Salaki Reynaldo Joshua
KANGWON NATIONAL UNIVERSITY

DIABETIC RETINOPATHY: AN AUTOMATED ANALYSIS USING RESTNET50

Lecturer:
Prof. Park Sanguk

2023

SALAKI REYNALDO JOSHUA

Ph.D Student

Department of Electronics,Information and Communication Engineering

Description of Mid Exam

Collect data on your field of study and implement an artificial intelligence prediction model by utilizing the artificial intelligence model (classification or regression model) you have learned in this class.

- Collecting data in your research field (Example: hydrogen data, renewable energy data, etc.)
- Perform data preprocessing
- Building a deep learning model
- Check overfitting by graph
- Automatic interruption of learning
- Model training and prediction
- It is okay to implement it with a program other than Colab

Due Date : 2023-05-09 00:00

Global Goals

TABLE OF CONTENTS

Introduction

DIABETIC RETINOPATHY

Diabetic retinopathy (DR) is the major ocular complication of diabetes mellitus, and is a problem with significant global health impact. Major advances in diagnostics, technology and treatment have already revolutionized how we manage DR in the early part of the 21st century. For example, the accessibility of imaging with optical coherence tomography, and the development of antivascular endothelial growth factor (VEGF) treatment are just some of the landmark developments that have shaped the DR landscape over the last few decades. Yet, there are still more exciting advances being made. Looking forward to 2030, many of these ongoing developments are likely to further transform the field. First, epidemiologic projections show that the global burden of DR is not only increasing, but also shifting from high-income countries towards middle- and low-income areas [1]

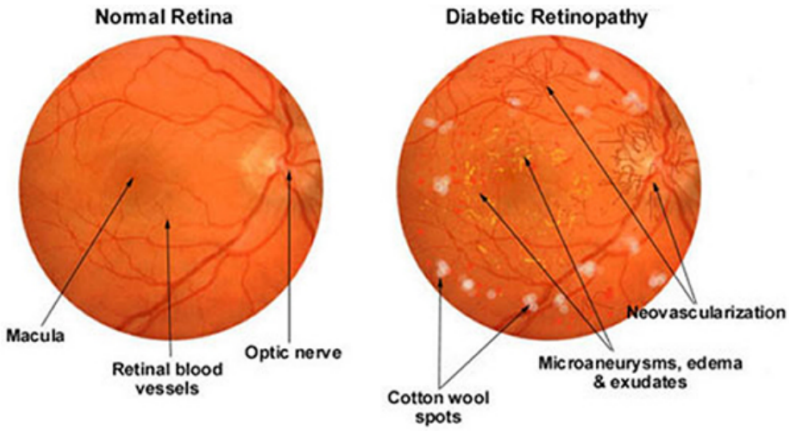

Figure 1. A healthy patient and a patient with diabetic retinopathy as viewed by fundus photography [2]

An automated tool for grading severity of diabetic retinopathy would be very useful for accerelating detection and treatment. Recently, there have been a number of attempts to utilize deep learning to diagnose DR and automatically grade diabetic retinopathy. This includes this competition and work by Google. Even one deep-learning based system is FDA approved. Clearly, this dataset and deep learning problem is quite well-characterized [3].

Reference :

[1] Tien-En Tan, and Tien Yin Wong. 2023. Diabetic retinopathy: Looking forward to 2030. Frontiers in Endocrinology. 13:1077669. doi: 10.3389/fendo.2022.1077669

[2] Alexander Rakhlin. 2018. Diabetic Retinopathy detection through integration of Deep Learning classification framework.bioRxiv preprint doi:

[3] Diabetic Retinopathy, ResNet50, Binary, Cropped. https://www.kaggle.com/code/tanlikesmath/diabetic-retinopathy-resnet50-binary-cropped/notebook

Introduction

RESTNET50

ResNet stands for Residual Network and is a specific type of convolutional neural network (CNN) introduced in the 2015 paper “Deep Residual Learning for Image Recognition” by He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. CNNs are commonly used to power computer vision applications. ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks [4].

The original ResNet architecture was ResNet-34, which comprised 34 weighted layers. It provided a novel way to add more convolutional layers to a CNN, without running into the vanishing gradient problem, using the concept of shortcut connections. A shortcut connection “skips over” some layers, converting a regular network to a residual network.
The regular network was based on the VGG neural networks (VGG-16 and VGG-19)—each convolutional network had a 3×3 filter. However, a ResNet has fewer filters and is less complex than a VGGNet. A 34-layer ResNet can achieve a performance of 3.6 billion FLOPs, and a smaller 18-layer ResNet can achieve 1.8 billion FLOPs, which is significantly faster than a VGG-19 Network with 19.6 billion FLOPs (read more in the ResNet paper, He et, al, 2015).
The ResNet architecture follows two basic design rules. First, the number of filters in each layer is the same depending on the size of the output feature map. Second, if the feature map’s size is halved, it has double the number of filters to maintain the time complexity of each layer [4].

In Figure 2, Left: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Right: a residual network with 34 parameter layers (3.6 billion FLOPs) [5].

