Assignment 2 - Group: 91

Group members:

1. **540371165**
2. **540367825**
3. **540130045**

Abstract

For the age estimation task, we use three different architectures of CNN which are Resnet18, EfficientNet and Swin Transformer. We evaluate and compare these models based on key metrics such as MSE(Mean Squared Error). Swin Transformer V2 achieved the lowest validation MSE but showed signs of overfitting. ResNet-18 demonstrated strong convergence on the training data but faced challenges on unseen data as its validation MSE doesn't improve much over time. EfficientNet delivered stable and consistent results, achieving competitive accuracy while minimizing computational demand. All these findings provide important insights into the capabilities and limitations of each architecture, helping inform their suitability for age estimation tasks.

1. Introduction

The problem we choose falls in the category of regression. We focus on apparent age estimation, which is to predict the age of individuals from still images using machine learning techniques. The original dataset is IMDB-WIKI dataset, containing over 500k face images with age and gender labels.

Apparent age estimation has numerous real-world applications. For example, companies can use apparent age information to deliver personalised advertisements and product recommendations based on the user's estimated age to improve target accuracy. It can also be used to detect individuals in specific age ranges for security checks or to monitor restricted areas that are age-sensitive like alcohol and tobacco stores.

In 2015, the winner of the ChaLearn LAP tackled the task with CNNs of VGG-16 architecture which are pre-trained on ImageNet for image classification. The network is also called Deep EXpectation(DEX). It first detects the face in the test image and then extracts the CNN predictions from an ensemble of 20 networks on the cropped face. The CNNs of DEX are fine-tuned on the crawled images and then on the provided images with apparent age annotations, it doesn't use facial landmarks. By posing the age estimation problem as a deep classification problem followed by a softmax expected value refinement, they achieved state-of-the-art results for age estimation.

The deep learning techniques our team implements are Resnet-18, EfficientNet and Swin Transformer V2.

2. Data

2.1 Data description and exploration

The IMDB-Wiki dataset is one of the largest publicly available datasets of face images with gender and age labels for training. A list of the most popular 100,000 actors as listed on the IMDb website are crawled from their profiles date of birth, name, gender and all images related to that person. The face images are crawled in the same way from pages of people from Wikipedia with the same meta information. In total the dataset contains 460,723 face images from 20,284 celebrities from IMDb and 62,328 from Wikipedia, thus 523,051 in total. Images without timestamps are removed. Images with single faces are assigned with biological age for they are likely to show the actor and correct timestamp and date of birth. For this assignment, we use a subset of the original IMDB-Wiki dataset which contains 40,000 face images stored in 100 folders.

The dataset also comes up with a separate *.mat* file containing all the meta information which helps us further explore and process the data. The metadata describe the 'dob', 'photo_taken', 'full_path', 'gender', 'name', 'face_location', 'face_score', 'second_face_score', 'age' of the individuals of the images. The 'face_location', 'face_score' and 'second_face score' are the former work results of the IMDB-Wiki Faces dataset. The face_location points out the "face" position, and the face_score decides the quality and confidence of this face whether it is human face, the second_face scores decide the multiple human face existence situation of the images.

With the face_score and second_face_score, we are able to remove bad quality images which are not suitable for age estimation. However, after reviewing the images, we find that the face score doesn't perform well on identifying face images since the result is from a work of VGG-16 in 2015. The face scores are not reliable enough to conclude from our EDA process.. It can neglect some pictures that have a small face or the face is far away in the picture. Besides, the white race's face images are likely to achieve a higher score than the yellow and black race. People with moustaches are also more likely not to be considered as a face image. Therefore, we decided to use Insightface to detect face images and filter valid data.

The second_face_score is used to detect face images with more than one face. Some pictures do have 2 or 1.5 human faces so this is useful since we only have one age label per image. We use an experienced threshold of 8 to define the picture whether regarded as a noise for an exotic number of people or a receivable picture that only has some human-like figure or has an obvious human face as a main character who should be recognized by the model.

2.2 Pre-processing

2.2.1 Data Cleaning

First, we import the opencv package and use it to load the image from the Wiki-Face dataset directory. We also define functions to get image paths from the directory and also from dataframe for we are going to use information from the .mat file.

The dataset contains a .mat file which contains scores for face detection. For the face detection part, we use Insightface. It is an open-source deep learning framework for face detection, recognition, and analysis, providing state-of-the-art models and tools for various face-related tasks, including face alignment, verification, and feature extraction(Deng, J., Guo, J., Xue, N., & Zafeiriou, S. (2019)). We use the Face Analysis app with a pre-trained model 'antelope' to detect faces. We categorise the results into two folders. Images with faces are categorised into a "clean" folder, while images detected with no face are categorised into a 'Noise' folder.

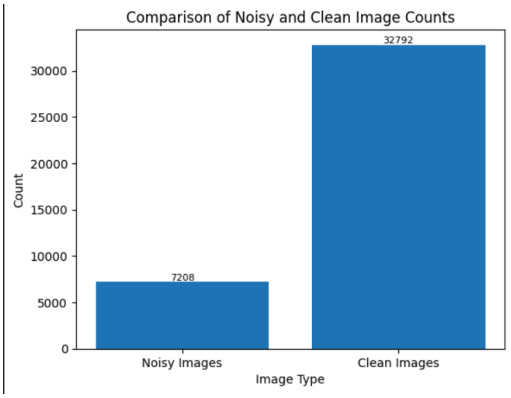

Figure 1. Comparison of Noisy and Clean Image Counts

After categorising the face images into two folders, we use second_face_score to clean the dataset a second time as discussed in the dataset exploration part. We load the .mat file and build a dictionary to convert it to a dataframe. We then filter valid data with age between 0 and 130, face score greater than 0.0 and second face score less than 8 or NAN to create a 'filter_wiki' data frame. Data that do not satisfy the condition are put into a 'delete_wiki' dataframe.

