Multi Cancer Dataset

Group 16
林宥騰、黃思樺、林哲維、蕭靖宸

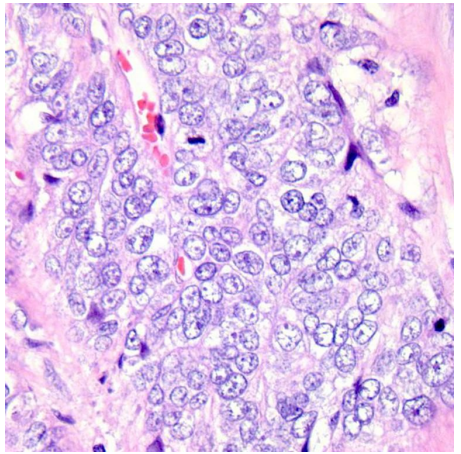
Outline

- Dataset
  - data size and structure
  - subset, 8 kinds of cancer
- Model
  - Environment
  - Resnet
  - Subset Result
- Prevent Overfitting
  - K-Fold
  - Dropout
  - Elastic Net
  - Simple Model
  - Mix

DataSet - Overview

- All datas are pictures. With 8 main cancer classes and 26 subclasses. Cancer's subclasses like early-stage, pre-stage etc. Different from each other.

- 8 main cancer contain: Acute Lymphocytic Leukemia (ALL)、Brain Cancer、Breast Cancer、Cervical Cancer、Kidney Cancer、Lung and Colon Cancer、Lymphoma、Oral Cancer



Sample picture of
Lung and Colon Cancer

DataSet - Overview

	Label 1	Label 2	Label 3	Label 4	Label 5
ALL	Benign	Early	Pre	Pro	
Brain Cancer	Glioma	Meningioma	Pituitary Tumor		
Breast Cancer	Benign	Malignant			
Cervical Cancer	Dyskeratotic	Koilocytotic	Metaplastic	Parabasal	Superficial-Intermediate
Kidney Cancer	Normal	Tumor			

DataSet - Overview

	Label 1	Label 2	Label 3	Label 4	Label 5
Lung and Colon Cancer	Colon Adenocarcinoma	Colon Benign Tissue	Lung Adenocarcinoma	Lung Benign Tissue	Lung Squamous Cell Carcinoma
Lymphoma	Chronic Lymphocytic Leukemia	Follicular Lymphoma	Mantle Cell Lymphoma		
Oral Cancer	Normal	Oral Squamous Cell Carcinoma			

DataSet - Challenges

- Large dataset:
  Since all data are picture, and there are about 130K jpg files in total.
  Total size is 8.62G. We spent 10 minutes to load all dataset with using kagglehub in Colab
- Due to the problem, we run **one subset each time**, and based on the label of the subset for classification task.

DataSet - Data Preprocessing

- Take 5000 data for training, 2000 data for validation and 2000 for testing

```
train_dataset = MultiCancerDataset(train_images[:5000], train_labels[:5000])
valid_dataset = MultiCancerDataset(train_images[5000:min(7000, len(train_images))], train_labels[5000:min(7000, len(train_images))])
test_dataset = MultiCancerDataset(test_images[:2000], test_labels[:2000])
```

- For each subset we calculate mean、std and normalize respectively

```
mean: tensor([0.1575, 0.1575, 0.1575]), std: tensor([0.2441, 0.2441, 0.2441])
Dataset build
```

- ensured data label is balance

```
在 train_data 中的類別統計:
總類別數: 3

各類別統計:
類別 brain_tumor: 1623 筆
類別 brain_menin: 1692 筆
類別 brain_glioma: 1685 筆
```

```
在 valid_data 中的類別統計:
總類別數: 3

各類別統計:
類別 brain_menin: 679 筆
類別 brain_tumor: 638 筆
類別 brain_glioma: 683 筆
```

```
在 test_data 中的類別統計:
總類別數: 3

各類別統計:
類別 brain_tumor: 687 筆
類別 brain_menin: 655 筆
類別 brain_glioma: 658 筆
```

Model - Environment

- We use Google Colab as our environment
- And use Pytorch to build out model
- Here are problems we encounter during training
  - Batch size
    - In Medic dataset, training picture size cannot be too small
    - Resize picture to (224, 224)
    - Batch size for training is constraint to 32
  - Limit of calculate source
    - We used free version, the calculate source is limited
    - 40 epochs for each training
    - Each epochs takes 4 minutes.

Model - Resnet

- ResNet was proposed by Microsoft Research in 2015.
- Residual Learning:
  The core innovation of ResNet is "shortcut connections."
- Advantages:
  - Solves the vanishing gradient problem in deep networks
  - Enables training of much deeper networks (hundreds of layers or more)
  - Provides better feature extraction capabilities