Reference :

[4] Datagen. 2023. Resnet50. Link: https://datagen.tech/guides/computer-vision/resnet-50/#:~:text=ResNet%2D50%20is%20a%2050,ResNet%2D50%20Architecture
[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. 2015. Deep Residual Learning for Image Recognition. Link: https://arxiv.org/pdf/1512.03385.pdf

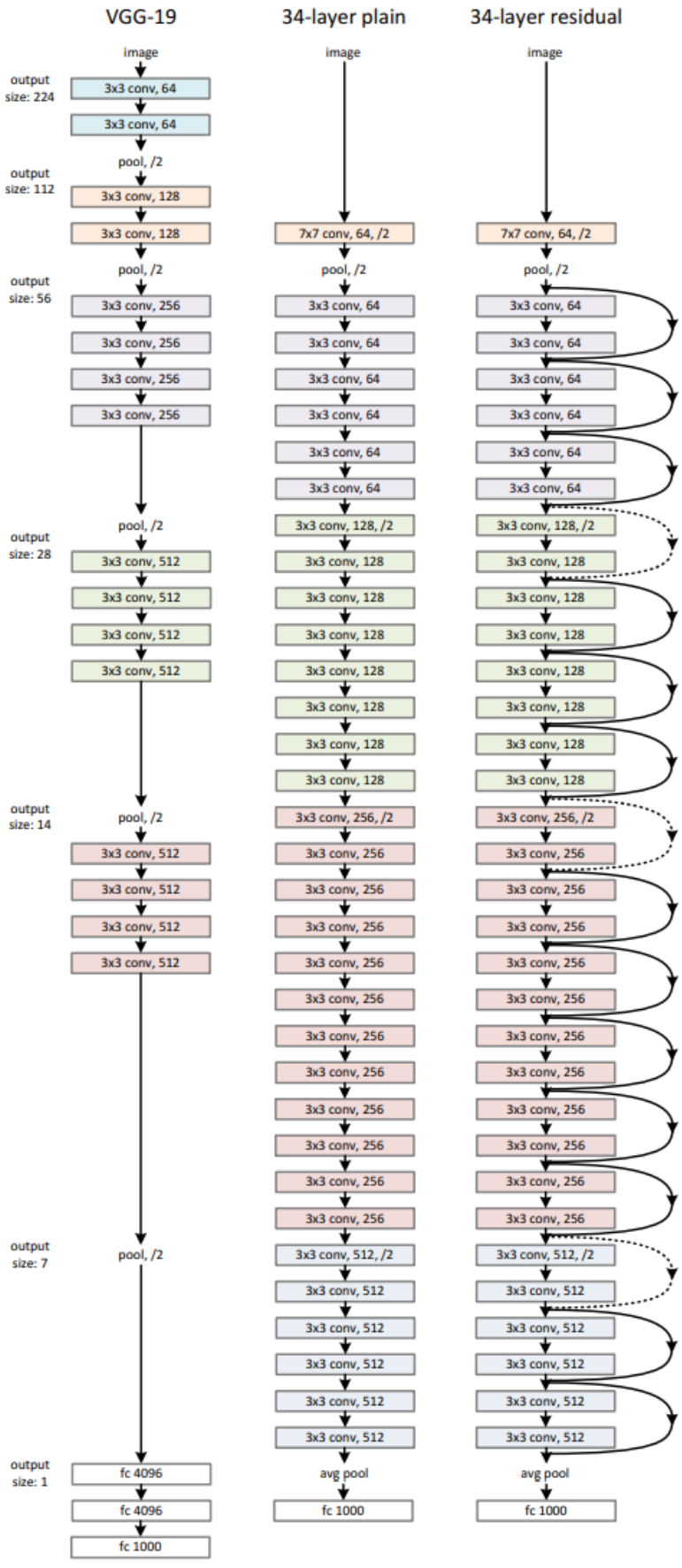

Figure 2. Figure 3. Example network architectures for ImageNet [5]

Section 1

LIBRARY

First step of developing analysis system using python is importing the library

```
import os
files = os.listdir('../input/resized_train/resized_train')
print(len(files))
```

```
35126
```

```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

```
from fastai import *
from fastai.vision import *
import pandas as pd
import matplotlib.pyplot as plt
```

```
print('Make sure cuda is installed:', torch.cuda.is_available())
print('Make sure cudnn is enabled:', torch.backends.cudnn.enabled)
```

```
Make sure cuda is installed: True
Make sure cudnn is enabled: True
```

Figure 3. Import Library

Section 2

READING DATA

Here we will open the dataset with pandas, check distribution of labels, and oversample to reduce imbalance.

Input

```
base_image_dir = os.path.join('..', 'input')
df = pd.read_csv(os.path.join(base_image_dir, 'trainLabels_cropped.csv'))
df['path'] = df['image'].map(lambda x: os.path.join(base_image_dir,'resized_train_cropped/resized_train_cropped','{}.jpeg'.format(x)))
df = df.drop(columns=['image'])
df = df.sample(frac=1).reset_index(drop=True) #shuffle dataframe
df['level'] = (df['level'] > 1).astype(int) # Disease or no disease
df.head(10)
```

Figure 4. Input for Reading Data

Output

	Unnamed: 0	Unnamed: 0.1	level	path
0	3918	3922	0	../input/resized_train_cropped/resized_train_c...
1	30349	30365	0	../input/resized_train_cropped/resized_train_c...
2	29257	29273	0	../input/resized_train_cropped/resized_train_c...
3	12141	12149	0	../input/resized_train_cropped/resized_train_c...
4	31509	31525	0	../input/resized_train_cropped/resized_train_c...
5	16843	16854	0	../input/resized_train_cropped/resized_train_c...
6	5850	5854	0	../input/resized_train_cropped/resized_train_c...
7	29087	29103	0	../input/resized_train_cropped/resized_train_c...
8	29245	29261	0	../input/resized_train_cropped/resized_train_c...
9	10415	10422	0	../input/resized_train_cropped/resized_train_c...

Figure 5. Output for Reading Data

Input

```
df['level'].hist(color="purple", edgecolor= 'black',figsize = (15, 7))
```

Figure 6. Input Histogram for Reading Data

Output

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f137813b7f0>
```

