step 2 Unfreeze Layers *ResNet50* has already learned features such as edges and shapes in its early layers. We aim to reuse these general features by freezing the parameters and not updating them in backpropogation. Freezing layers also reduces computational cost and prevents overfitting. The decision on how many layers to unfreeze depends on the similarity between the ShapeStacks data and the ImageNet dataset. The more similar they are, the more layers can be frozen to retain their learned parameters. Initially, we froze all layers, but both train and validation accuracy stagnated at around 23%. This likely stemmed from the differences between the datasets: ImageNet is designed for object classification, while the ShapeStacks subset only contains different 3D shapes. Therefore, we unfroze more later layers, which capture abstract and task-specific features and found unfreezing the last two layers gave the superior results.

Step 3 Add Linear Layers We extended the last fully connected layer by adding linear layers, each followed by a ReLU activation function to introduce non-linearity. This modification helps the model learn more complex patterns. We settled on adding two linear layers that boosted the performance without increasing the risk of overfitting.

Step 4 Batch Size Batch size plays a critical role in determining computational power [7]. Smaller batch sizes require less memory and can lead to noisier gradients, which may help escape local minimum but can also introduce instability. In contrast, larger batch sizes typically speeds up training but possibly causes the model to get stuck in local minimum. According to [7], a batch size of 32 is a recommended starting point. We gradually increased the batch size up to 128, which frequently terminated training due to memory constraints but indeed led to faster convergence. After multiple experiments, a batch size of 64 provided the best balance, offering both stable and effective performance.

Step 5 Learning Rate We employed the Adam optimiser, which is commonly chosen for its ability to adaptively adjust the learning rate during training. The objective function is cross-entropy loss given that this is a classification task. To determine the initial learning rate, we referred to the approach outlined by [8], which suggests gradually increasing a small learning rate linearly throughout the early iterations. Due to the limited computation power, we could only test a limited set of values ranging from 1e-4 to 5e-2. Among these values, 1e-2 provided the most promising accuracy. The model exhibited overfitting, while the validation accuracy continued to increase during the first 20 epochs. Therefore, we considered this learning rate acceptable but recognised the need for addressing overfitting.

Step 6 Dropout Dropout can mitigate overfitting by setting a fraction of neurons to zero randomly and scaling the remaining active neurons by a factor of $1/(1-p)$, where p is the dropout probability. After several trials, we found that a rate of 0.3 yielded the best results. However, overfitting persisted, and the validation accuracy plateaued at around 50%.

Step 7 Data Augmentation To enhance the model's ability to generalise and alleviate overfitting, data augmentation techniques are introduced. It is beneficial for training neural networks if we safely assume that we can obtain more information from the existing dataset by modifying images to increase the diversity of the training data [9]. Common tricks include flipping, cropping, and colour space transformation. Below are the operations we applied with PyTorch[3]:

- `TrivialAugmentWide` Introduced by [10], this method utilises a single random transformation from a predefined list of augmentations with random strength. It is parameter-free, so no tuning is required, making it an easy-to-use approach, especially when there is limited experience with image processing. We specified black as the fill-in colour.
- `RandomResizedCropAndInterpolation` Crop the image to a random size (between 8% and 100% of the original size) and a random aspect ratio (between 3/4 and 4/3). The image is then resized to the target size of the model using random interpolation.
- `RandomHorizontalFlip` This method randomly flips the image horizontally with a probability of 0.5.
- `ColorJitter` Randomly alters the brightness, contrast, saturation, and hue of images, providing diversity in

[2]Fig. 1 is originated from [6].

[3]Note that data augmentation was only applied on the training dataset, not on the validation and test sets.

colour properties.

- `Normalize` We normalised the image pixel values to a uniform range. This avoids pixel saturation and ensures consistent input values, improving performance.
  Due to data augmentation, the validation accuracy grew to about 60% and there is no sign of overfitting. Whereas, there is a significant gap between the training and validation accuracy and the reasons are discussed later.

Step 8 Adjusting Class Weights Class imbalance exists in our dataset (see Fig. 2). In order to give more importance to minority classes during training, we computed class weights inversely proportional to the class frequencies by:

$$\text{Weights} = \frac{\text{Total Number of Samples in the Dataset}}{\text{Total Number of Classes} \times \text{Number of Samples in Class}}$$

The class weights are shown in Table. II. For instance, class 1 and 2 has a lower weight since there are more instances with Stable Height 1 and 2 in the dataset.