With the 'filter_wiki' dataframe we can load the 'full_path' parameter to track the cleaned image's path. The double-cleaned dataset's distribution of different columns is displayed below.

Through observing Figure 2, the most common ages of this dataset are from 20 to 30, and the labels are not balanced. In this case, the model fitting may have a large challenge, it is possible to learn too many features of common age and ignore the features of uncommon age like age 90.

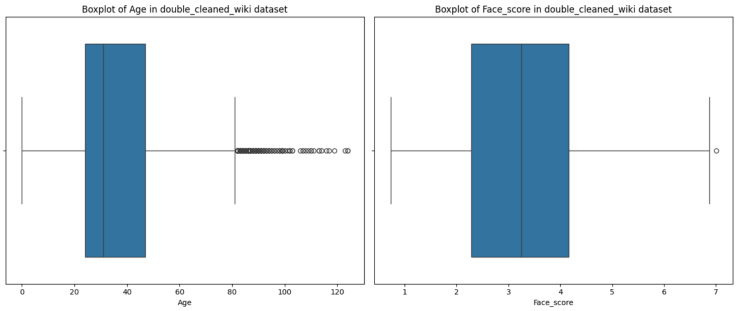

Figure 2. Boxplot of age and face score in the double-cleaned dataset

After cleaning the dataset twice, we categorise images into different folders based on each image's age. We replace the base path of the cleaned images dataset folder with a relative path so that each image can be matched with the dataframe's full path column. For each matched image found, we extract the age value from the age column and create a subfolder for the image by age. We use the make_folder() function to ensure the folder exists and after that, we delete folders containing less than 10 images.

2.2.2 Normalisation and spilt the dataset

We split the dataset into train, validation, and test sets. 80% of the dataset are for the training purpose, 10% for validation and the remaining for evaluating the model's performance. We maintain a consistent split ratio across all age groups to ensure each age-specific folder has the same proportion of images for training, validation, and testing.

We use transforms.ToTensor() function to convert the image to tensor and apply it to the training set to create the dataset for computing the mean and standard deviation.

We define a set of preprocessing operations in the transform() function. ensureThreeChannels() is to ensure that all input images have three colour channels (R, G, B) which can avoid errors when the dataset contains images with

inconsistent modes. We resize all images to 224x224 pixels to ensure uniform input size, making them compatible with the neural networks we are going to implement.

$$Z = (X - \mu) / \sigma \quad (1)$$

Using the pre-calculated mean and standard deviation to normalise pixel values, ensuring the data has a zero mean and unit variance (1). This can help the model converge faster during training.

We then organise images into separate folders for training, validation, and testing, using folder names as class labels. Using ImageFolder() to load all the datasets and apply the defined transformation. We build a customised DataSet Class Inheriting the Dataset Class of the pytorch module. The reason is for data loaders multi-paralleling data inputting to the model. During the process of training, we find out that If the DataLoader is built by the embedded dataset class but not customised Dataset Class Inheriting the Dataset Base Class, the DataLoader will meet kernel corruption error for multi-paralleling working even if the data loader has a num_workers parameter to appoint the concurrency.

3. Methodology

3.1 Resnet18

3.1.1 Theory

In the traditional convolutional neural network (CNN), the input is processed by multiple convolutional layers and pooling layers to produce a high-level feature representation. However, when the network gets deeper, the propagation of the gradient will encounter difficulties, resulting in degraded model performance. Through using residual blocks, which retrain input information in the network, ResNet enables the neural network to learn the difference between input and output (residual), rather than directly learning complex output features (Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun, 2016). This structure can effectively solve the problem of disappearing gradients so that the model can converge faster and build deeper networks.

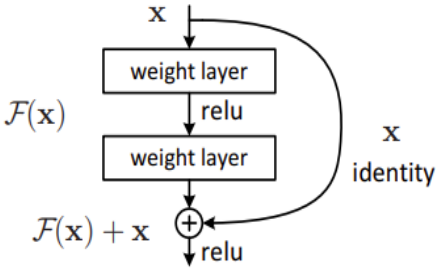

Figure 3. Residual learning: a building block (Kaiming He et al., 2016)

Figure 4 shows the residual learning of the Resnet model. Through adding the identity mapping in the network learning, the network is designed as F(x) + x.

$$y = F(x, \{W_i\}) + W_s x \quad (2)$$

" We can also use a square matrix W_s in Eqn. (2)" (Kaiming He et al., 2016).

3.1.2 Strengths and weakness

Strengths:

1. Solving gradient disappearance problems in a deep network.

Gradient disappearance or explosion is a common problem in traditional deep convolutional neural networks (CNN). By introducing skip connections, ResNet is able to pass input directly to subsequent layers without the gradient disappearing.

2. Less number of parameters and computation, compared with ResNet-101 and ResNet-50.

The ResNet-18 has only 11.7 million parameters, much less than the ResNet-50 (about 25.5 million parameters) and ResNet-101 (about 44.5 million parameters), which makes it more suitable for scenarios with limited computing resources and datasets.

