Artificial Intelligence Concepts

Assignment2

HONG Yifan
22043798g

Code on Kaggle Notebook
https://www.kaggle.com/code/evanhong99/comp5511-cv

A COMP5511 Homework Assignment

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

November 12, 2022

1 How to run my code?

You can run my code through Kaggle Notebook (because they have free GPUs). All the data are well prepared.

https://www.kaggle.com/code/evanhong99/comp5511-cv

2 Task1: Classification with ResNet

2.1 Introduction

It is known to all that neural network works on a mechanism called "Backpropagation" which calculates the gradients of the loss function with respect to each weight by chain rule and updates the network.

2.1.1 Gradient Vanishing

Gradient vanishing occurs when the derivative or slope gets smaller as we go backward with every layer during backpropagation.

The training will take much longer when the weights update is relatively too small at the earlier layers. In the worst case, the update converges to 0, and our network will be "dead" at the first several layers.

A vanishing gradient problem occurs with the sigmoid and tanh activation function because the derivatives of the sigmoid and tanh activation functions are between 0-0.25 and 0â€“1. Therefore, the weights' updates are small, and the new weight values are very similar to the old ones. [3]

We can avoid this problem by using other activation functions such as LeakyReLU, or using pre-training and fine-tuning. Last but not least, ResNet was designed to solve this problem.

2.1.2 Gradient Exploding

Gradient exploding is the exact opposite of gradients vanishing. It happens when the derivatives are consecutively and significantly greater than 1. So our model will get hard to converge.

There are some solutions, such as gradient clipping or regularization.

2.1.3 Why Identity Mapping Avoids Gradient Exploding and Vanishing

According to K. He et al [2], they explain the details of why identity mapping is useful. The original Residual Unit in [1] performs the following computation:

$$\begin{gathered}\mathbf{y}_l = h(\mathbf{x}_l) + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l) \\ \mathbf{x}_{l+1} = f(\mathbf{y}_l)\end{gathered} \tag{1}$$

Here $\mathbf{x}_l$ is the input feature to the l-th Residual Unit. $\mathcal{W}_l = \left\{ \mathrm{W}_{l,k}|_{1\leq k\leq K} \right\}$ is a set of weights (and biases) associated with the l-th Residual Unit, and K is the number of layers in a Residual Unit (K is 2 or 3 in [1]). $\mathcal{F}$ denotes the residual function.The function f is the operation after element-wise addition, and in [1] f is ReLU. The function h is set as an identity mapping: $h(\mathbf{x}_l) = \mathbf{x}_l$

If f is also an identity mapping: $\mathbf{x}_{l+1} \equiv \mathbf{y}_l$, we obtain:

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l). \tag{2}$$

Recursively we will have:

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i) \tag{3}$$

For any deeper unit L and any shallower unit l, there exhibits some nice properties. (i) The feature $\mathbf{x}_L$ of any deeper unit L can be represented as the feature $\mathbf{x}_l$ of any shallower unit l plus a residual function in the form of $\sum_{i=l}^{L-1} \mathcal{F}$, indicating that the model is in a residual fashion between any units L and l. (ii) The feature $\mathbf{x}_L = \mathbf{x}_0 + \sum_{i=0}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)$, of any deep unit L, is the summation of the outputs of all preceding residual functions (plus $\mathbf{x}_0$). This is in contrast to a "plain network" where a feature $\mathbf{x}_L$ is a series of matrix-vector products, say, $\prod_{i=0}^{L-1} W_i \mathbf{x}_0$ (ignoring BN and ReLU).

Equation 3 also leads to nice backward propagation properties. Denoting the loss function as $\mathcal{E}$, from the chain rule of backpropagation we have:

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \left(1 + \frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right) \tag{4}$$

Equation 4 indicates that if and only if when the $\frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i) = -1$, the gradient will be 0. But it is almost impossible for all the terms $\frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} \mathcal{F}$ to be -1 in a mini-batch.This implies that the gradient of a layer does not vanish even when the weights are arbitrarily small. [2]

2.2 Model

We can see from the official source code that ResNet18 is constructed with four layers. Every layer is a "Basic Block," shown in Figure 1. It passes the input to the output and lets the model fit the residual. This process is called "Residual Learning."