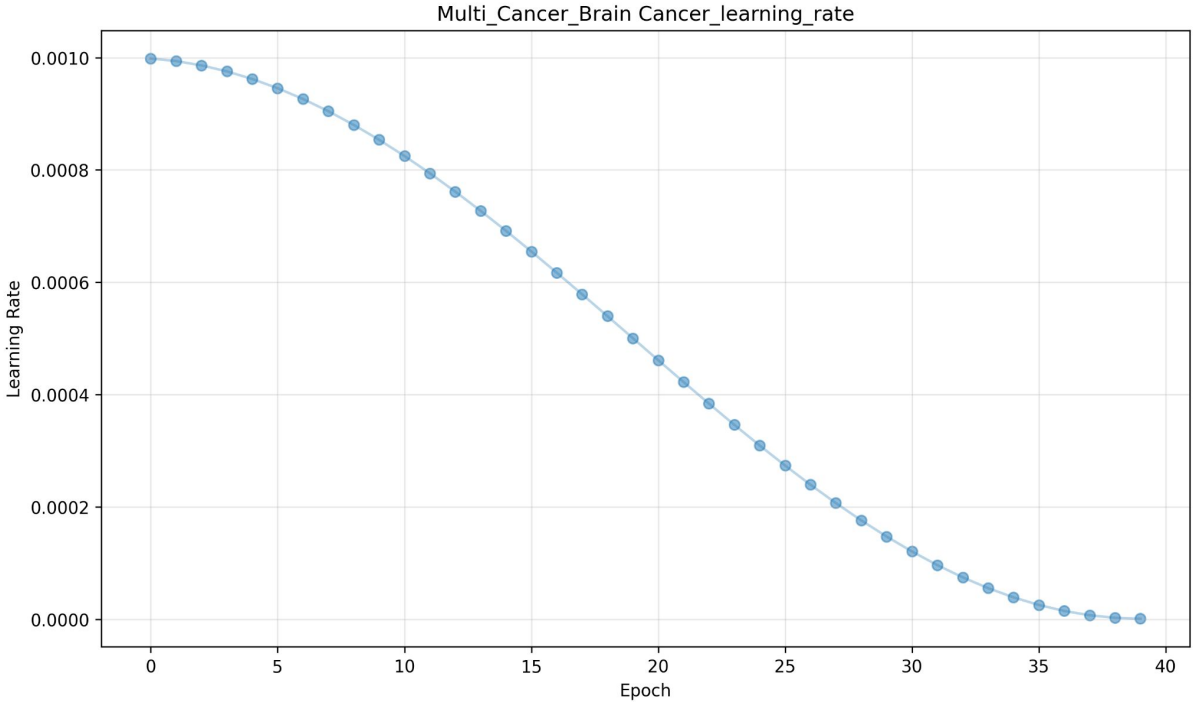
Model - Resnet

```
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(negative_slope=0.01, inplace=True)
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
                nn.BatchNorm2d(out_channels)
            )

        self.relu = nn.LeakyReLU(negative_slope=0.01, inplace=True)

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out += self.shortcut(x)
        out = self.relu(out)
        return out
```

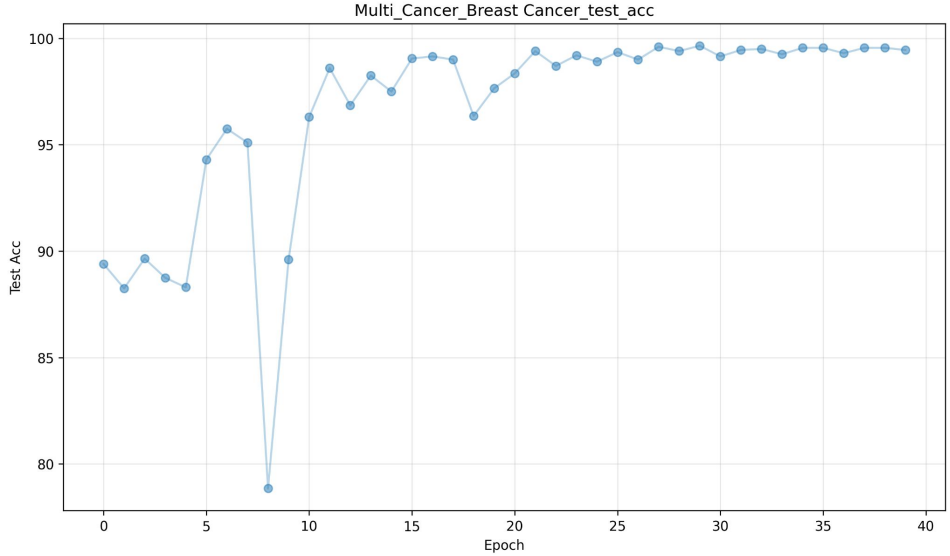
Model - Resnet

```
model = nn.Sequential(
        # Head
        nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
        nn.BatchNorm2d(64),
        nn.LeakyReLU(negative_slope=0.01, inplace=True),

        # Backbone-0 (64 -> 64)
        BasicBlock(64, 64),
        BasicBlock(64, 64),

        # Backbone-1 (64 -> 128)
        BasicBlock(64, 128, stride=2),
        BasicBlock(128, 128),

        # Backbone-2 (128 -> 256)
        BasicBlock(128, 256, stride=2),
        BasicBlock(256, 256),

        # Backbone-3 (256 -> 512)
        BasicBlock(256, 512, stride=2),
        BasicBlock(512, 512),
        nn.AdaptiveAvgPool2d((1, 1)),

        # Predictor
        nn.Flatten(),
        nn.Linear(512, n_classes)
)
```