Weakness:

For large datasets or complex tasks, expressiveness may be inadequate.

Its feature extraction ability is limited compared with deeper networks such as ResNet-50 and ResNet-101. This is because fewer convolutional layers and shallower network depths limit its ability to extract high-level semantic features.

3.1.3 Architecture and hyperparameters

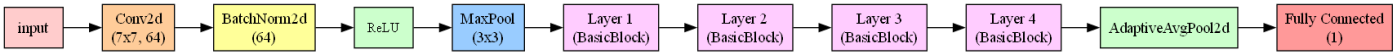

Figure 4. Architecture of ResNet-18

ResNet-18 is a shallower convolutional neural network with 18 layers.

The following are the key architecture designs of the ResNet-18:

1. Convolution layer: the first few layers of the model are standard convolution and pooling operations.
2. Residual blocks: ResNet-18 consists of multiple residual blocks.
3. Global average pooling layer: at the end of the network, a global average pooling layer is used.
4. Fully connected layer: the final output is a fully connected layer for regression prediction.

The following are the key hyperparameters we selected to tune the model:

1. Learning rate: the learning rate determines the stride size of the model weight adjustment at each update. A suitable learning rate can fit the model faster and better.
2. Optimizer: the optimizer decides how to update the parameters of the model according to the gradient. Some optimizers are better able to explore more complex loss surfaces to avoid falling into local optimality.

3.2 EfficientNet

3.2.1 Theory

It's common knowledge that deeper ConvNet can capture richer and more complex features and generalize well on new tasks. However, deeper networks are also more difficult to train due to the vanishing gradient problem. Wider networks tend to be able to capture more fine-grained features and are easier to train. While extremely wide but shallow networks are likely to have difficulties in capturing higher level features (Zagoruyko & Ko modakis, 2016). With higher resolution input images, ConvNets can potentially capture more fine-grained patterns, but the accuracy gain diminishes for very high resolutions, and it can significantly increase computation.

EfficientNet is designed to solve the problem. The core idea behind EfficientNet is a scaling method. By investigating the effects that every scaling strategy has on the effectiveness and performance of the model, the results show that although scaling a single dimension can help improve model performance, balancing the scale in all three dimensions is the best way to increase model performance overall (Tan & Le,2019). Therefore, the new scaling method EfficientNet uses uniformly scales all dimensions of depth, width, and resolution using a compound coefficient. The formula for scaling is :

$$depth: d = \alpha^{\varphi} width: w = \beta^{\varphi} resolution: r = \gamma^{\varphi}$$
$$s.t.\ \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2\ \alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

α, β, γ are constants that can be determined by a small grid search. By scaling with a constant ratio so that it can balance the width, depth, and resolution parameters.

EfficientNet uses depth-wise separable convolutions to lower computational complexity without sacrificing representational capability. Inverted residual blocks are also employed to further optimise resource usage. These blocks start with a lightweight depthwise convolution followed by point-wise expansion and another depthwise convolution (Tan & Le,2019).

3.2.2 Strengths and Weakness

Strength: EfficientNet is suitable for computer vision tasks and it consistently achieves SOTA performance on multiple benchmarks like ImageNet. It reached 84.3% accuracy on Imagenet top-1, which was the toppest in 2019. Therefore, EfficientNet is suitable for our age estimation task. By utilising compound scaling to balance width, depth and resolution, EfficientNet delivers high accuracy with lower computational costs because traditional CNNs often scale only one aspect at a time, which can lead to inefficiencies. Moreover, the compound scale method can minimise the chances of overfitting, especially when working with smaller datasets. This balance helps the network generalise better without relying on large amounts of data.

Compared to other CNN architectures like ResNet or VGG, EfficientNet has fewer parameters. The lightweight version EfficientNet-B0 only has 5.3 million parameters which is much lower than Resnet or VGG. This significantly fastens the training and inference process.

Weakness: The core idea of EfficientNer, the compound scaling is a double-edge sword. For scaling all three dimensions needs careful fine tune and understanding, It can be harder for developers to modify the architecture compared to simpler models. While the base model EfficientNet-B0 is lightweight, a larger model like EfficientNet-B7 will still cost a lot of computational resources.

EfficientNet also has limited interpretability. Its multiple layers and complex structure like MBConv and squeeze-and-excitation (SE) blocks can make it difficult to explain how specific predictions are generated.

3.3.3 Architecture and Hyperparameter

According to Tan & Le, The architecture of Efficinetnet-B0 can be summarized as follow:

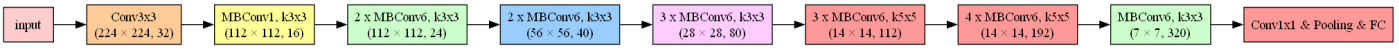

Figure 5. Architecture of EfficientNet-B0

1.MBConv layers are the backbone of EfficientNet, derived from MobileNetV2. They expand features using depth-wise convolutions, process them, and compress them back to reduce computational load.

2. The 3x3 kernels are commonly used for general-purpose feature extraction and 5x5 kernels are for capturing broader patterns and contextual information,

3. As the input resolution becomes smaller, the network moves from capturing low-level details to high-level abstract patterns.
4. The global average pooling layer compresses all spatial information into a single vector, which is fed into a fully connected layer to make the final classification or prediction.
The hyperparameters we are going to change in EfficientNet are the same in Resnet18.