The implementation of identity mapping by shortcuts is quite simple. What we need to do is just element-wise addition. So to cut off identity mapping, we can just simply add an if-statement (shown in the right of Figure 1).

```
def forward(self, x: Tensor) -> Tensor:
    identity = x
    ...
    if self.shortcut:
        out += identity
    ...
    return out
```

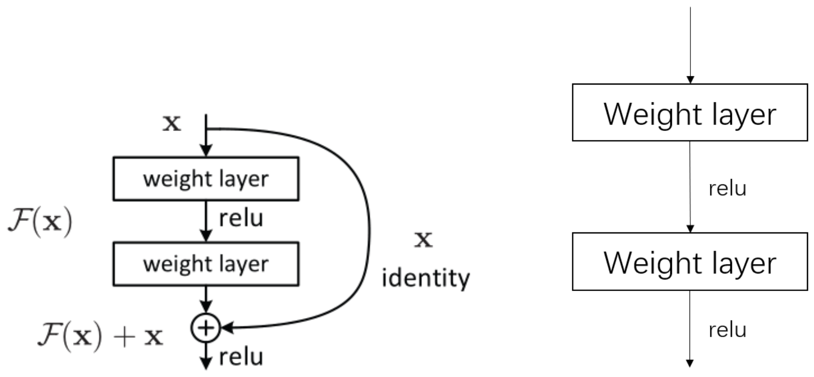


Figure 1: basicblock with/without shortcut

Figure 2 is the whole structure of ResNet18.

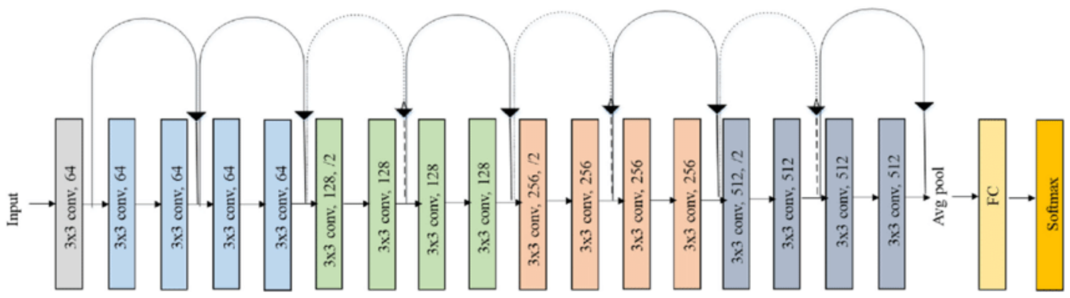


Figure 2: ResNet18 Structure

2.3 Experiment

2.3.1 Basic Settings

- criterion = CrossEntropyLoss
- optimizer = Adam
- scheduler=ReduceLROnPlateau
- num_epochs = 80
- learning_rate = 0.001
- initialization
  - Convolution Layers: kaiming_normal
  - BatchNorm Layers: weights are set to 1 and biases are zeros.

– Zero-initialize the last BN in each residual branch so that the residual branch starts with zeros, and each residual block behaves like an identity. This improves the model by 0.2 0.3.

2.3.2 Training

I trained each model for 80 epochs. It's enough (even a bit over-fitted after 60 epochs), and the models converge well, shown in Figure 3. The accuracy stays at about 85%, shown in Figure 4.

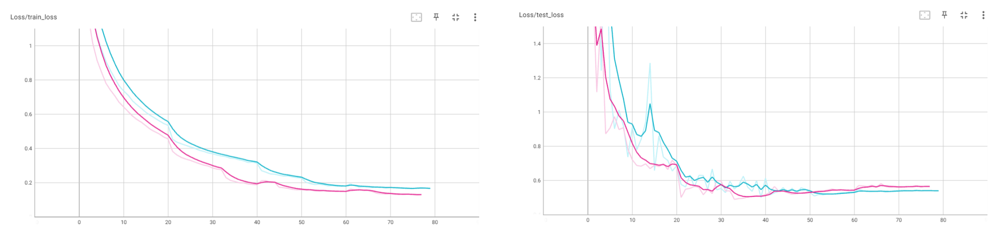


Figure 3: model loss. The red line indicates the model with shortcut

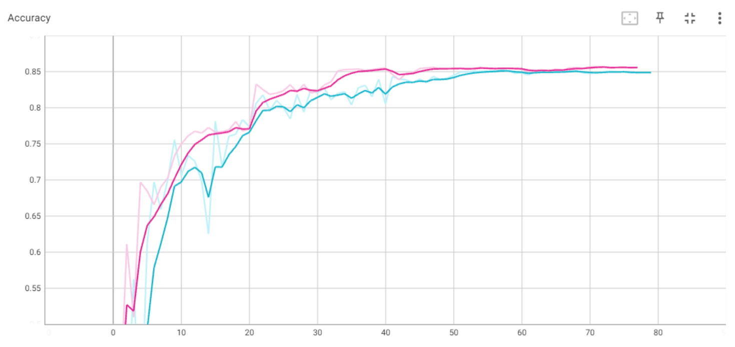


Figure 4: model accuracy. The red line indicates the model with shortcut

2.4 Conclusion

Figure 3 and Figure 4 show that identity mapping is helpful while training. But when the model converges, we can't see any advantages. I think the main reason is that our model is simple enough that gradient vanishing or exploding doesn't occur while training.

