National Tsing Hua University
Fall 2023 11210IPT 553000
Deep Learning in Biomedical Optical Imaging
Homework 3

WEN CHI CHEN[1]

[1]*National Tsing Hua University, IPT, HOPE Lab*

Student ID:

1. Model Selection: GoogLeNet & DenseNet

I choose **GoogleNet** to be my first transfer learning pre-trained model.

Previously, the mainstream of network structure breakthroughs was generally deeper (layers) and wider (neuron) networks.

However, there exist some disadvantages:

(1) Training set is limited and parameters are too much, the model will be easily overfitting.

(2) The larger the network, the greater the computational complexity.

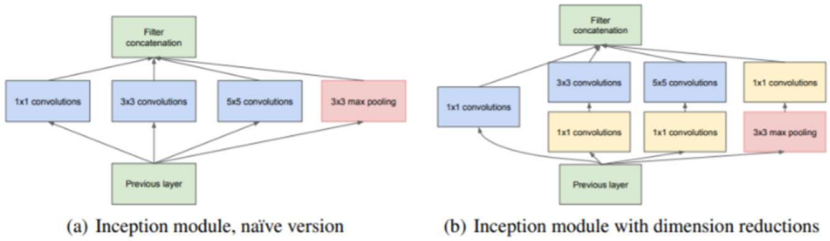
(3) As #layers increase, the gradient is likely to disappear in further backward propagation.

(4) Limited computing resources:

When Every two Conv layers are series together, which leads to an increasing calculation as the number of filters increases. However, the increased capacity is not used efficiently (for example, most weights end up close to zero), which creates waste.

GoogLeNet holds a core architecture: Inception. They take limited resources into account, optimizing the model so that it requires less memory or calculations. In the original Inception module, shown in Figure 1(a), 1×1 conv, 3×3 conv, 5×5 conv, and 3×3 max pooling will work serially to the input-to-output process. Conv/max pooling of different sizes will lead to extraction of multiple feature. However, the inputs of all conv are based on all the outputs of the previous layer, which obviously imposes a heavy computation (not to mention the 5x5 conv). Moreover, the image channel through max-pooling will remain unchanged, which will cause the feature channel to become larger as the depth of the CNN becomes deeper.

Therefore, modified original Inception module, as shown in the Fig 1(b), use 1x1 conv(s), which not only reduces the dimensionality of the feature image, but also increases the non-linearity of the network. Overall, Inception Network is the module on top of the stack, and some layers in the middle are combined with max-pooling of stride=2 to reduce resolution.



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

Fig. 1. Architectures of original Inception Module & modified Inception Module

The details of the architecture are as follows, and shown in Fig 2 as well:

(1) All Conv (including those in Inception) are followed by ReLu()
(2) The original input image is 224x224x3, and each pixel value of the image is subtracted from the average value.
(3) To avoid gradient disappearance, the network adds 2 additional softmax() to conduct the gradient forward.
(4) The network finally uses average pooling to replace the fully connected layer. GAP has several benefits: i. Enhanced the relationship between feature map and category. ii. There are no parameters to optimize in GAP, so the use of parameters can be reduced and overfitting can be avoided. iii. GAP combines spatial information and is more robust to spatial transformation of input features.

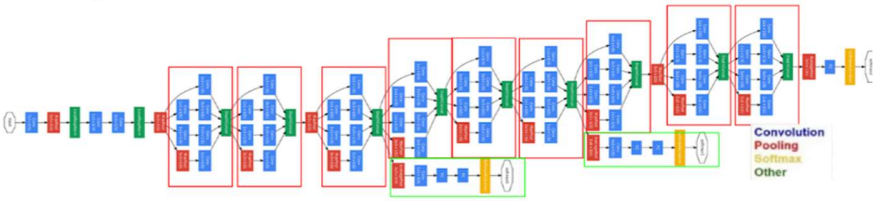


Fig. 2. Architecture detail of GooLeNet

The second pre-trained model I choose is **DenseNet**.