3.3 Swin Transformer V2

3.3.1 Theory

According to the Swin Transformer paper of Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo (2021a), The Swin Transformer is a vision Transformer model, which can capably serve as a general-purpose backbone for computer vision. The Swin Transformer adapts the transformer from language to vision domains, resolving new challenges like the large variations in the scale of visual entities and the high resolution of pixels in images compared to words in text. The Shifted windowing scheme of the Swin Transformer brings greater efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for cross-window connection. This hierarchical architecture of the Swin Transformer has the flexibility to model at various scales and has linear computational complexity with respect to image size.(Liu et al., 2021a, p. 1).
Compared to the Vit-Vision Transformer, the Swin Transformer introduces a linear computation complexity and higher resolution method of hierarchical feature maps for image inputting that utilises the local window computation advantage of the self-attention mechanism.(Liu et al., 2021a, p. 1)
According to the Swin Transformer V2 paper of Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, Baining Guo (2021b), The Swin Transformer V2 is an improved version of Swin Transformer, which is released by the same team of Swin Transformer after 8 months of the releasing of its first version. The main idea of Swin Transformer V2 is to explore large-scale models in computer vision. The three main techniques of Swin Transformer V2 are the residual-post-norm method combined with cosine attention to improve training stability, the log-spaced continuous position bias method to effectively transfer models pre-trained using low-resolution images to downstream tasks with high-resolution inputs, and the self-supervised pre-training method, SimMIM to reduce the needs of vast labelled images.(Liu et al., 2021b, p.1)

3.3.2 Strengths and Weakness

Strengths:

The Swin Transformer V2 broke a series of new records and was the State Of Art model after the Swin Transformer on 4 representative vision tasks, including Imagenet-V2 image classification, CCO object detection, ADE20K semantic segmentation and Kinetics-400 video action classification. (Liu et al., 2021b, p. 1)
The main strengths of Swin Transformer:
- An efficient model compared to the former transformer family models.
- Elastic for large dataset input and training
- Enabling the scaling law of the large model of Computer Vision
- Better performance upper boundary on Computer Vision Tasks compared to CNNs through Large Dataset and Sufficient and Appropriate Training

Weakness:

- High Computational Resources requirement: the GPU memory consumption of Swin Transformer V2 is very high, especially for high-resolution images or the family version of larger model variants.
- Training time: Slower Training time compared to CNN due to the computational overhead of self-attention mechanisms, even with window partitioning.
- High Dataset Requirement: Swin Transformer requires high data quality and diversity, otherwise the model effect might worsen
- Pre-training Requirement: Usually requiring the pre-trained weights from large datasets, worse performance if training from scratch on smaller datasets
- Slower Inference time: The Swin Transformer has a more complicated model structure that slows the inference process compared to CNNs.

3.3.3 Architecture and Hyperparameter:

As shown in **Figure 6 (Liu et al., 2021a)**, The architecture of the Swin Transformer is a combination of several stages of Swin Transformer Blocks and Preprocessing Blocks such as Linear Embedding Blocks or Patch Merging Blocks. The Swin Transformer Block uses windowed self-attention instead of global self-attention, which means each block divides the input image into local windows computes self-attention within windows and has less computational complexity. Meanwhile, the Swin Transformer blocks incorporate a hierarchical structure by progressively reducing the resolution of the input feature maps across multiple stages, similar to how CNNs do. This is realised through the merging operations between stages. The other part of the Transformer Block is similar to traditional Transformer Blocks. The transformer Blocks are connected through one by one sequence. (Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo (2021a))

As is shown in **Figure 7(Liu et al., 2021b)**, The Swin Transformer is the extended version of the former Transformer Blocks, which is designed to further enhance its performance and scalability, especially for high-resolution images and larger vision models. The main innovation of Swin Transformer remains the same while the new techniques about model efficiency, stability and representation power are introduced. The Swin Transformer Blocks' shifted window Self-Attention is upgraded into a window-based multi-head self-attention(W-MSA) block, which introduces the residual-post-norm method combined with cosine attention and log-scaled initialization for the Self-attention Layer Norm.

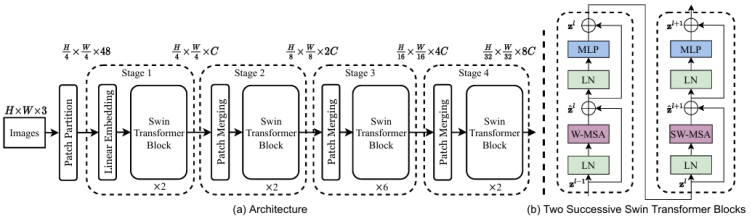

Figure 6. (a) Swin Transformer overview architecture;(b) Two Successive Swin Transformer Blocks

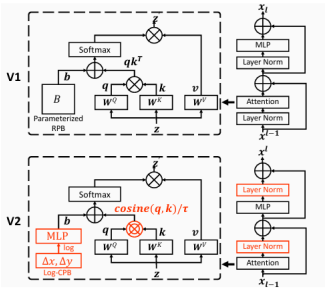

Figure 7. Swin Transformer V2 Blocks

Hyperparameter:
Considering the computational complexity, The Swin Transformer V2 model is fixed as the swinv2_cr_tiny_224, which is a compressed and tiny version of the Swin Transformer since we lack computing resources.
The hyperparameters we are going to change are learning rate and optimizer, which are the same in Resnet18.