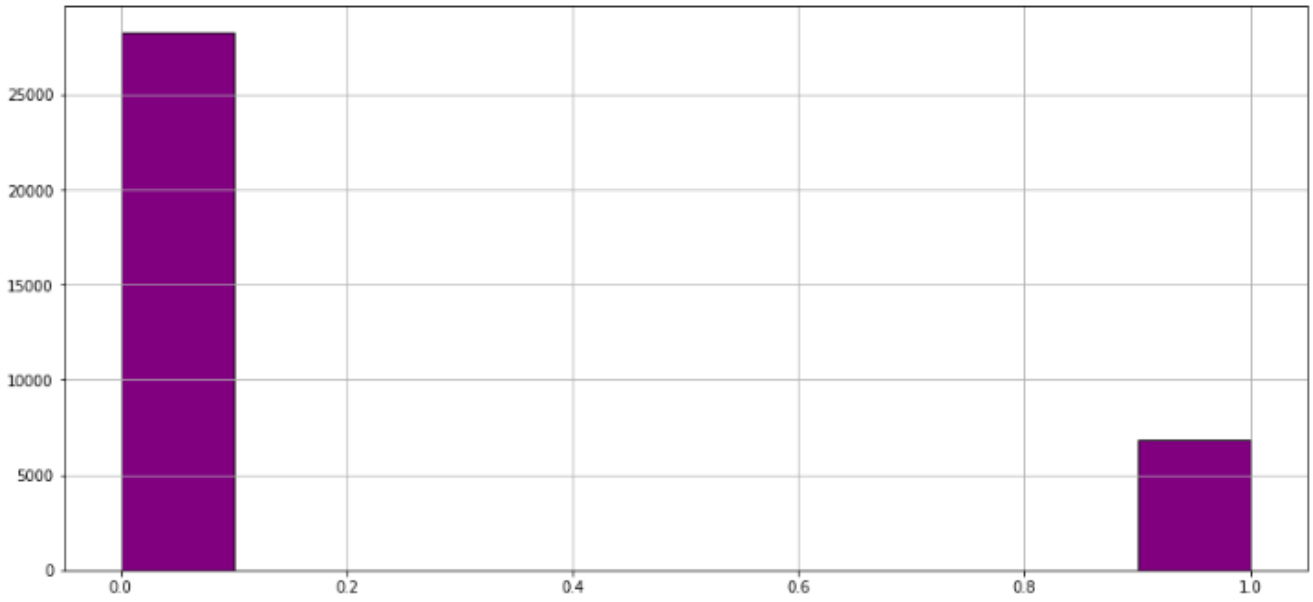

Figure 7. Output Histogram for Reading Data

Input

Here we will perform an 80%/20% split of the dataset, with stratification to keep similar distribution in validation set

```
from sklearn.model_selection import train_test_split
train_df, val_df = train_test_split(df,test_size=0.2)
```

```
train_df['level'].hist(color="purple", edgecolor= 'black',figsize = (15, 7))
len(val_df)
```

```
7022
```

Figure 8. Validation Set Input

Output

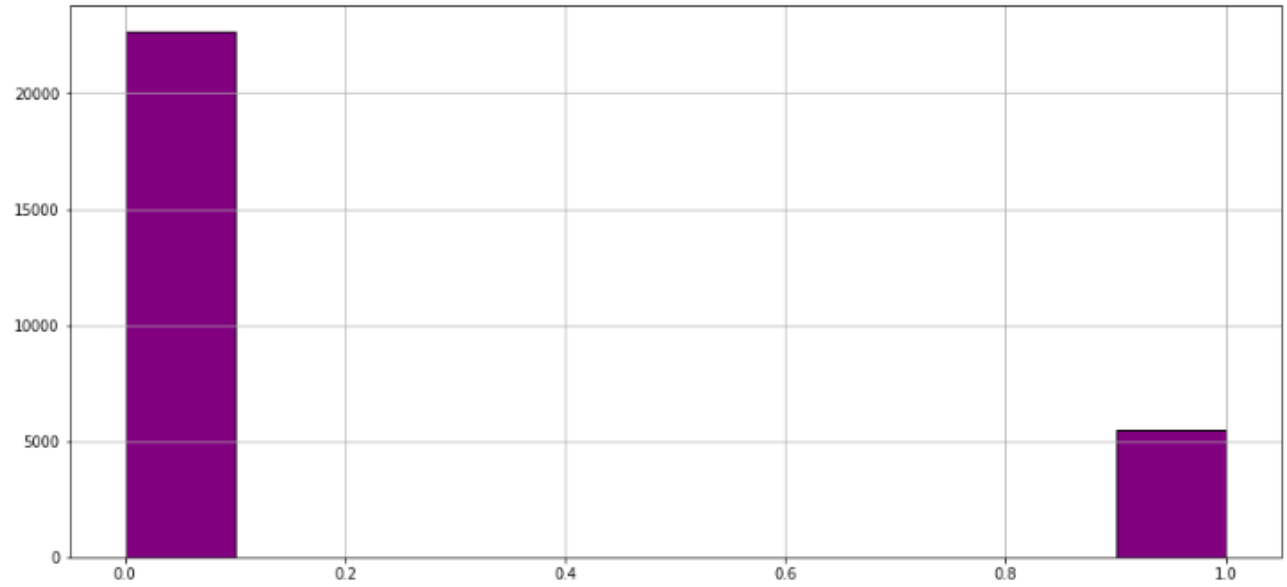

Figure 9. Histogram Validation Set Input

```
df = pd.concat([train_df,val_df]) #beginning of this dataframe is the training set, end is the validation set
len(df)
```

```
35108
```

```
bs =16 #smaller batch size is better for training, but may take longer
sz=512
```

Figure 10. Training Set

Section 3

LOAD DATASET

Here, we will load the dataset into the `ImageItemList` class provided by `fastai`. The fastai library also implements various transforms for data augmentation to improve training. While there are some defaults that I leave intact, I add vertical flipping (`do_flip=True`) as this has been commonly used for this particular problem.