Originally, ResNet can train deeper CNN models, thereby achieving higher accuracy. The core of the ResNet model is to train a deeper CNN network by establishing "shortcuts, skip connections" between the front layer and the back layer (Fig 3(a)), Through the concept of shortcut, if the gradient disappears in a certain layer due to the deep learning depth, it will not affect other weights, which improves the backpropagation of gradients during the training.

The basic idea of the DenseNet module is the same as that of ResNet, but it establishes a *dense* connection between all previous layers and the sequential layers in a feed-forward manner. A traditional convolutional network with L layers has L connections, and there are L*(L + 1)/2 direct connections between each layer and the following layer ((Fig 3(b))). Compared to ResNet, this is a dense connection. Moreover, DenseNet directly concat feature maps from different layers, which can realize feature reuse and improve efficiency.

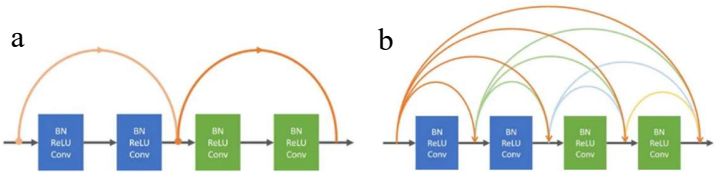


Fig. 3. Schematic of Architecture idea of (a) ResNet, (b) DenseNet

In general, the advantages of DenseNet are mainly reflected in the following aspects:

(1) DenseNet improves the back propagation of gradients, making the network easier to train. Since each layer can directly access the final error signal, implicit "deep supervision" is achieved;
(2) The parameters are smaller and the calculation is more efficient. Since DenseNet uses concat features to implement short-circuit connections, achieve feature reuse.
(3) Adopt a smaller growth rate, so the unique feature map of each layer is relatively small.
(4) Due to feature reuse, the final classifier uses low-level features.

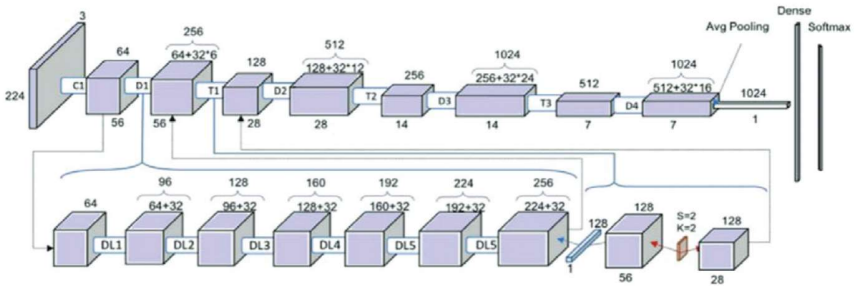


Fig. 4. The architecture of DenseNet121 with Dense block (D) Transition blocks (T) and Dense Layers (DL).

The architecture of DenseNet121 is shown in Fig 4. In Dense block (D), the feature maps of each layer are of the same size and can be connected in the channel dimension. For the Transition layer (T), it mainly connects two adjacent Dense blocks. In D, the structure contains BN+ReLU+1x1 Conv+BN+ReLU+3x3 Conv, and in T, there are BN+ReLU+1x1 Conv+2x2 AvgPooling. Note that he output form of DenseNet is classifier layer but not fully connected layers, so the code in Lab 5 should be modified to be what it like in Fig 5.

```
#num_ftrs = model.fc.in_features
num_ftrs = model.classifier.in_features
```

```
# model.fc =
model.classifier
```

Fig. 5. Output feature for DenseNet (white code) and GooLeNet (green code)

The performances of the above two pre-trained module for the training set in course are shown in the Fig 6.