4. Experiment Results

4.1 Experimental Setting

4.1.1 Dataset details

	Details
Dataset Details	The dataset includes 40,000 face images and associated metadata from IMDB-Wiki Faces, after the cleaning process, 32769 images are sent to the later stages. As discussed in part 2, we first clean the non-human noise by Insightface - detect_face(images) function and then clean the multi-human picture by the 'second_face score' and 'age'.Categorizing images into different folders based on each image's age, split them into three sets for training, testing and validating. Apply a transformer to all the data and use a data loader to efficiently feed the transformed data into the model during training.
Regression Task	This dataset is for Age estimation: Predict the age of individuals in the images using machine learning techniques, such as convolutional neural networks (CNNs), RNN or Transformer.
models	1.ResNet-18 2.EfficientNet-b0 3. Swin Transformer V2 with MLP - swinv2_cr_tiny_224 with 3 layers Multiple Perceptron Layers

4.1.2 Model Details and Setting:

4.1.2.1 EfficientNet

The EfficientNet is set by pytorch customised Class inheriting the pytorch.nn.Module Base Class. The model is originated by Torchvision.models.efficientnet_b0(weights=None), extracting the model structure and initialising the

model by random value but not any pre-trained weights. The last classifier of EfficientNet is set as a regression regression neuron. The forward function returns the EfficientNet model.

4.1.2.2 Resnet 18

The ResNet-18 is also set by pytorch customised Class inheriting the pytorch.nn.Module Base Class. ResNet-18 is originated by Torchvision.models.resnet18(weights=None). In order to fit the regression task, we change the last all connection layer. The output of this layer is one.

4.1.2.3 Swin Transformer V2 with MLP

The Swin Transformer v2 with MLP model first loads the structure of the Swin Transformer v2 model from timm package. The output of the Swim Transformer V2 model is the number of classes, then this output would as the input of the MLP model. In the MLP model, all connection layers and activate function GELU() help the model to capture non-linear patterns.

4.2 Performance Metric of different models

This Metric is given based on the double-cleaned dataset of 32769 images which is normalized and reshaped into 224*224 size.The Test Mse of the implementation of ResNet-18, EfficientNet-b0 and Swin Transformer V2 are 152.0446, 146.4457 and 183.4713.

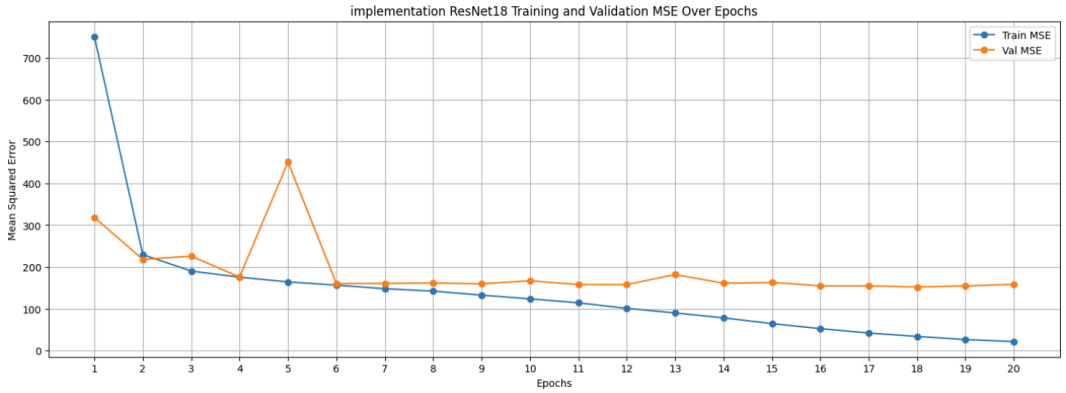

Figure 8. Training and validation MSE of implementation of ResNet-18 over epochs

Although the training loss is able to continue to decrease, the test Mse could not decrease after the sixth epoch. It means that although ResNet-18 can fit the training dataset very well, it lacks generalization ability. The features that the ResNet-18 captured can only be useful in training datasets. Maybe build a more complex model like ResNet-101 or enhance data to help the model capture different features that may help the model to have a higher performance.

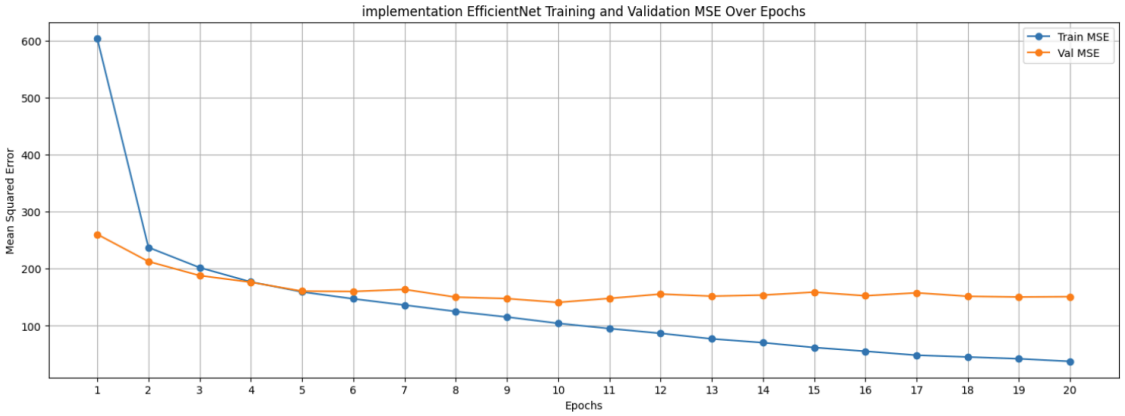

Figure 9. Training and validation MSE of implementation of EfficientNet over epochs

As we can see, the training MSE decreased steadily from 604.3596 to 37.6186, which shows a good learning ability. However, the validation MSE doesn't improve much after epoch 5, ranging from 147 to 152. The gap between Train MSE and Validation MSE indicates that the model may have overfitted the training data. The validation MSE didn't improve substantially even with further training. The reason may be that the learning rate (0.001) might have been too large, resulting in unstable convergence since the validation losses fluctuated in the last several epochs.