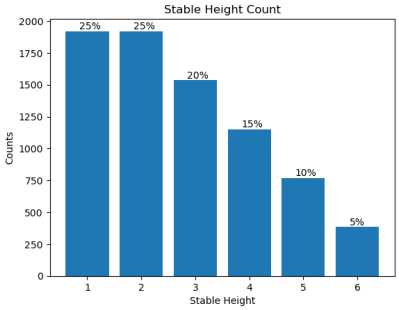

Fig. 2. Stable Height Counts

TABLE II
CLASS WEIGHTS ON STABLE HEIGHT

Stable Height	1	2	3	4	5	6
Weights	0.6667	0.6667	0.8331	1.1106	1.667	3.335

However, the accuracy of the validation set dropped to roughly 56% with adjusting class weights. We are still investigating the cause of this decline while reviewing papers with similar result. [11] suggests that the impact of importance weighting in deep learning varies based on factors such as early stopping, regularisation, and batch normalisation. They visualised decision boundaries during training and classification ratios over time, concluding that the weighting initially influences classification ratios, but the effects may diminish after sufficient epochs.

IV. RESULT

In this section, we will present and analyse the performance of our final model both visually and numerically.

First, we examine the accuracy and loss curves for both the training and validation datasets. As shown in Fig. 3, the validation losses decrease and remain lower than the training losses throughout the entire training. The accuracy curves for the training and validation datasets, both exhibit an upward trend, indicating an improvement in the model's accuracy. This suggests that the model doesn't overfit and consistently captures the key features from the input images throughout the training process. As a result, our fine-tuned model has an accuracy of 64.78% on validation dataset, and the accuracy on unseen test set in the Kaggle Competition is 64.06%.

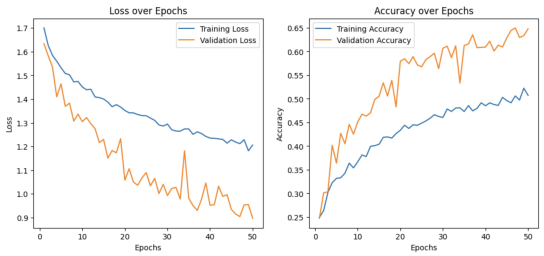

Fig. 3. Training History Curves

When training the classification model on the imbalanced dataset, accuracy can be misleading indicator to judge the performance as the model may achieve higher accuracy by simply predicting instances as the majority class. As a result, it is essential to examine the confusion matrix to for a more comprehensive understanding of the model's effectiveness.

The confusion matrix provides a summary of the outcomes of each class (Fig. 4), with corresponding metrics shown in Tables. III. The results indicate that classes with more data tend to have higher **Precision**, **Recall** and **F1-Score**.

Precision represents model's ability to correctly predict positive cases. We can see that the **Precision** of labels, Stable Height 1 and 2, are over 70%, while for the Stable Height 6 class, it drops below 40%. This indicates that the model is less effective at avoiding false positive for the Stable Height 6 class. The average **Precision** across all classes is 0.6060, suggesting the model has a moderate performance and there is room for improvement.

Recall measures how effectively a model identifies all relevant instances within a dataset. The **Recall** of classes of Stable Height 1 to 5 are higher than Sable Height 6, and the average **Recall** is 0.6159, meaning that the model can identify more than half instances correctly. It can be observed from the Table. III that the **Recall** score is higher than **Precision** score in classes of Stable Height 3, 5 and 6, which implies that model is good at capturing true positive cases but at the cost of more false positives while labelling these classes.

As for the **F1-Score**, a harmonic mean of **Precision** and **Recall**, is a more reliable indicator to evaluate the performance. It shows a downward trend as the number of instances in each class decreases.

Overall, labels with more instances in the validation set are more likely to be accurately predicted, and the model tends to better identify majority classes in the dataset. For classes of Stable Height 3 and 5, interestingly, the model makes several

false positive predictions to recognise these classes. Finally, the model struggles to effectively distinguish the class, Stable Height 6 due to the small amount of data.