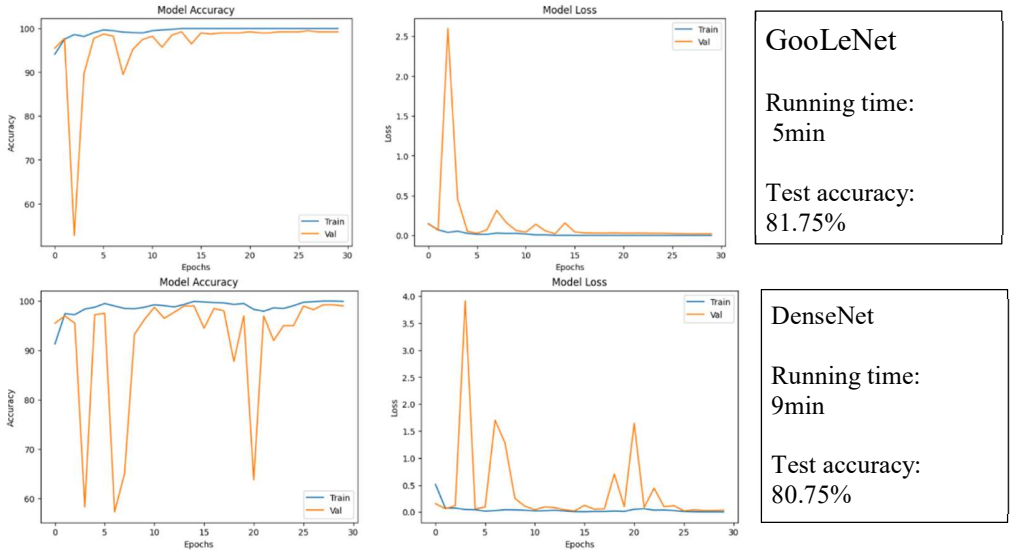


Fig 6. Performance comparison between GooLeNet and DenseNet

Both the pre-trained modules converge rapidly compared to the CNN/ANN we build before, showing that the weights of the two modules are well-trained. However, due to the module complexity, the running times increase. Although test set accuracies are still ~80%, I just learn from the course video, the accuracy of training set is close to that of validation set, meaning that the module is well-trained, the gap between test set is because of data distribution. The test set is from another dataset, which makes the trained-weight cannot fit well to it.

2. Fine-tuning the ConvNet

From GPT, it told me that the fine-tuning ConvNet has the following sequence:

(1) Load Pre-trained Model: done in Lab 5, red box in Fig 7
(2) Remove original fully connected layer: yellow box in Fig 7
(3) Freeze Convolutional Layers (Fix-Feature, task C): provide by TA, blue box in Fig 7
(4) Modify last fully connected for new task: purple box in Figure 7

Note that the feature of last layer from these two models (shown in Fig 5), so there are some differences of the function name for these two models. The differences are shown in orange, yellow & purple boxes in Fig 7.

```
# Load pre-trained model
#model = models.googlenet(weights='IMAGENET1K_V1')
model = models.densenet121(weights='IMAGENET1K_V1')

# Model output feature
#num_ftrs = model.fc.in_features        # for googlenet
num_ftrs = model.classifier.in_features # for densenet

# Remove fully connected layers
#model.fc = nn.Sequential()             # for googlenet
model.classifier = nn.Sequential()      # for densenet

# ConvNet as fixed feature extractor (freeze parameters)
for param in model.parameters():
        param.requires_grad = False

# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to ``nn.Linear(num_ftrs, len(class_names))``.
#model.fc = nn.Sequential(              # for googlenet
model.classifier = nn.Sequential(       # for densenet
        nn.Linear(num_ftrs, 256),
        nn.ReLU(),
        nn.Dropout(0.5),
        nn.Linear(256, 2)
)
print(model)
model = model.cuda()
```