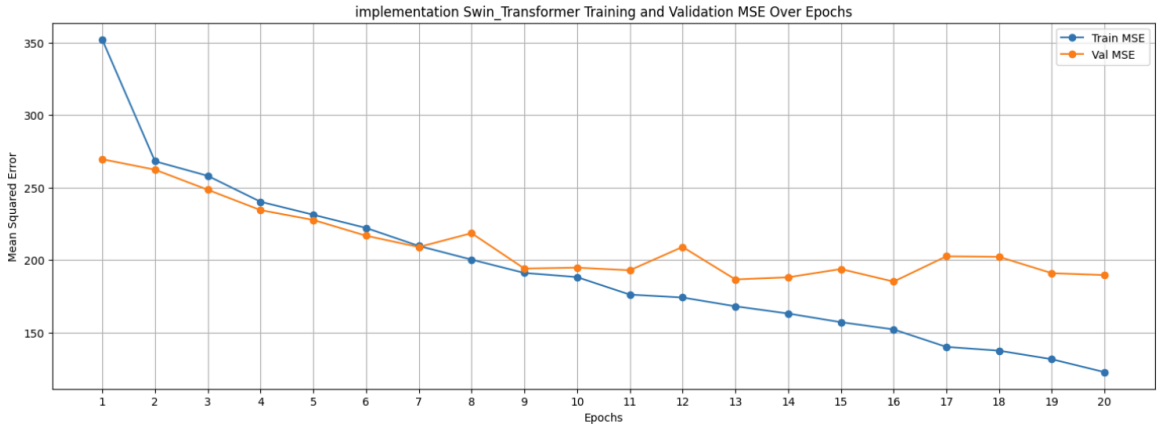

Figure 10. Training and validation MSE of implementation of Swin Transformer V2 over epochs

Unlike the other two models, Swin Transformer V2 fits lower, it reached the lowest validation MSE at epoch 16. As the same as other models, Swin Transformer V2 can also fit very well on the training dataset, which means overfitting because the testing dataset can not get a better performance. In order to get better performance, maybe tuning the learning rate and optimizer can be a good choice.

4.3 Hyperparameter tuning

Model	Train Mse	Test Mse	Learning Rate	Optimizer	Epoch	Batchsize
ResNet-18	115.3515	147.0566	0.001	Adam	12	512
EfficientNet-b0	91.5581	133.3530	0.0001	AdamW	19	128
Swin Transformer V2	117.6851	173.0199	0.001	AdamW	23	64

4.3.1 ResNet-18

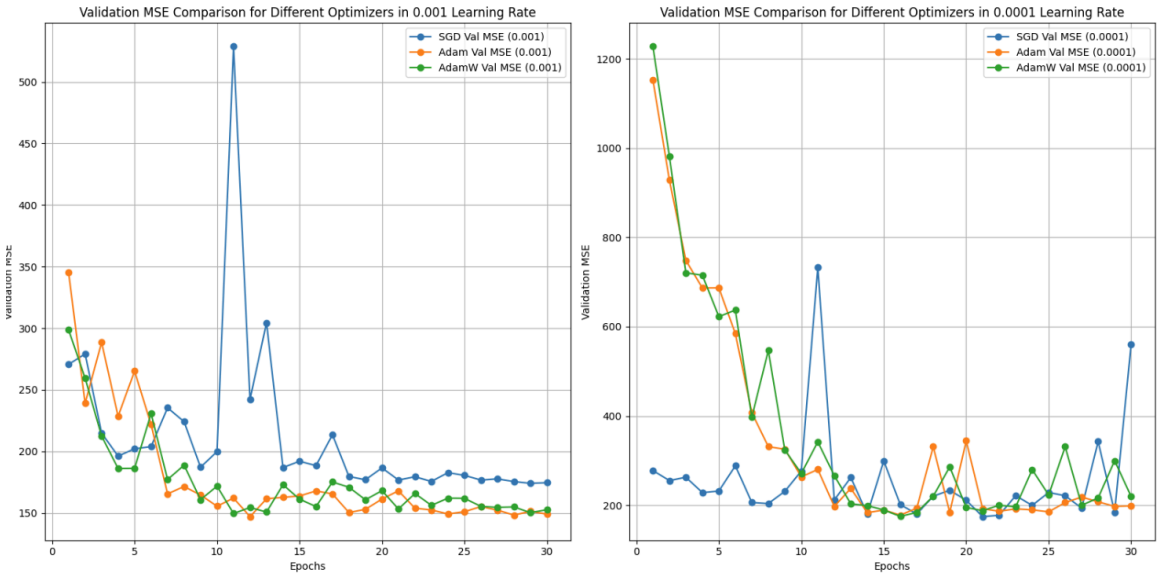

Figure 11. Resnet's Validation MSE Comparison for Different Optimizers in Two Learning Rates

It is not difficult to find that the results of hyperparameter adjustment that SGD fluctuates more violently. That's because SGD updates the weights with a fixed learning rate at each step. When the gradient value is large, the update step may be large, resulting in excessive weight update, resulting in large fluctuations. By contrast, both Adam and AdamW employ adaptive learning rates, this mechanism makes updates smoother, thus reducing volatility. As a result, The AdamW optimizer with a 0.001 learning rate got the best result of the validation dataset at epoch 12.