Typically, one would use the `ImageDataBunch` class to load the dataset much easier, but as I will adjust the splitting and add oversampling in future kernels, I have used this customized creation of the DataBunch using the `data_block` API.

Input

```
tfms = get_transforms(do_flip=True,flip_vert=True,max_rotate=360,max_warp=0,max_zoom=1.0,max_lighting=0.1,p_lighting=0.5)
src = (ImageList.from_df(df=df,path='./',cols='path') #get dataset from dataset
        .split_by_idx(range(len(train_df)-1,len(df))) #Splitting the dataset
        .label_from_df(cols='level') #obtain labels from the level column
      )
data= (src.transform(tfms,size=sz,resize_method=ResizeMethod.SQUISH,padding_mode='zeros') #Data augmentation
        .databunch(bs=bs,num_workers=4) #DataBunch
        .normalize(imagenet_stats) #Normalize
       )
```

```
data.show_batch(rows=3, figsize=(12,12))
```

Figure 11. Input Load Dataset

Output

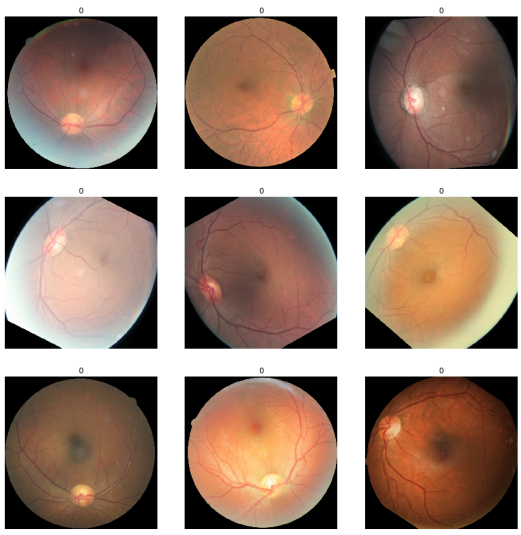

Figure 12. Output Load Dataset

```
print(data.classes)
len(data.classes),data.c
```

```
[0, 1]
(2, 2)
```

Figure 13. Print Data Class

Section 4

TRAINING

We use transfer learning, where we retrain the last layers of a pretrained neural network. I use the ResNet50 architecture trained on the ImageNet dataset, which has been commonly used for pre-training applications in computer vision. Fastai makes it quite simple to create a model and train.

```
import torchvision
from fastai.metrics import *
from fastai.callbacks import *
learn = cnn_learner(data, models.resnet50, wd = 1e-5, metrics = [accuracy,AUROC()],callback_fns=[partial(CSVLogger,append=Tru
```

```
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.cache/torch/checkpoints/resnet50-19c8e357.pth
100%|██████████| 102502400/102502400 [00:00<00:00, 154739169.85it/s]
```

Figure 14. Import torchvision

We use the learning-rate finder developed by Dr. Leslie Smith and implemented by the fastai team in their library:

```
learn.lr_find()
```

Figure 15. LR Finder

LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.

```
learn.recorder.plot(suggestion=True)
```

```
Min numerical gradient: 9.12E-07
Min loss divided by 10: 3.98E-03
```

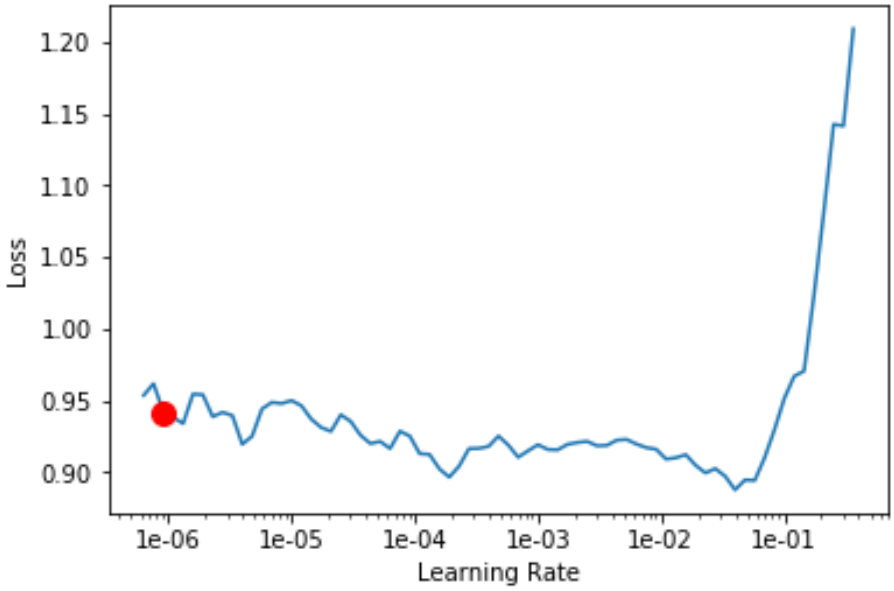

Figure 16. Learning Rate

Here we can see that the loss decreases fastest around `lr=1e-2` so that is what we will use to train:

```
learn.fit_one_cycle(1,max_lr = 1e-2)
```

epoch	train_loss	valid_loss	accuracy	auroc	time
0	0.300815	0.289540	0.890645	0.892545	26:23