Fig 7. Model code for fine-tuning ConvNet

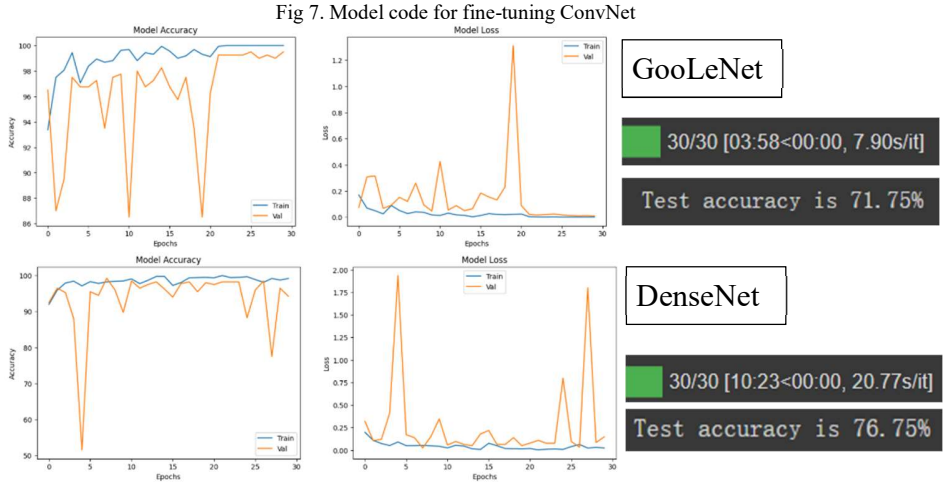


Fig 8. Model performance for Fine-Tuning ConvNet

The Fine-tuning the ConvNet performances are shown in Fig 8. The results seem to be close to that of model without fine-tuning. Compare the curves and running times in Fig 6 & 8, I find that all of them converge in the first few iterations. There is no improvement in running time since fine-tuning ConvNet involves training the entire model (self-define fully connected layer), including the pre-trained convolutional layers, on a new dataset.

3. ConvNet as Fixed Feature Extractor

The code is like what it shows in Fig 7. The only difference is to remember run the freeze parameters code (blue box in Fig 7). Note that the Fine-tuning ConvNet do not run the freeze parameters.

About their performance, in Fig 9, both the test accuracies of model boost to ~85%. The highest test accuracy since I start learning the deep learning. Moreover, from the accuracy and loss curves, these models seem to be unconverging. In other word, if I increase the epoch, the test accuracies have the opportunities to increase as well. Last, the running time decrease dramatically compared to Fig 6. The reason should be the freeze parameter function. In my whole training process, all the parameters of pre-trained model are fixed, so the only improving parameters are in the fully connected layers I add at the last; therefore, the entire calculation will be simplified and the running time will reduce a lot.

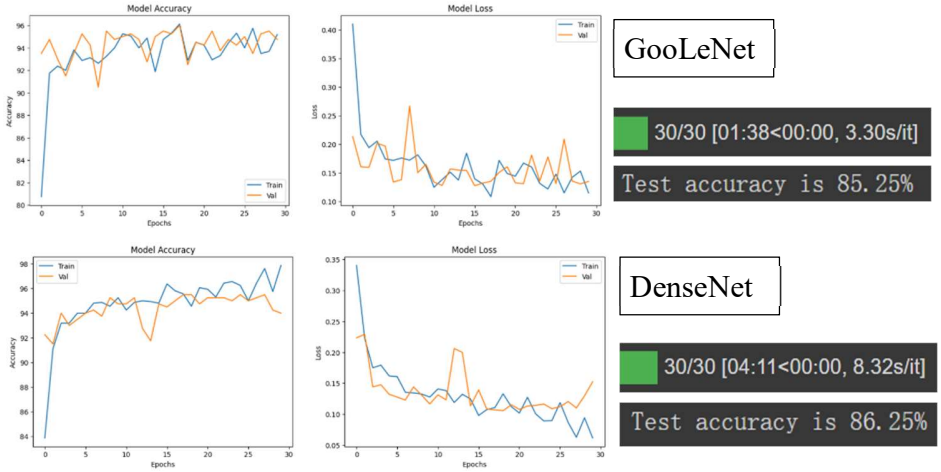


Fig 9. Model performance for Fixed Feature ConvNet

4. Comparison and Analysis

Following table 1 is the comparison of the two approaches when adapting a pre-trained model to a new task.