3 Task2: Imbalanced Face Mask Detection

3.1 Introduction

In this task, we need to implement some classification algorithms to solve the problem of face mask detection. However, the dataset we have is extremely imbalanced, where the number of samples in "with_mask" class is much greater than that of samples in "without_mask" class. One of the simple yet effective methods to alleviate imbalanced problems is to use data augmentation.

Data Augmentation

Data augmentation in data analysis is a technique used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It acts as a regularization and helps reduce overfitting when training a machine learning model.

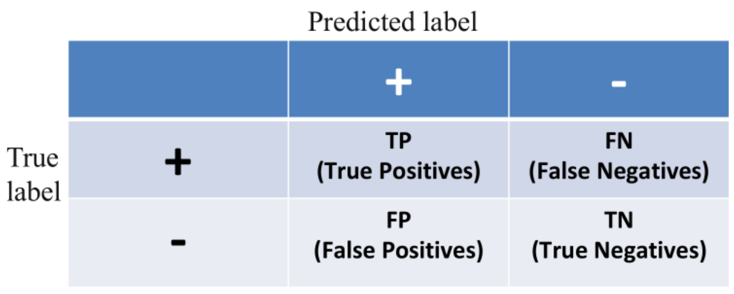
There are several ways to do so in image-type data, such as flipping, rotating, cropping, adding noise, etc.

Accuracy or G-Mean?

From the formula (Equation 5), we can see the difference between these two metrics. Accuracy evaluates the performance on whole dataset, while G-Mean takes more information into consideration. G-Mean is the geometric mean of accuracy of true samples and accuracy of false samples.

When our data is extremely imbalanced, and our model predict the true sample, the majority sample quite well, then we can still obtain a well accuracy. However, the G-Mean can point out the problem that if our model fits the minority samples well enough. If our model bad performs on minority samples, the G-Mean will be low.

However, whenever the data is balanced, these two metrics are quite similar to each other.



Predicted label

		+	-
True label	+	TP (True Positives)	FN (False Negatives)
	-	FP (False Positives)	TN (True Negatives)

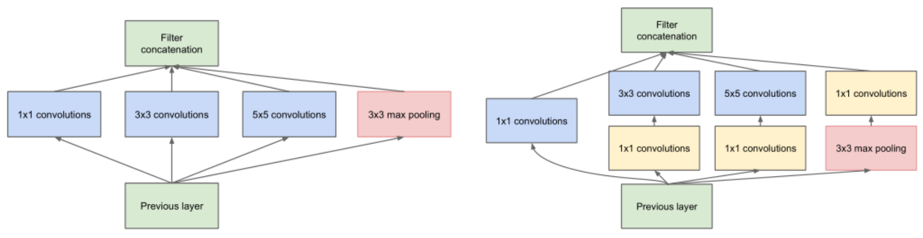
Figure 5: confusion matrix

$$
\begin{aligned}
Accuracy &= \frac{TP+TN}{TP+FN+FP+TN} \\
G-Mean &= \sqrt{\frac{TP}{TP+FN} * \frac{TN}{FP+TN}}
\end{aligned}
\tag{5}
$$

3.2 Model

To enrich the diversity of this assignment and learn from more SOTA models, I decide to tackle this task with GoogLeNet [4]. Because this model has a better performance and less parameters than VGG. What's more? It's not so complicated and heavy to train, compared with some later SOTA models such as the Transformers.

Christian Szegedy et al [4] proposed a deep convolutional neural network architecture named "Inception."



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 6: inception module

This structure (Figure 6) makes it possible for the network to have different receptive fields by different convolutional kernels. Additionally, this structure is easy to apply parallel computing. To reduce parameters, GoogLeNet used 1*1 convolutional kernels and maxpooling units.

Figure 7 shows the whole structure of GoogLeNet. To enhance the ability of classification, GoogLeNet also added two auxiliary classifiers.