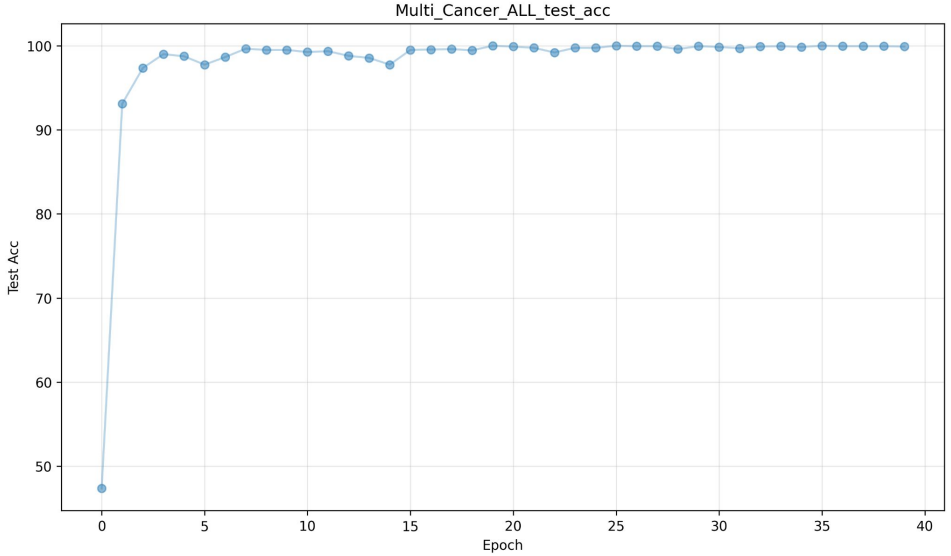
Model - Learning Rate



Model - Result

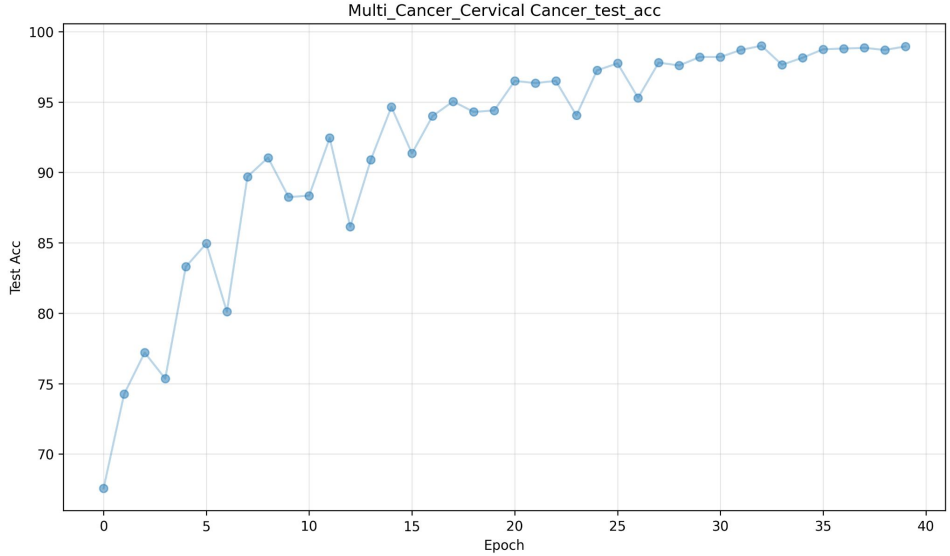


Breast Cancer

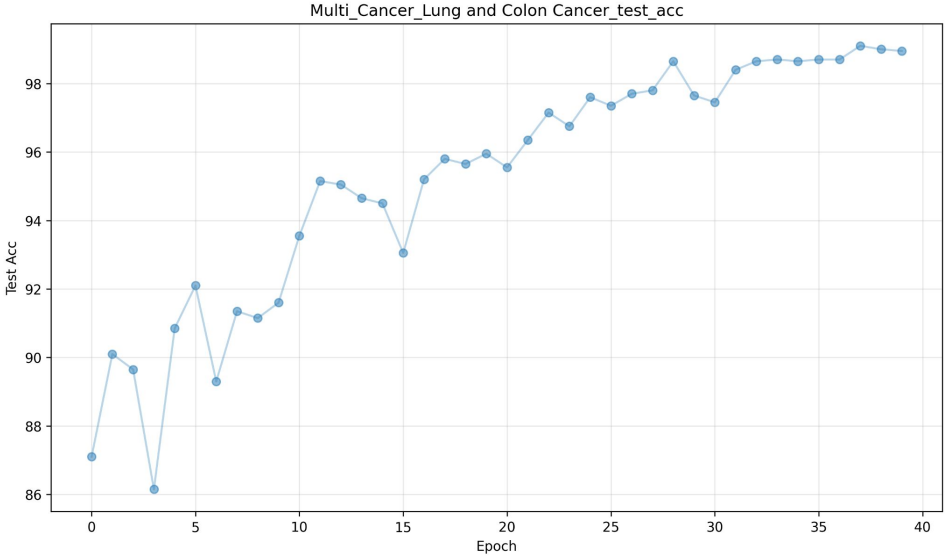


ALL

Model - Result

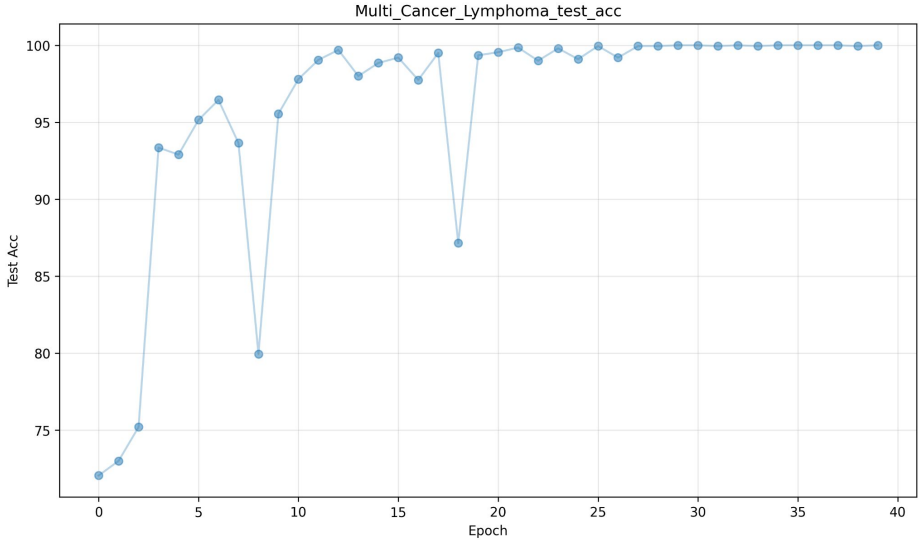


Cervical Cancer

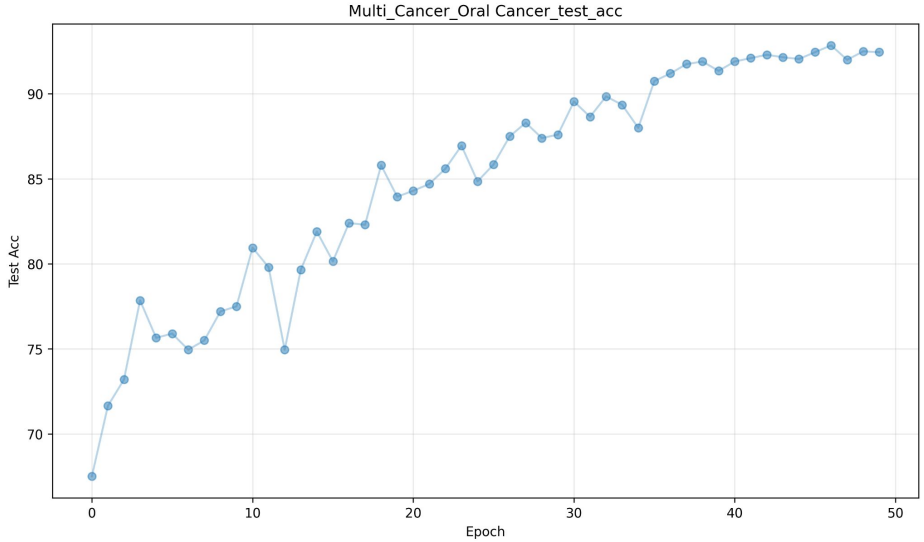