Table 1. Comparison between

	Fine-tuning the ConvNet	ConvNet as Fixed Feature Extractor
Approach	Train the entire model, including both the convolutional base and the fully connected layers.	Freeze the weights of the pre-trained modules, and train only the final fully connected layer on the new dataset.
Pros	1. Allows the model to adapt its learned features to the specific characteristics of the new task. 2. Can capture task-specific patterns in both lower and higher-level features.	1. Faster training since only a small portion of the model is being updated. 2. Less risk of overfitting, especially with a limited amount of new data.
Cons	1. Requires more computational resources and time compared to using a fixed feature extractor. 2. May risk overfitting on a smaller dataset.	1. May not capture task-specific patterns in lower-level features as effectively as fine-tuning 2. Limited adaptation to the specific characteristics of the new task.

GPT claim that when the dataset is relatively large, computational resources are sufficient and the new task is significantly different from the original pre-training task, it suggests us to use Fine-tuning ConvNet; however, when the data amount is limited, computational resources are limited, and the lower-level features are likely to be relevant to the new task, Fixed Feature Extractor ConvNet is adorable.

Since our data amount is only 2000 and 400 for training/validation and test set, and we can only use T4 GPU from the google server, Fixed Feature Extractor ConvNet is the better choice for us. From the performances in this report, they support this argument in the perspective of running time, test accuracy and the convergence. It seems that Fixed Feature Extractor ConvNet can still improve its performance if the epoch is larger due to the decreasing loss and increasing accuracy when the model is trained at 30th iteration.

5. Test Dataset Analysis

I just review all the course video, I found that the key point of challenges in enhancing the performance on the test dataset is because of the dataset is obtained from different distributions. The following Fig 10 can explain the accuracy gap between validation and test set is not due to the overfitting from the trained model. In all my trained model, the training and validation set all get >95% accuracy, so there should be no avoidable error (label versus training set) & variance (training set versus validation set), which means that the model is well-trained in the first 1600 dataset and be validate by the following 400 un-trained images. Therefore, the low accuracy of test set is because of data mismatch. From the code in Lab 5, it is obviously that the training & validation sets are downloaded from a dataset, and test set is downloaded from another one.

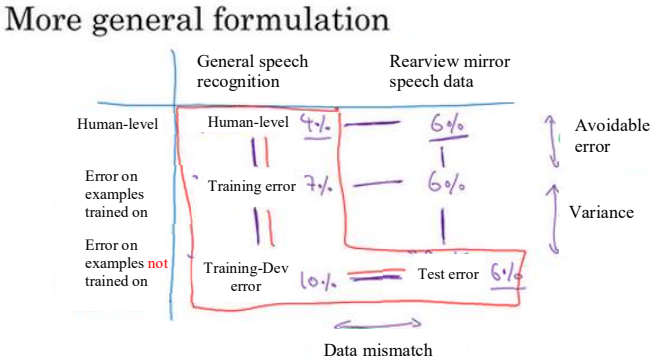


Fig 10. Table illustrate Bias and Variance With Mismatched Data

References

1. Gaurav Jee, Harshvardhan Gm, Mahendra Kumar Gourisaria , “Efficacy Determination of Various Base Networks in Single Shot Detector for Automatic Mask Localisation in a Post COVID Setup”
2. 快速理解 Pre-Trained Model、Transfer Learning 之間差異，並且實作 PyTorch 提供的 Pre-Trained Model
3. Christian Szegedy, Wei Liu, Yangqing Jia,” Going Deeper with Convolutions”
4. 卷積神經網路(Convolutional neural network, CNN): 1×1 卷積計算在做什麼 | by Tommy Huang | Medium
5. https://www.youtube.com/watch?v=2BH49JG_sTs&list=PLkDaE6sCZn6E7jZ9sN_xHwSHOdjUxUW_b&index=17