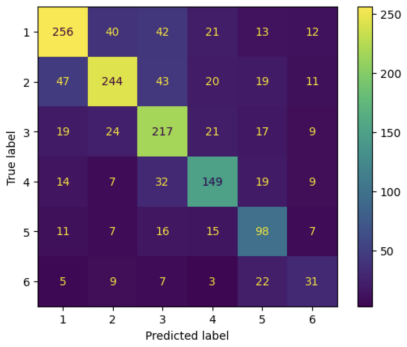

Fig. 4. Confusion Matrix

TABLE III
KEY METRICS FROM THE CONFUSION MATRIX

Stable Height	1	2	3	4	5	6	Average
Precision	0.7272	0.7371	0.6078	0.6506	0.5212	0.3924	**0.6060**
Recall	0.6666	0.6354	0.7068	0.6478	0.6363	0.4025	**0.6159**
F1-Score	0.6956	0.6825	0.6536	0.6492	0.5730	0.3974	**0.6075**

V. DISCUSSION

During the training process, we observe that the validation accuracy is consistently higher than the training accuracy, with a difference of up to 16%, and the test accuracy is similar to the validation accuracy. This gap is unlikely caused by overfitting or underfitting since our model can generalise reasonably well after 50 epochs with the training accuracy of roughly 50%, validation and test accuracy over 60%. We have ensured that the training and validation sets are stratified, hence, we infer that the difference might come from regularisation.

One regularisation method we used is dropout, which forces the model to rely on different neurons during training, so it becomes less sensitive to the specific weights of neurons. During evaluation, dropout is disabled, meaning all neurons are active and contribute to predictions. The common consequence of this method is lower training accuracy compared to validation accuracy, as seen in our accuracy and loss curves.

Another regularisation technique we used to mitigate overfitting is data augmentation, which increases the diversity of the training dataset by image cropping, rotation, colour jitter and so on. Applied with this method, the model prevents learning the same patterns and noise from the training data. However, during the evaluation process, the model faces the unaugmented validation data that is easier to predict. This might be one of the reasons causing the validating losses lower than the training losses during the entire training process.

To verify our assumptions, we sampled 10% of the unaugmented training data, fed them into the model in each epoch, and plotted its loss and accuracy (Fig. 5) along with the original training history shown in Fig. 3. As speculated, the figure shows that the green curves of the unaugmented training data generally remains the same trend as the other curves, but demonstrates slightly lower loss and higher accuracy, particularly in the later epochs. This implies that the augmented training data is the main cause of the performance gap between training and validation data, and was effective in challenging the model and help it generalise.

Fig. 5. Training History Curves with Unaugmented Training Curve

VI. CONCLUSION

With the above steps and finding, we have trained a model to predict the Stable Height of a shape stack with an validation accuracy of 64.78%. During the tuning, we have found that generally, a larger learning rate suits better for training a deeper and more complex network, referred to [8]. The data augmentation contributes most to the performance improvement with a growth in accuracy for more than 10%. It also mitigates overfitting caused by imbalanced classes and image similarity. Finally, by comparing the accuracy and loss curves of the validation and unaugmented training dataset, we can conclude that our model doesn't overfit or underfit given gap between training and validation accuracy.

VII. FUTURE IMPROVEMENTS

There are still some improvements can be made to our machine learning pipeline and model architecture.

First of all, as we gained insight of leveraging different fine-tuning techniques and the prediction mistakes made by the model, we can switch to other advanced and specific models that might be more suitable for this task.

Secondly, we utilised only the images in the provided dataset to predicts one class - Stable Height, whereas there is a unused spreadsheet, *train.csv*, that contains categorical features such as Shapeset, Type, Total Height, Instability Type, and Camera Angle. We can incorporate those information in the training and re-define the goal of our model.

As a result, our ongoing improvement is to develop a multi-task model that predicts not only the Stable Height, but also other features provided in the spreadsheet. By training under this pipeline, the model can learn from multiple targets and improve the predictions on our final goal, Stable Height.

REFERENCES

[1] O. Groth, F. B. Fuchs, I. Posner, and A. Vedaldi, "Shapestacks: Learning vision-based physical intuition for generalised object stacking," 2018. [Online]. Available: https://arxiv.org/abs/1804.08018

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019. [Online]. Available: https://arxiv.org/abs/1801.04381

[4] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," 2019. [Online]. Available: https://arxiv.org/abs/1905.02244

[5] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," 2020. [Online]. Available: https://arxiv.org/abs/1912.11370

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[7] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," 2012. [Online]. Available: https://arxiv.org/abs/1206.5533

[8] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay," 2018. [Online]. Available: https://arxiv.org/abs/1803.09820

[9] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, Jul 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0

[10] S. G. Müller and F. Hutter, "Trivialaugment: Tuning-free yet state-of-the-art data augmentation," 2021. [Online]. Available: https://arxiv.org/abs/2103.10158

[11] J. Byrd and Z. C. Lipton, "What is the effect of importance weighting in deep learning?" 2019. [Online]. Available: https://arxiv.org/abs/1812.03372