Figure 17. Learn. lr=1e-2

```
learn.recorder.plot_losses()
```

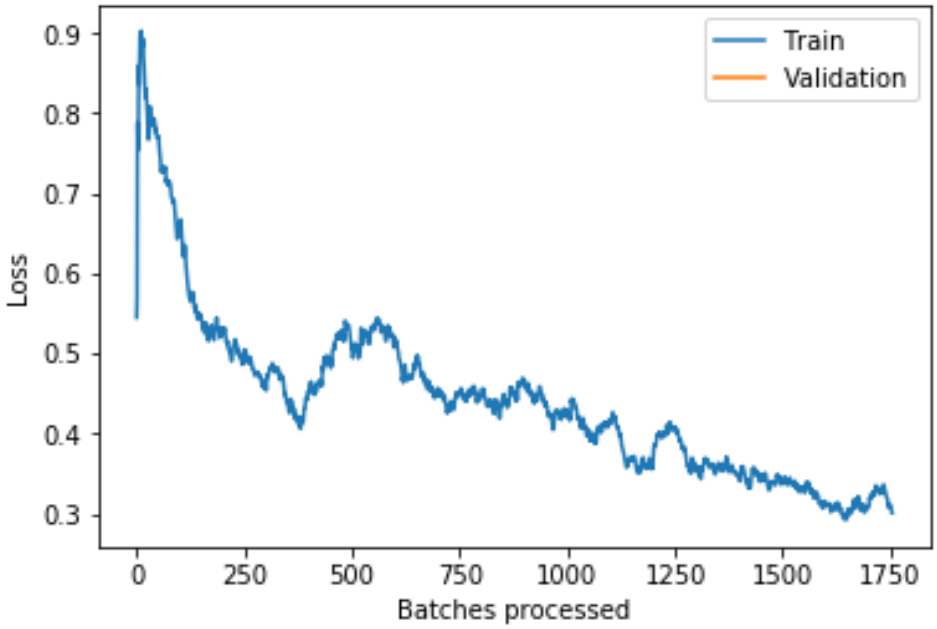

Figure 18. Learning Rate

```
learn.save('stage-1-512')
```

Figure 19. Learn Save

The previous model only trained the last model. We can unfreeze the rest of the model, and train the rest of the model using discriminative learning rates. The first layers aren't changed as much, with lower learning rates, while the last layers are changed more, with higher learning rates. We use the learning rate finder again, and use a range of learning rates for different layers in the neural network.

```
learn.load('stage-1-512')
```

Figure 20. Learn Load

```
Learner(data=ImageDataBunch;

Train: LabelList (28085 items)
x: ImageList
Image (3, 512, 512),Image (3, 512, 512),Image (3, 512, 512),Image (3, 512, 512),Image (3, 512, 512)
y: CategoryList
0,0,0,1,0
Path: .;

Valid: LabelList (7023 items)
x: ImageList
Image (3, 512, 512),Image (3, 512, 512),Image (3, 512, 512),Image (3, 512, 512),Image (3, 512, 512)
y: CategoryList
1,0,0,0,0
Path: .;

Test: None, model=Sequential(
  (0): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (4): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (downsample): Sequential(
          (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
      )
      (2): Bottleneck(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
      )
    )
    (5): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
```

Figure 21. Output Learn Load (a)

```
  (1): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
  )
  (2): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
  )
  (3): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
  )

6): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
    (downsample): Sequential(
      (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
  )
  (2): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
  )
  (3): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
  )
```

Figure 22. Output Learn Load (b)

```
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (7): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
)
(1): Sequential(
  (0): AdaptiveConcatPool2d(
    (ap): AdaptiveAvgPool2d(output_size=1)
    (mp): AdaptiveMaxPool2d(output_size=1)
  )
  (1): Flatten()
  (2): BatchNorm1d(4096, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (3): Dropout(p=0.25)
  (4): Linear(in_features=4096, out_features=512, bias=True)
  (5): ReLU(inplace)
  (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (7): Dropout(p=0.5)
  (8): Linear(in_features=512, out_features=2, bias=True)
)
```

Figure 23. Output Learn Load (c)

```
), opt_func=functools.partial(<class 'torch.optim.adam.Adam'>, betas=(0.9, 0.99)), loss_func=Flattene
dLoss of CrossEntropyLoss(), metrics=[<function accuracy at 0x751816817268>, AUROC()], true_wd=True,
bn_wd=True, wd=1e-05, train_bn=True, path=PosixPath('.'), model_dir='models', callback_fns=[functool
s.partial(<class 'fastai.basic_train.Recorder'>, add_time=True, silent=False), functools.partial(<cla
ss 'fastai.callbacks.csv_logger.CSVLogger'>, append=True)], callbacks=[], layer_groups=[Sequential(
  (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace)
  (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (4): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (10): ReLU(inplace)
  (11): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (13): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (14): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (15): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (16): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (17): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (19): ReLU(inplace)
  (20): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (21): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (22): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (23): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (24): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (25): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (26): ReLU(inplace)
  (27): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (28): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (29): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (30): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (31): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (32): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (33): ReLU(inplace)
  (34): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
  (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (36): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (37): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (38): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (39): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (40): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (42): ReLU(inplace)
  (43): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (44): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (45): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (46): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (47): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (48): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (49): ReLU(inplace)
  (50): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (51): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (52): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (53): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (54): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (55): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (56): ReLU(inplace)
```

Figure 24. Output Learn Load (d)

```
), Sequential(
  (0): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (4): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (6): ReLU(inplace)
  (7): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
  (8): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (9): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (10): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (11): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (13): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (14): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (15): ReLU(inplace)
  (16): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (17): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (18): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (19): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (20): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (21): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (22): ReLU(inplace)
  (23): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (24): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (25): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (26): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (27): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (28): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (29): ReLU(inplace)
  (30): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (31): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (32): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (33): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (34): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (35): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (36): ReLU(inplace)
  (37): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (38): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (39): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (40): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (41): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (42): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (43): ReLU(inplace)
  (44): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (45): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (46): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (47): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (48): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (49): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (50): ReLU(inplace)
  (51): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
  (52): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (53): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (54): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (55): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (56): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (57): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (58): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (59): ReLU(inplace)
  (60): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (61): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (62): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (63): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (64): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (65): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (66): ReLU(inplace)
```

