Melanoma-Discriminator Report

11710116 沈静然 (Peter S)	11710823 卫焱滨 (Wei Yanbin)
11710422 王奥 (Wang Ao)	11710215 彭维源 (Peng Weiyuan)

Project Introduction

Task, Motivation & Data Source

In this project, we are going to complete a binary classification so as to decide whether a given image (along with some additional metadata) represents a melanoma disease.

Melanoma is a deadly disease, but if caught early, most melanomas can be cured with minor surgery. That is why identification is so important.

The data is from SSIM-ISIC Melanoma Classification Challenge: https://challenge2020.isic-archive.com/

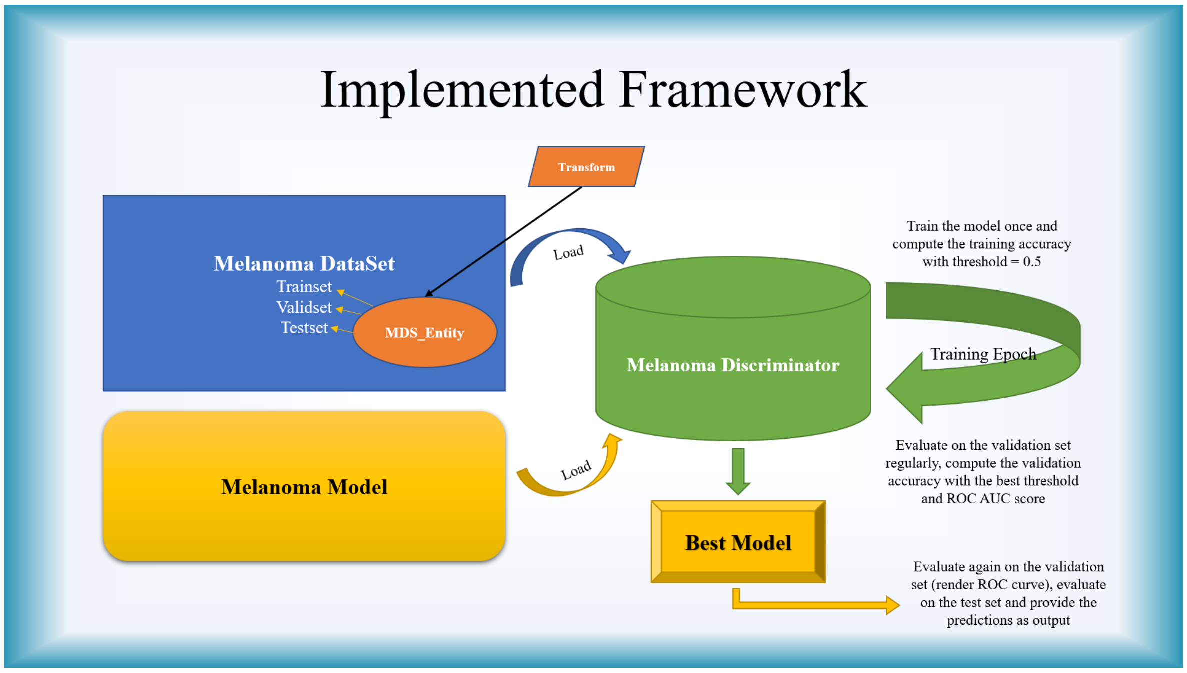
Workload (Labor Division)

- Peter S (Shen Jingran)
  - Framework Design & Implementation
    - Build up customized dataset `MelanomaDataSet` with metadata inputs
    - Implement the training-validation framework
    - Implement the evaluation (validation-test) framework
  - Use EfficientNet to construct the training model
  - Implement `DataAnalyzer` to briefly measure data imbalance
- Wei Yanbin
  - VGG-Like CNN
  - Metadata
- Peng Weiyuan
  - ResNet/ResNeXt
  - Misclassified Image Analysis
- Wang Ao
  - VGG
  - Data Augmentation

Project Details

Framework [Peter S]

The structure of the framework is briefly summarized as the figure below:


Implemented Framework
Transform
Melanoma DataSet
Trainset
Validset
Testset
MDS_Entity
Load
Melanoma Discriminator
Train the model once and compute the training accuracy with threshold = 0.5
Training Epoch
Evaluate on the validation set regularly, compute the validation accuracy with the best threshold and ROC AUC score
Melanoma Model
Load
Best Model
Evaluate again on the validation set (render ROC curve), evaluate on the test set and provide the predictions as output

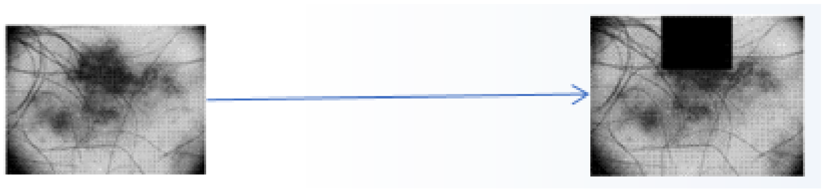
Data Augmentation [Wang Ao]

Because our train set only has 748 pictures. We think it is not enough to train a model which has good performance. And We try to use data augmentation to expend our train set.

Some Methods we have tried

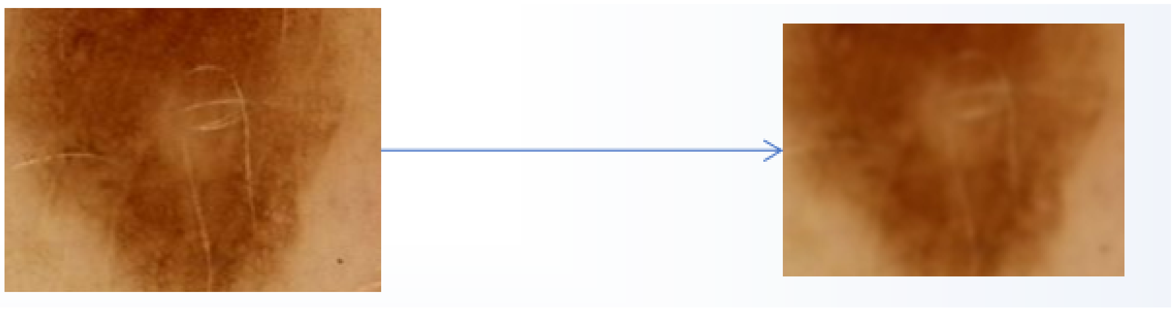
We have tried some basic methods, such as **resize** the picture, **crop** the picture randomly, **flip** the picture, **rotate** the picture at a random angle.

What's more, we also have tried **adding noise in picture and random masking** which means random selecting an area and setting the pixel to zero.



Above picture shows an example of random masking processing. After the processing of random masking, the top area of the picture turns black.

We all know that the Hair on the skin has a bad impact on the diagnosis of skin diseases. And in order to reduce the difference of each pixel value to achieve image blur and smooth pixels. We also use the method of **Gaussian blur**.



Above picture shows an example of Gaussian blur processing. After the processing of Gaussian blur, hair has less effect on diagnosis.

And we also plan to **adjust the brightness and contrast** of picture, **add hair to the image**.

Test of Data Augmentation

After we apply the method of data augmentation to model training. We found that it can only improve performance slightly. Which means that if we only use data augmentation but do not select proper parameters, the performance will not be good.

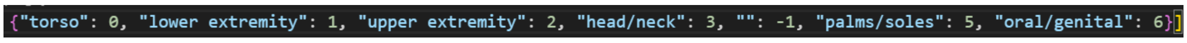
Metadata [Wei Yanbin]

To aggregate the metadata into our project. We use a method which looks a little bit simply.

Encode the enum into numbers

First, encode the types of items value into a number.