4.3.2 EfficientNet

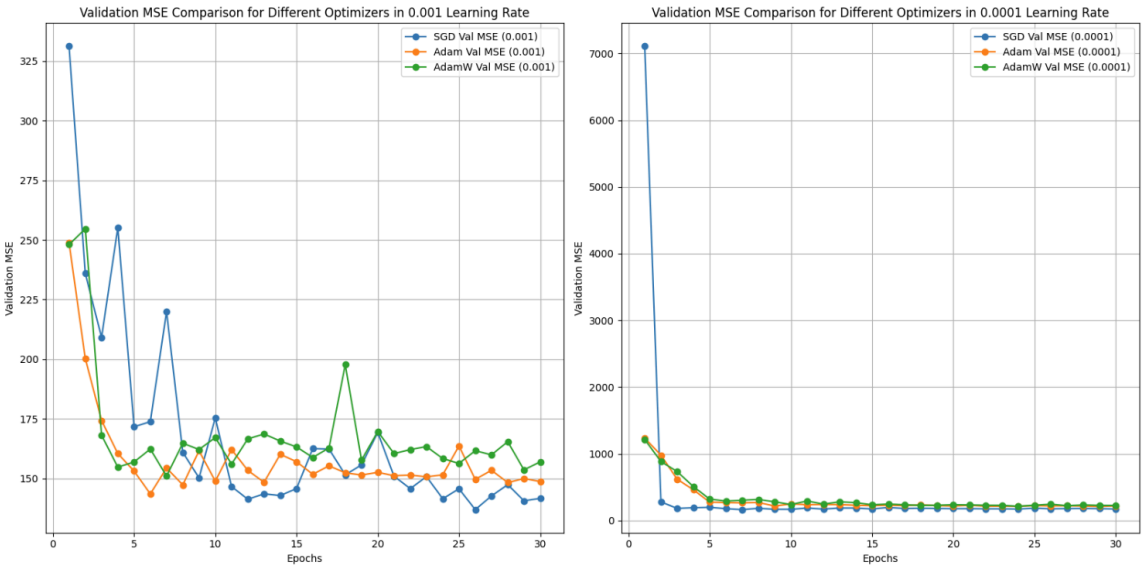

Figure 12. EfficientNet's Validation MSE Comparison for Different Optimizers in Two Learning Rate

The validation MSE of a Learning rate of 0.001 with all three optimizers fluctuates, especially with SGD and AdaM. Compared to the learning rate of 0.0001, a higher learning rate causes the model to overshoot the optimal point and result in unstable convergence. The result of setting the learning rate as 0.0001 with all three optimizers shows that the model improves more consistently over time and generalizes better results.

4.3.3 Swin Transformer V2

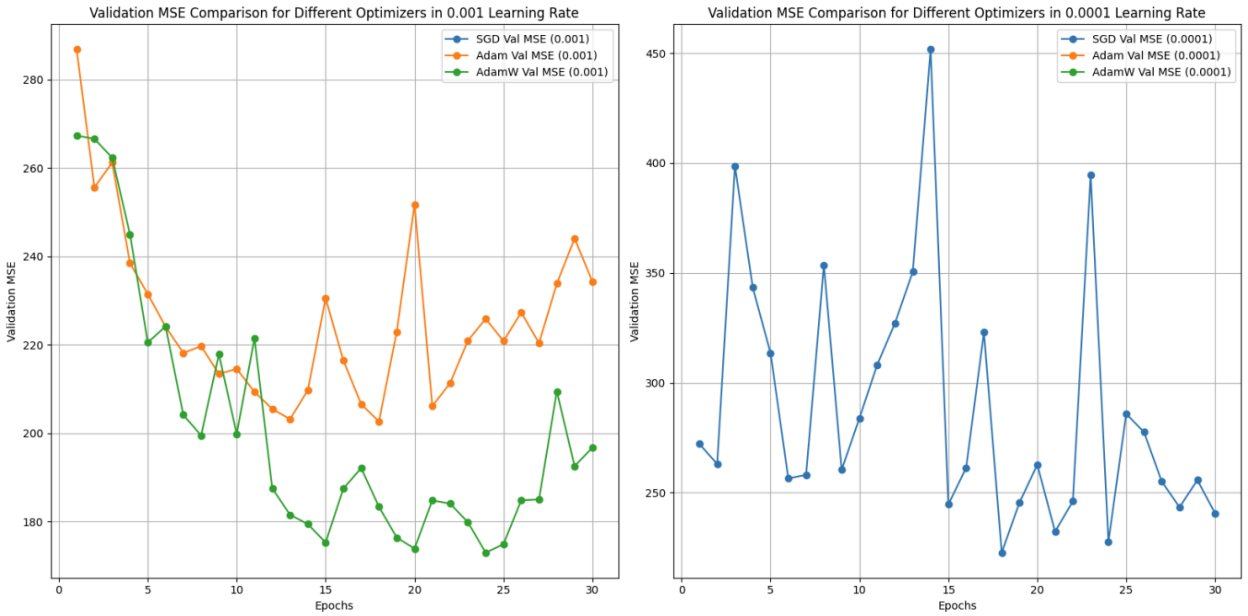

Figure 13. Swin Transformer V2's Validation MSE Comparison for Different Optimizers in Two Learning Rate

The Swin Transformer V2 takes 1.2 to 2 minutes per epoch for fine-tuning, it costs more time than other models. In the part of hyperparameter tuning, the AdamW optimizer decreased faster than optimizer Adam and finally got the best performance. By contrast, although we set a lower learning rate for SGD, it still decreased very low and sometimes increased dramatically. Although it still has a performance close to the Adam optimizer, the best option is AdamW.