Figure 25. Output Learn Load (e)

```
), Sequential(
  (0): AdaptiveAvgPool2d(output_size=1)
  (1): AdaptiveMaxPool2d(output_size=1)
  (2): Flatten()
  (3): BatchNorm1d(4096, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (4): Dropout(p=0.25)
  (5): Linear(in_features=4096, out_features=512, bias=True)
  (6): ReLU(inplace)
  (7): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): Dropout(p=0.5)
  (9): Linear(in_features=512, out_features=2, bias=True)
)], add_time=True, silent=None)
```

Figure 26. Output Learn Load (f)

```
learn.unfreeze()
```

Figure 27. Learn Unfreeze

```
learn.lr_find()
learn.recorder.plot()
```

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```

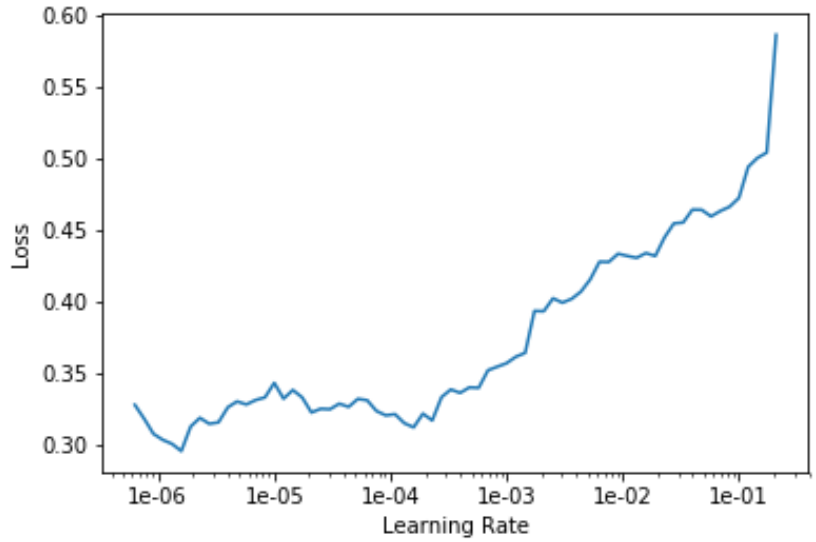

Figure 28. LR Finder

```
learn.fit_one_cycle(16, max_lr=slice(1e-6,1e-3))
```

epoch	train_loss	valid_loss	accuracy	auroc	time
0	0.309796	0.287437	0.892638	0.898791	27:32
1	0.292138	0.266092	0.899046	0.907896	27:36
2	0.314994	0.264474	0.899331	0.909767	27:41
3	0.308940	0.281859	0.893635	0.905164	27:44
4	0.307807	0.258364	0.903887	0.920777	27:45
5	0.311770	0.246806	0.907732	0.925023	27:47
6	0.251889	0.229282	0.912715	0.933077	27:44
7	0.250837	0.236006	0.910295	0.934041	27:42
8	0.278792	0.227283	0.916987	0.937943	27:40
9	0.276880	0.219994	0.918696	0.938203	27:42
10	0.231003	0.219166	0.918411	0.940106	27:39
11	0.242905	0.211952	0.921543	0.942848	27:47
12	0.214952	0.209923	0.921828	0.944746	27:35
13	0.214136	0.208065	0.923537	0.944748	27:40
14	0.220825	0.207859	0.921971	0.944531	27:40
15	0.254305	0.208727	0.921828	0.944648	27:47