Lung and Colon Cancer

Model - Result

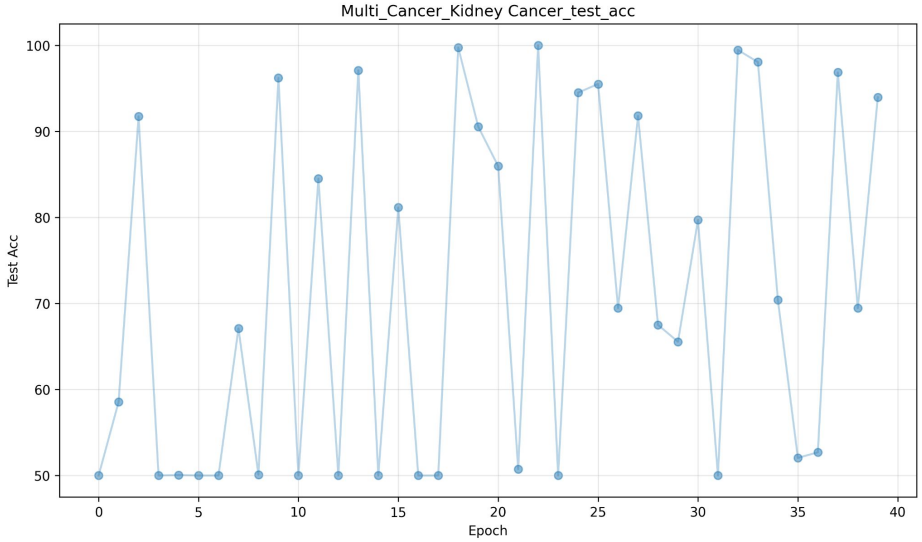


Lymphoma

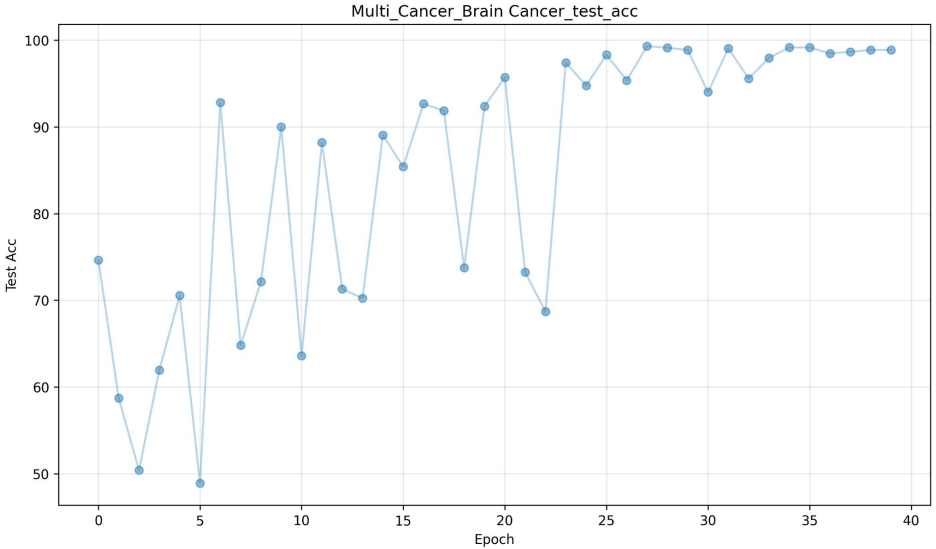


Oral Cancer

Model - Result

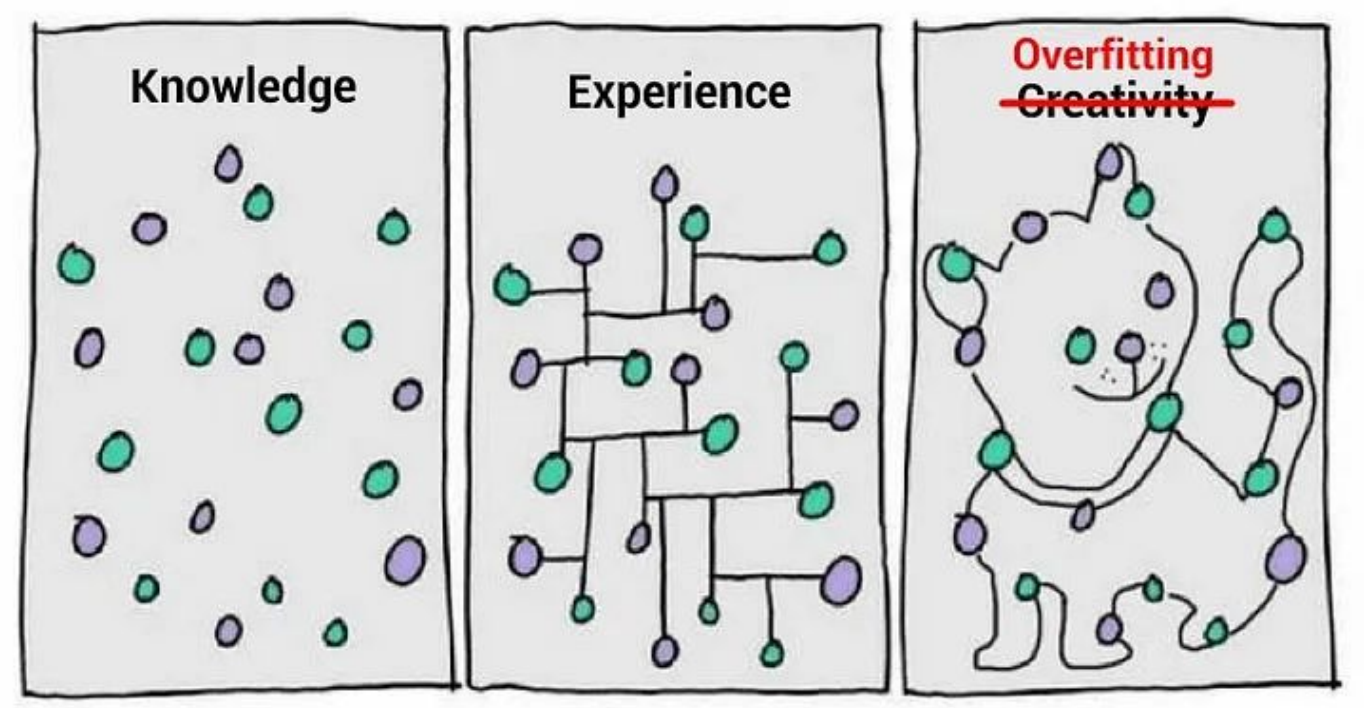


Kidney Cancer



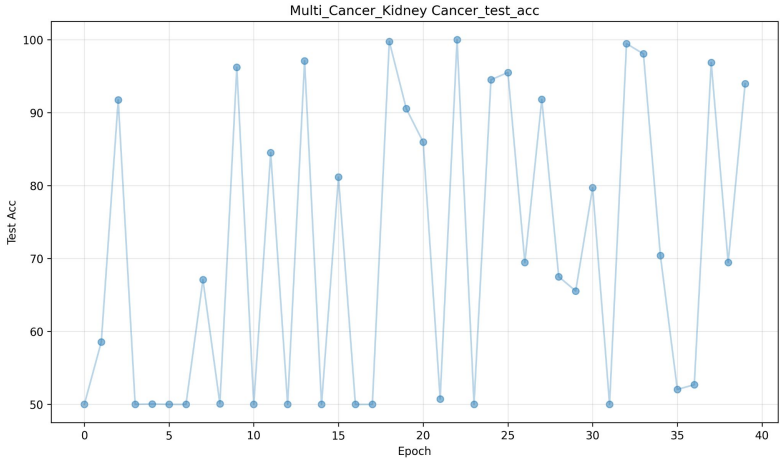
Brain Cancer

Challenges encountered in training



Challenges encountered in training

- Overfitting
  - Obviously, Kidney and Brain cancer has overfitting problem by observed the plot.
  - **We choose Brain Cancer to improve.** Since the Kidney Cancer looks hopeless.
  - Then we try 4 method to solve it
    dropout, using simple model, K-Fold and Elastic Net.