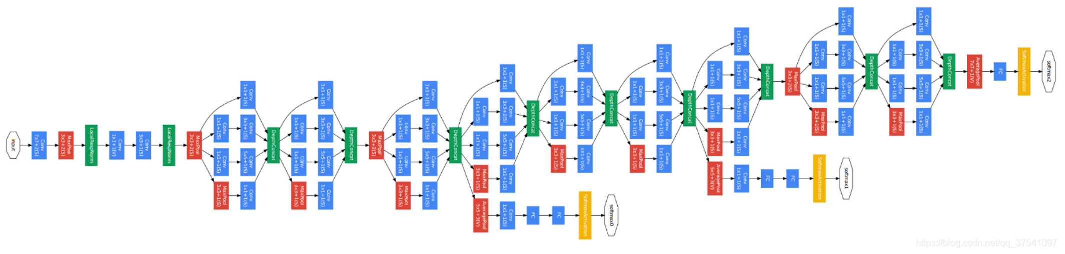


Figure 7: googlenet module

3.3 Experiment

3.3.1 Basic Settings

- criterion = CrossEntropyLoss
- optimizer = Adam
- scheduler=ReduceLROnPlateau
- num_epochs = 100
- learning_rate = 0.001
- batch_size=64
- loss_weight=[0.7,0.2,0.1]. This is the weight of three classifiers, including two auxiliary classifiers.

3.3.2 Data Augmentation

The original data is quite imbalanced.

	train	valid	test
with mask	500	300	500
without mask	60	40	90

Table 3: Imbalanced data

Pytorch gives us a convenient way to augment data. I choose some transformations, such as flipping, rotation, cropping, etc. Transformations are randomly picked. In the end, the number of samples of minority class increases to 360. The result is shown in Figure 8.

```
transform_rules=transforms.RandomChoice([
    transforms.RandomHorizontalFlip(0.7),
    transforms.RandomRotation(20),
    transforms.GaussianBlur((3,3), sigma=(0.1, 2.0)),

    ↪   transforms.Compose([transforms.Pad(10),transforms.Resize((128,128))]),
    transforms.RandomResizedCrop((128,128),scale=(0.8, 1.0)),
    transforms.RandomAdjustSharpness(1.25),
    transforms.RandomAdjustSharpness(0.75),
    transforms.RandomAutocontrast(),
],p=[0.7,0.7,0.3,0.5,0.6,0.2,0.2,0.3])
```

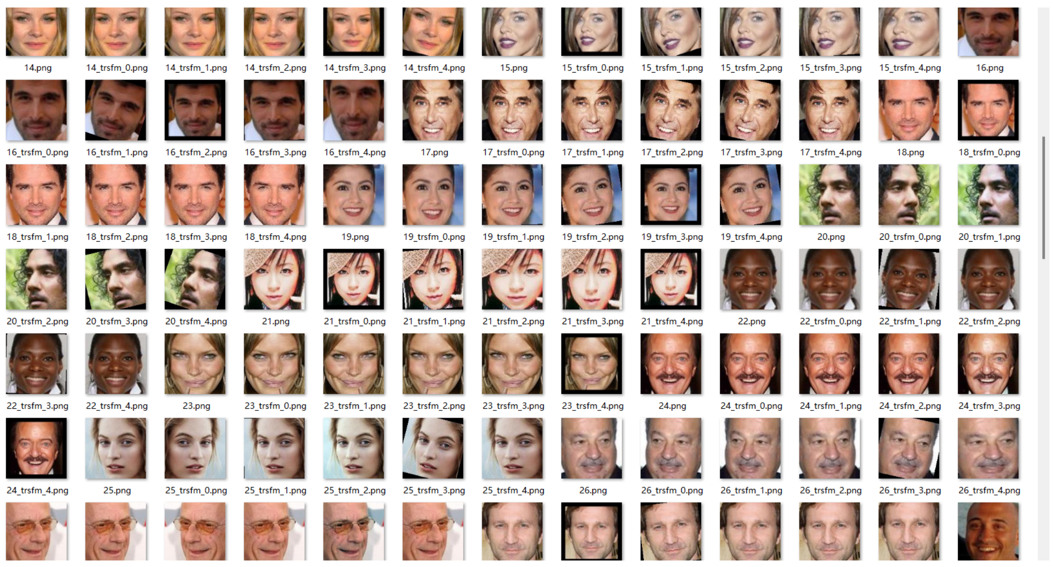


Figure 8: data augmentation

3.3.3 Training

I separately trained my model on the original dataset and the augmented dataset. Figure 9 shows that data augmentation is helpful, that our model obtains a lower loss and converges faster.