Figure 29. Learn 16

```
learn.recorder.plot_metrics()
```

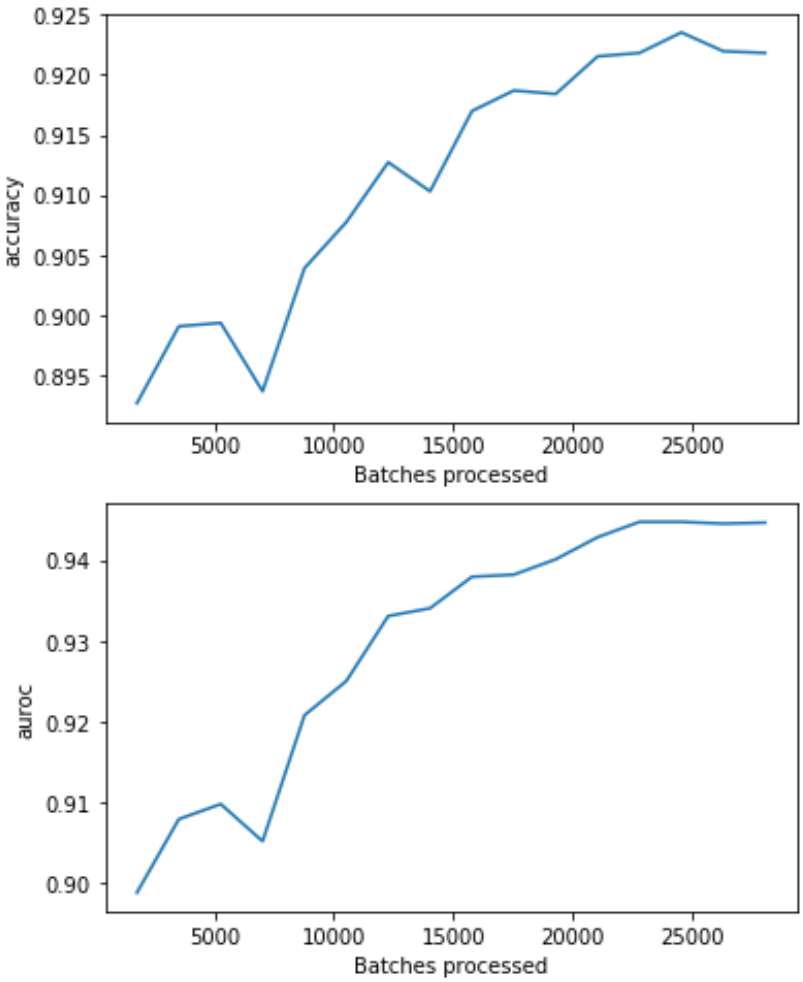

Figure 30. Learn Recorder Plot Matrix

```
learn.save('stage-2-512')
```

Figure 31. Learn Save

Section 5

CHECKING RESULT

We look at our predictions and make a confusion matrix.

```
interp = ClassificationInterpretation.from_learner(learn)

losses,idxs = interp.top_losses()

len(data.valid_ds)==len(losses)==len(idxs)
```

```
True
```

Figure 32. Classification Interpretation

```
interp.plot_top_losses(9, figsize=(15,11))
```

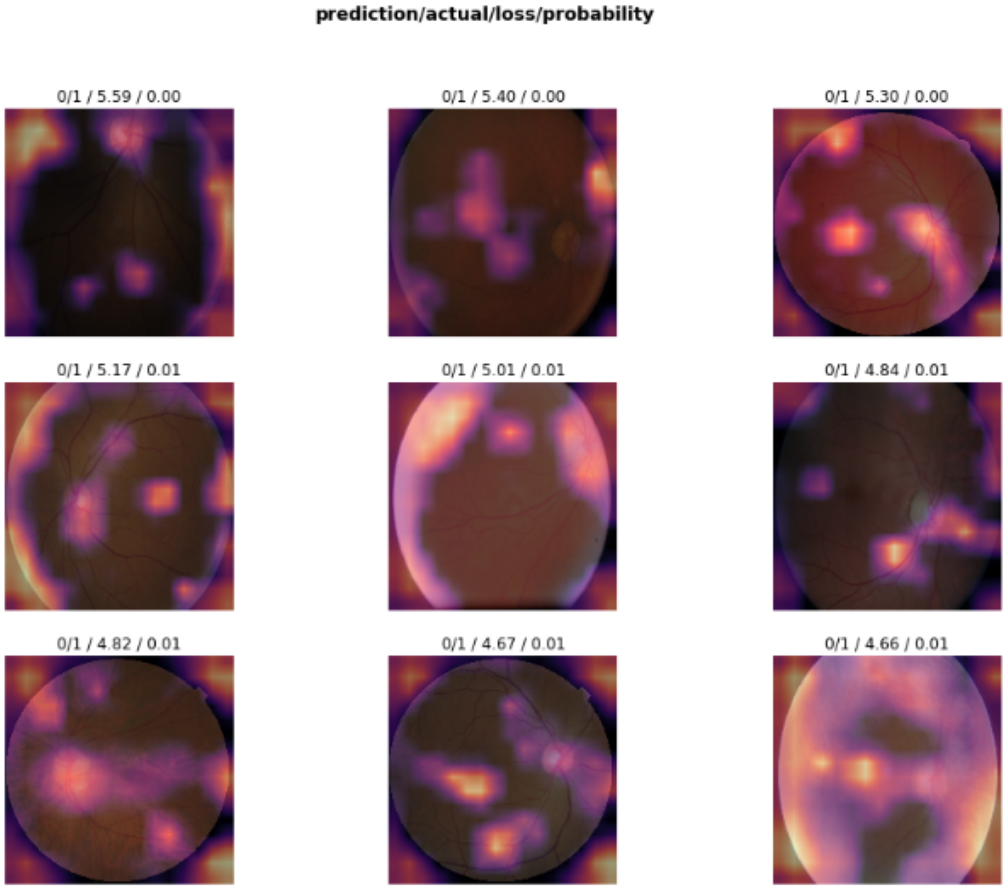

Figure 33. Intrepretation Top Losses

```
interp.plot_confusion_matrix(figsize=(12,12), dpi=60)
```