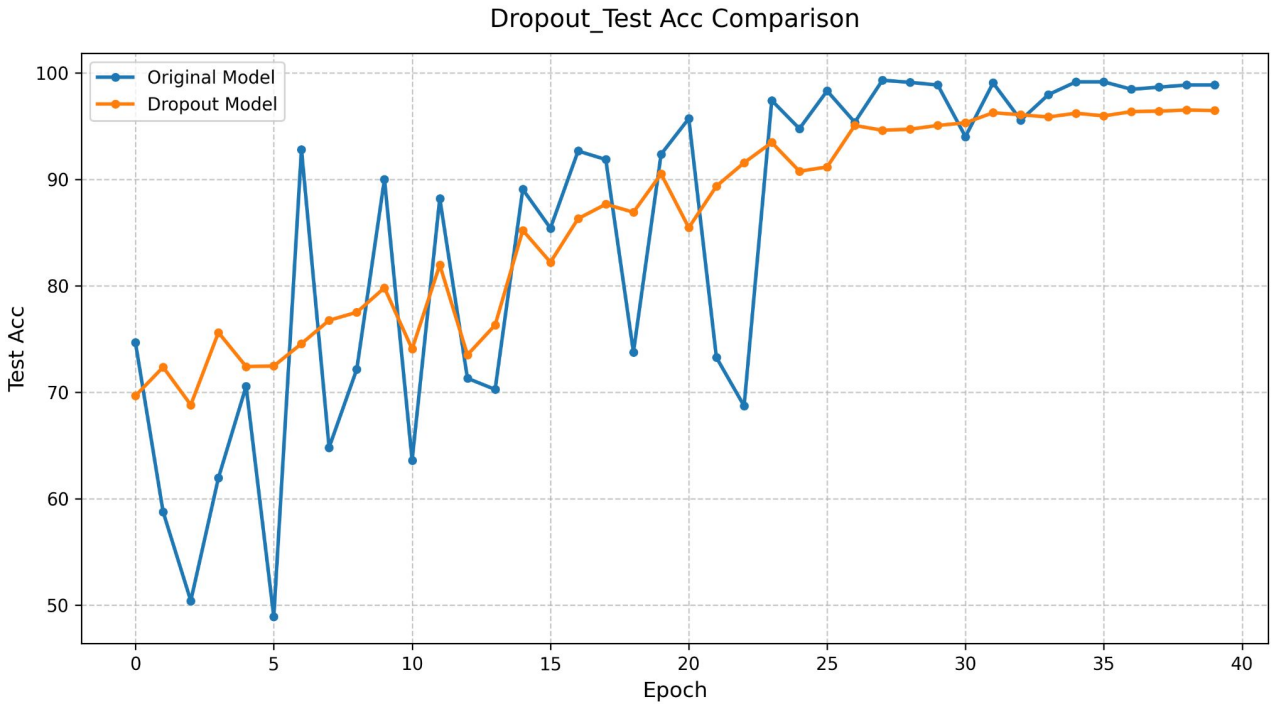


Prevent Overfitting - Dropout

- Add dropout layer after each layer,
- The parameter of dropout gradually increases. From p=0.1 to p=0.5

```
model = nn.Sequential(
        # Head
        nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
        nn.BatchNorm2d(64),
        nn.LeakyReLU(negative_slope=0.01, inplace=True),
        nn.Dropout2d(0.1),

        # Backbone-0 (64 -> 64)
        BasicBlock(64, 64),
        BasicBlock(64, 64),
        nn.Dropout2d(0.2),

        # Backbone-1 (64 -> 128)
        BasicBlock(64, 128, stride=2),
        BasicBlock(128, 128),
        nn.Dropout2d(0.3),

        # Backbone-2 (128 -> 256)
        BasicBlock(128, 256, stride=2),
        BasicBlock(256, 256),
        nn.Dropout2d(0.4),

        # Backbone-3 (256 -> 512)
        BasicBlock(256, 512, stride=2),
        BasicBlock(512, 512),
        nn.Dropout2d(0.5),

        nn.AdaptiveAvgPool2d((1, 1)),

        # Predictor
        nn.Flatten(),
        nn.Dropout(0.5),

        nn.Linear(512, n_classes)
)
```

Prevent Overfitting - Dropout



Prevent Overfitting - Simple Model

- Reduce the complexity of origin model
- Origin: 3 -> 64 -> 128 -> 256 -> 512 -> n_classes
- Simple: 3 -> 32 -> 64 -> 128 -> 256 -> n_classes

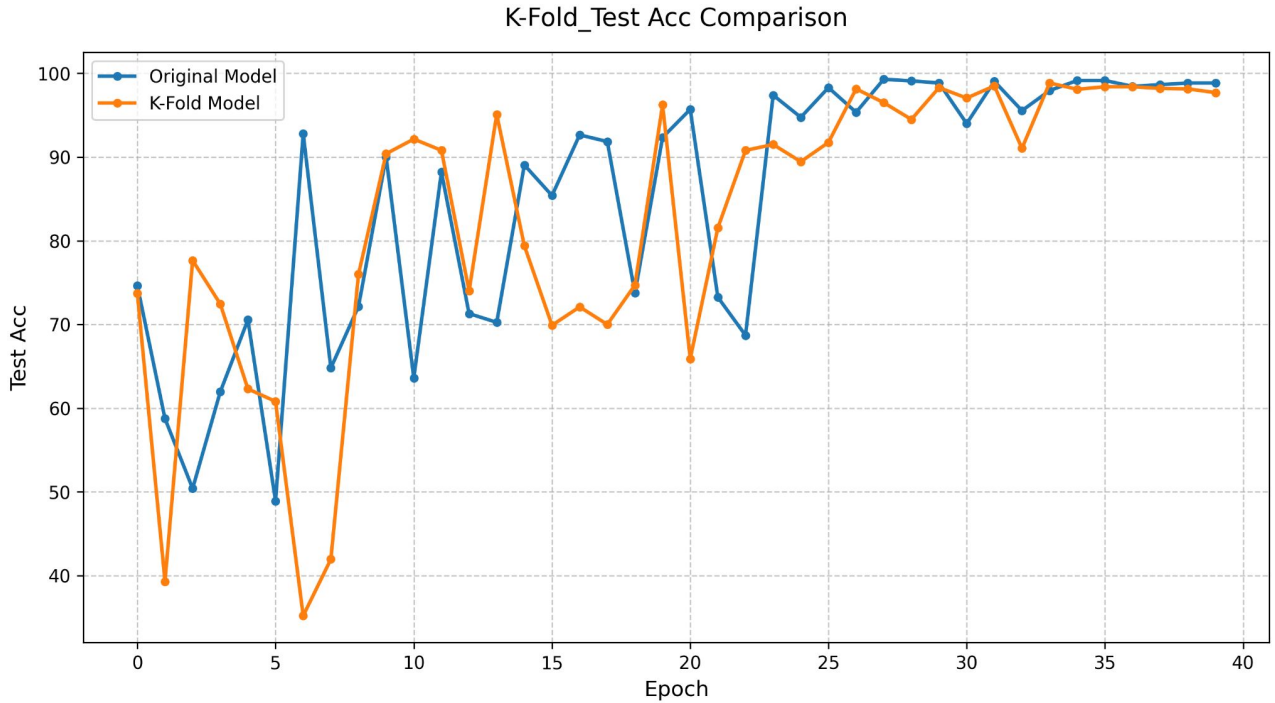
```
model = nn.Sequential(
        # Head
        nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1, bias=False),    # 降低初始通道數從64到32
        nn.BatchNorm2d(32),
        nn.LeakyReLU(negative_slope=0.01, inplace=True),

        # Backbone-0 (32 -> 64)
        BasicBlock(32, 64, stride=2),    # 直接在第一個Block進行降採樣

        # Backbone-1 (64 -> 128)
        BasicBlock(64, 128, stride=2),

        # Backbone-2 (128 -> 256)
        BasicBlock(128, 256, stride=2),
        BasicBlock(256, 256),    # 保留一個額外的Block在最後階段

        nn.AdaptiveAvgPool2d((1, 1)),

        # Predictor
        nn.Flatten(),
        nn.Linear(256, n_classes)    # 降低最終特徵維度從512到256
)
```