Last but not least, Figure 10 shows that data augmentation is of great importance in solving the problem of data imbalance. As accuracy increases about 0.02 on valid set and test set, the gmean value increases about 0.045 on valid and test set, which means that data augmentation not only enhances model's capability but also alleviates the problem of data imbalance.

```
original data
train
    train_loss: 0.0069437
valid
    valid_loss: 0.3874206
    valid_accuracy: 0.9647
    valid_gmean 0.9244
test
    test_loss 0.4772630558049275
    acc 0.9389830508474576
    gmean 0.9179687721631203

augmented data
train
    train_loss: 0.0027061
valid
    valid_loss: 0.1844178
    valid_accuracy: 0.9882
    valid_gmean 0.9714
test
    test_loss 0.3229771693379192
    acc 0.9576271186440678
    gmean 0.9567769971222251
```

3.4 Conclusion

In this part, I explained the definition of data augmentation and distinguished the differences between accuracy and G-Mean. Then I inspected the structure of GoogLeNet and Inception.

From experiment result we draw that data augmentation is of great use when data is imbalanced, and G-Mean is more suitable in this situation.

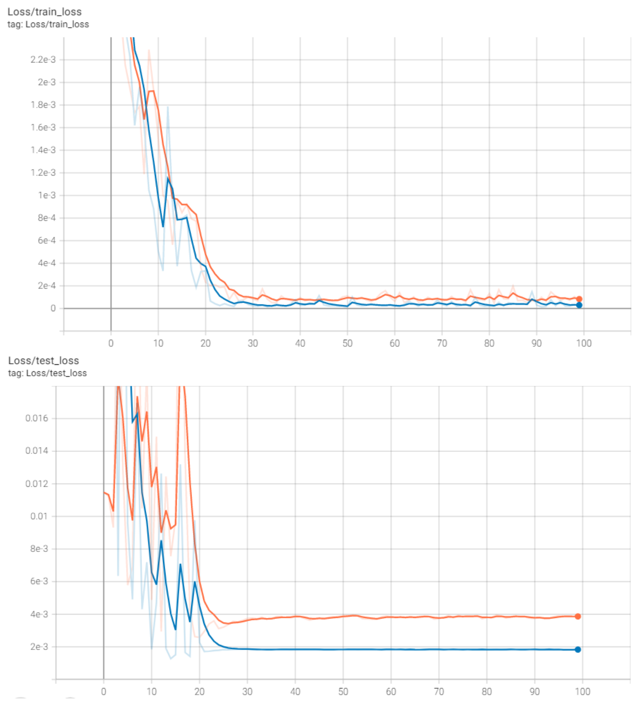


Figure 9: Loss. The blue line indicates the model trained on augmented dataset

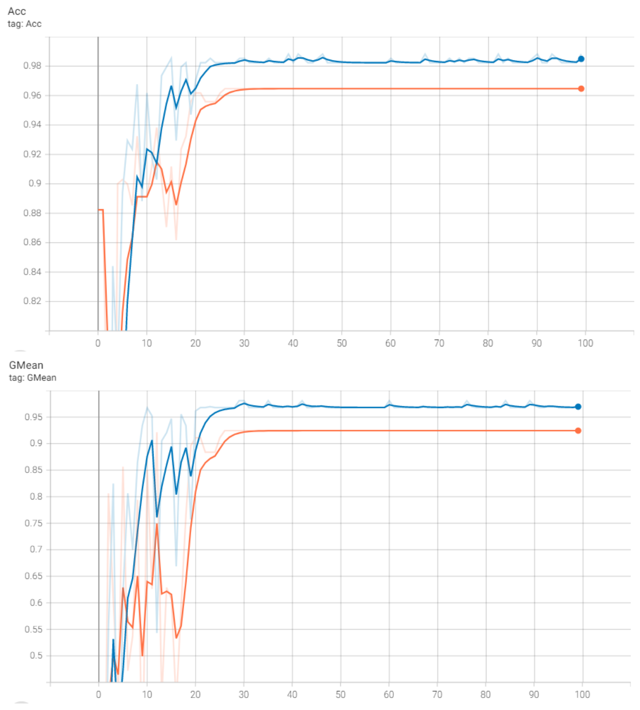


Figure 10: Accuracy and G-Mean. The blue line indicates the model trained on augmented dataset

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[3] Margi patel. Vanishing and exploding gradients in neural networks, 2021.

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.