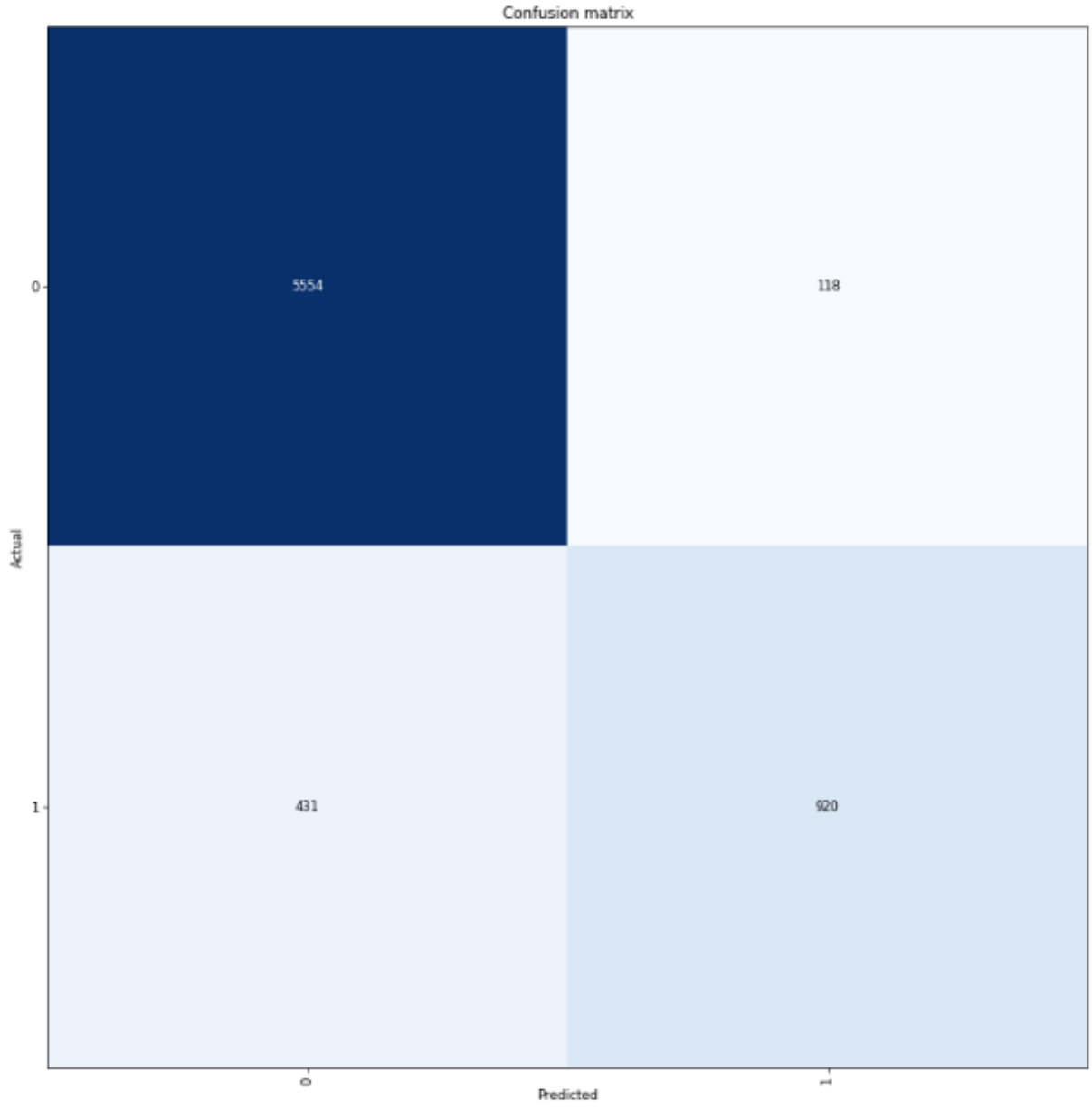

Figure 34. Interpretation Plot Confusion Matrix

```
from sklearn.metrics import roc_auc_score, confusion_matrix
def auc_score(y_score,y_true):
    try:
      auc = torch.tensor(roc_auc_score(y_true,y_score[:,1]))
    except:
      auc = torch.tensor(np.nan)
    return auc

probs,val_labels = learn.get_preds(ds_type=DatasetType.Valid)
print('Accuracy',accuracy(probs,val_labels)),
print('Error Rate', error_rate(probs, val_labels))
print('AUC', auc_score(probs,val_labels))


tn, fp, fn, tp = confusion_matrix(val_labels,(probs[:,1]>0.5).float()).

specificity = tn/(tn+fp)
sensitivity = tp/(tp+fn)

print('Specificity and Sensitivity',specificity,sensitivity)
```

```
Accuracy tensor(0.9218)
Error Rate tensor(0.0782)
AUC tensor(0.9446)
Specificity and Sensitivity 0.9791960507757405 0.6809770540340488
```

Figure 35. Specificity and Sensitivity (b)

Our model has decent AUC (around 0.94), which is similar to some of the results demonstrated previously. However, this is still not close to AUC=0.99 demonstrated by Google.

We can use an optimized threshold as well:

```
from sklearn.metrics import roc_curve
def Find_Optimal_Cutoff(target, predicted):
    """ Find the optimal probability cutoff point for a classification model related to event rate
    Parameters
    ----------
    target : Matrix with dependent or target data, where rows are observations

    predicted : Matrix with predicted data, where rows are observations

    Returns
    -------
    list type, with optimal cutoff value

    """
    fpr, tpr, threshold = roc_curve(target, predicted)
    i = np.arange(len(tpr))
    roc = pd.DataFrame({'tf' : pd.Series(tpr-(1-fpr), index=i), 'threshold' : pd.Series(threshold, index=i)})
    roc_t = roc.ix[(roc.tf-0).abs().argsort()[:1]]

    return list(roc_t['threshold'])


print(probs[:,1])

threshold = Find_Optimal_Cutoff(val_labels,probs[:,1])
print(threshold)

print(confusion_matrix(val_labels,probs[:,1]>threshold[0]))

tn, fp, fn, tp = confusion_matrix(val_labels,probs[:,1]>threshold[0]).ravel()

specificity = tn/(tn+fp)
sensitivity = tp/(tp+fn)

print('Specificity and Sensitivity',specificity,sensitivity)
```

```
tensor([0.9883, 0.0116, 0.0209,  ..., 0.0086, 0.0051, 0.1310])
[0.13500292599201202]
[[4934  738]
 [ 176 1175]]
Specificity and Sensitivity 0.8698871650211566 0.8697261287934863
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:18: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
```

Figure 36. Specificity and Sensitivity (b)

Section 6

ACKNOWLEDGEMENTS

My gratitude to you Professor Park Sanguk for the class. I truly appreciate you and the time you spent. Thank you very much for the class. I enjoyed every minute of your lecture. Thank you, truly and sincerely.

Contact

Salaki Reynaldo Joshua
E-mail: salakirjoshua@kangwon.ac.kr

VIEW THE ASSIGNMENT

Google Drive

Google Colab

The assignment is available in Google Drive and Google Colaboratory.