Prevent Overfitting - Simple Model

Prevent Overfitting - K-Fold

- Split train dataset into K folds.
- Model train on K-1 folds, validation on 1 fold.
- Repeat K times, each fold is used as the validation set once.
- K: 5
- Epoch: 8

Prevent Overfitting - K-Fold

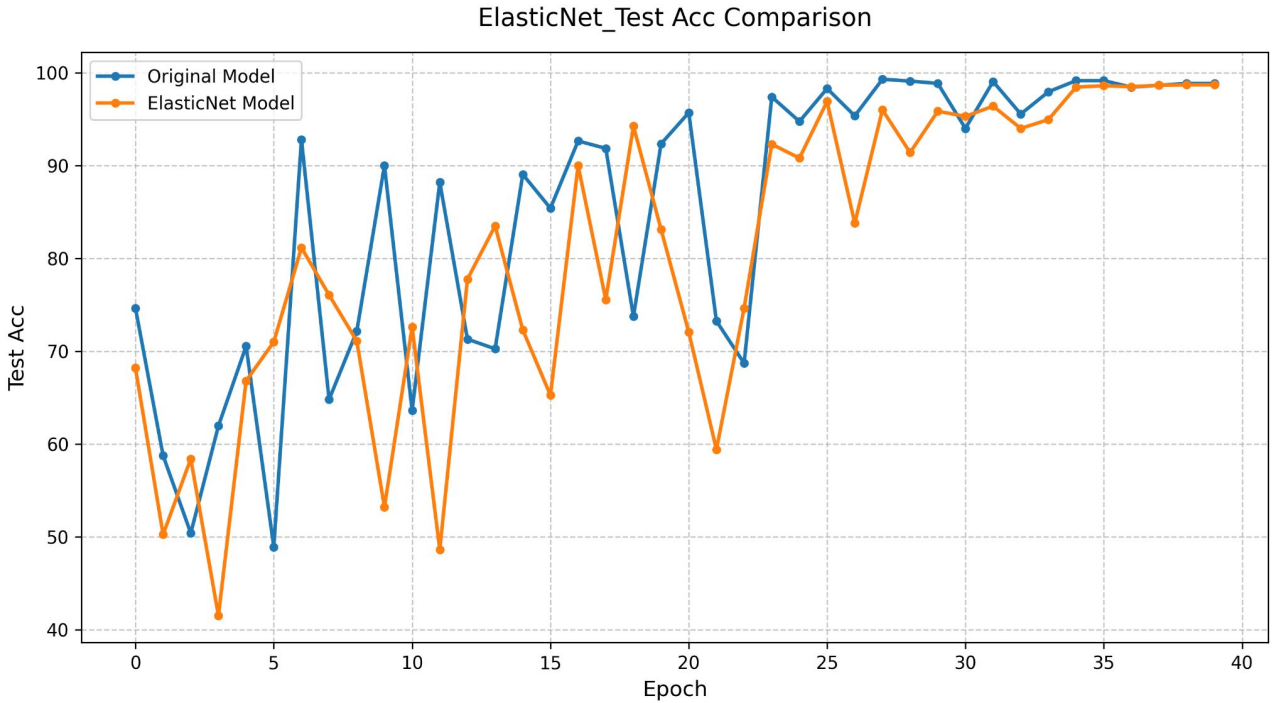


Prevent Overfitting - Elastic Net

- Combines L1 and L2 regularization
- Set weight: 1e-5 for L1, 1e-4 for L2

```
def elastic_net_loss(original_loss, model, lambda1=1e-5, lambda2=1e-4):
    l1_reg = torch.tensor(0., device='cuda')    # 初始化 L1 正則化
    l2_reg = torch.tensor(0., device='cuda')    # 初始化 L2 正則化
    for param in model.parameters():
        l1_reg += torch.norm(param, 1)    # L1 正則化
        l2_reg += torch.norm(param, 2) ** 2    # L2 正則化
    return original_loss + lambda1 * l1_reg + lambda2 * l2_reg    # 總損失
```