4.4 Comparison

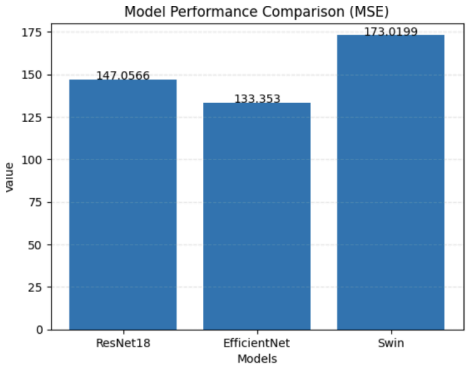

Figure 14. Model Performance Comparison(MSE)

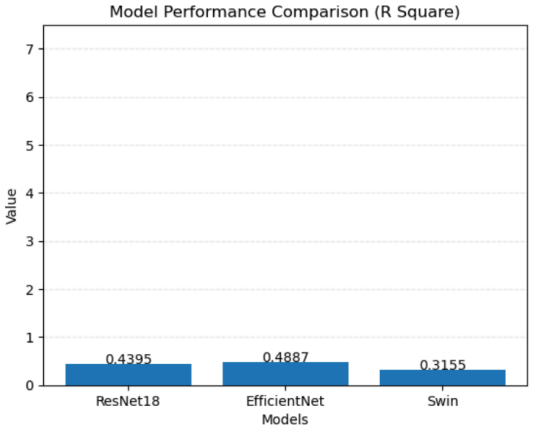

Figure 15. Model Performance Comparison(R Square)

By comparing the MSE, EfficientNet has the best performance, in contrast, ResNet's performance effect is not too bad, while Swin will perform poorly. By comparing the MAE, in fact the model's performance error is around one year. By comparing the R-square, the Swin model has a relatively worse fitting effect.

5. Conclusion

In conclusion, EfficientNet-b0 which has 133.3530 MSE on the testing dataset is the best model in this task, while ResNet-18 and Swin Transformer V2 have 147.0566 MSE and 173.01987 MSE respectively on the testing dataset. We found some limitations like imbalanced labels, hyperparameter tuning with high computational requirements, models prone to overfitting and the influence of noisy data. We tried to solve some of the problems through data preprocessing and hyperparameter tuning. We got high-quality images after data cleaning and got better performance after grid search with hyperparameters. Hyperparameter tuning can reduce the MSE by about 10, it is a meaningful process when conduct a deep learning task. We also tried to change the structure of the model in order to get higher performance, like Swin Transformer V2 with MLP.

After finishing the assignment, we have gained deeper insights into advanced neural network architectures and the various design strategies used to design and optimize them. For example, EfficientNet's compound scaling effectively balances width, depth, and resolution to achieve better performance with fewer parameters. Additionally, the Swin Transformer's hierarchical attention mechanism provided a new perspective on handling visual tasks by focusing on local and global features at different scales, enhancing the model's capacity to capture intricate patterns in images. Besides, we also developed practical skills in model training and management. Checkpointing mechanisms and automated saving of model states at different stages are very important when training complex networks with large datasets as it's very time-consuming. Early stopping during hyperparameter tuning can help prevent overfitting by halting training if the validation performance stops improving. Exporting training logs to text files for each epoch allowed for more effective tracking and comparison of results, improving the fine-tuning of hyperparameters and debugging of issues. All this experience we have learned can be applied to future projects and help us do a better job.

References

[1]Deng, J., Guo, J., Xue, N., & Zafeiriou, S. (2019). ArcFace: Additive Angular Margin Loss for Deep Face Recognition. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2019.00482

[2]Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778. https://doi.org/10.1109/CVPR.2016.90

[3]Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105-6114). PMLR.

[4]Zagoruyko, S., & Komodakis, N. (2016). Wide Residual Networks. Procedings of the British Machine Vision Conference 2016. https://doi.org/10.5244/c.30.87

[5]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).

Appendix

hardware	GPU: NVIDIA GeForce RTX 4090 CPU: Intel i9-14900KS 24-core 32-logical processors 6.2GHz RAM: 96GB DDR5 6800MHz SSD: Samsung 990 pro 2TB PCIE 4.0
software specifications	Windows 11 Pro python - 3.12.4 pytorch - 2.4.1 + cuda 12.4 onnx runtime-gpu 1.19.2 numpy - 1.26.3 numpy-base 1.26.4 onnx - 1.17.0 nbdime - 4.0.2 nbstripout - 0.7.1 notebook - 7.2.2 insightface - 0.7.3 Scikit-learn - 1.5.2 OpenCV - 4.10.0 Pandas - 2.2.2 Matplotlib - 3.9.2