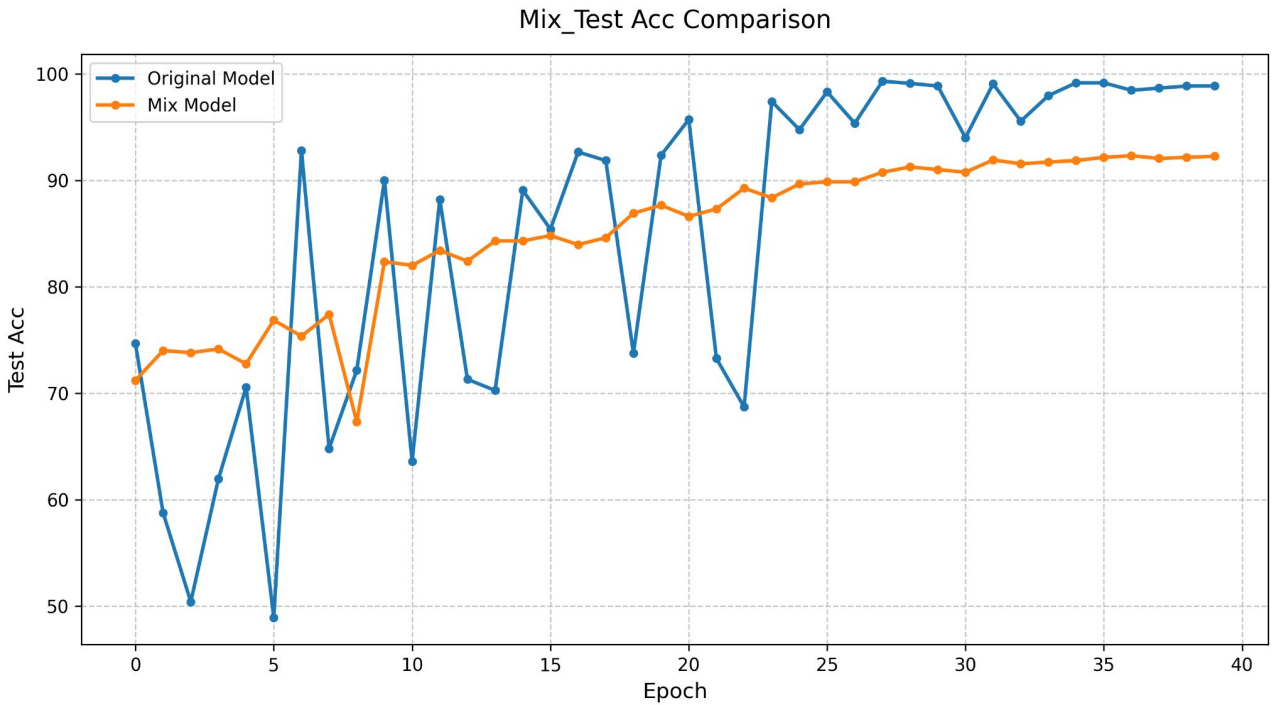
Prevent Overfitting - Elastic Net



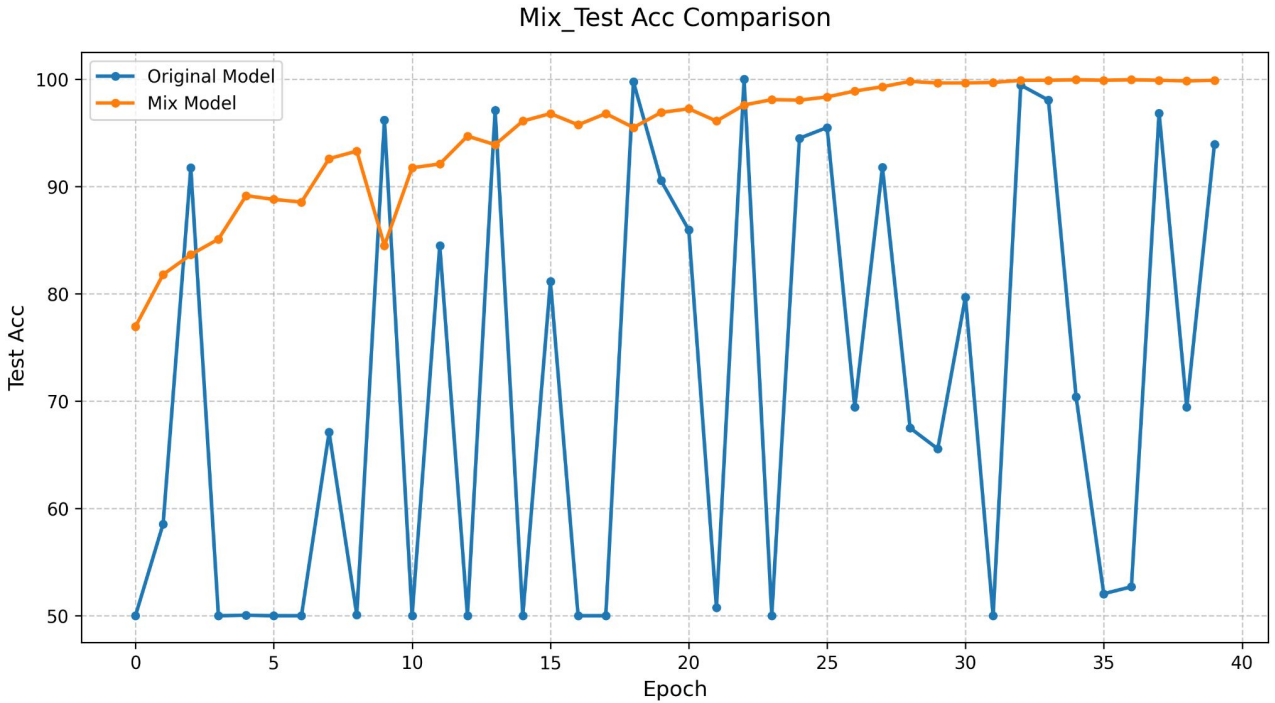
Prevent Overfitting - Mix

JUST MIX ALL METHOD TOGATHER

Prevent Overfitting - Mix



Kidney Miracle



References

- https://github.com/minyaho/SCPL/tree/main
- https://www.youtube.com/watch?v=zPEsBC4dgwY

Code

- Colab link
  Mix Version

THANKS FOR LISTENING