For columns in metadata csv, some of them has fixed several possible value like sex, it only has "male":"female":"(null) ". For this type of value, we can just encode them corresponding into "1":"0":"-1". Similarly, the encode approach of column "anatom_site_general_challenge" is as follows:

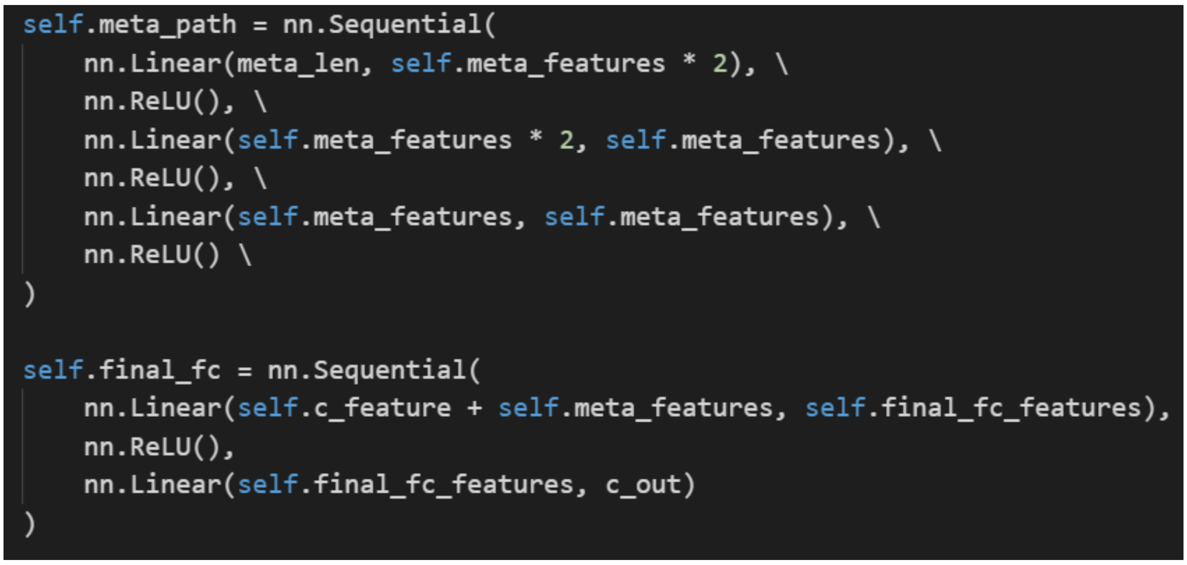


```
{"torso": 0, "lower extremity": 1, "upper extremity": 2, "head/neck": 3, "": -1, "palms/soles": 5, "oral/genital": 6}
```

And for another type of column like age, the value is meaningful and is a number itself. So we can just use age as a input of the network for metadata.

Modify models to mix image path and metadata path

The module should be modified yet. And the method to mixed metadata and image parts are aggregate them into the same FC layer.



```
self.meta_path = nn.Sequential(
    nn.Linear(meta_len, self.meta_features * 2), \
    nn.ReLU(), \
    nn.Linear(self.meta_features * 2, self.meta_features), \
    nn.ReLU(), \
    nn.Linear(self.meta_features, self.meta_features), \
    nn.ReLU() \
)

self.final_fc = nn.Sequential(
    nn.Linear(self.c_feature + self.meta_features, self.final_fc_features),
    nn.ReLU(),
    nn.Linear(self.final_fc_features, c_out)
)
```

Infomation Leakage Hazard

There is a problem need to be noticed carefully during the metadata processing. That is when we use metadata csv given, some columns of them directly reveals the target of dataset, such as diagnosis, benign, may cause information leakage hazard.

For more details, can refer to *Problems and Solutions* Part.

Dataset Structure [Peter S]

The Melanoma Dataset contains three sets: the training set, the validation set and the test set. There will be two transformation functions separately for training and evaluating. The evaluation process uses the validation set to confirm and the test set to predict.

Each of these three sets is essentially an `MDS_Entity` instance, which inherits the PyTorch's `Dataset` class and can be used to construct PyTorch's `DataLoader`. When `MDS_Entity` is initialized, a list of sequential sample labels of images will be pre-processed so as to help with ROC AUC computations of the validation set as well as predictions on the test set. The number of positive samples and negative samples will also be counted in order to help find the best threshold for the model on the validation set.

One sample will contain information as follow:

- A tensor processed from image transformations
- A set of metadata (as a dictionary)
- An ensemble tensor of the metadata, for training
- The label/target value for the sample

Model

Parameter Settings [Peter S]

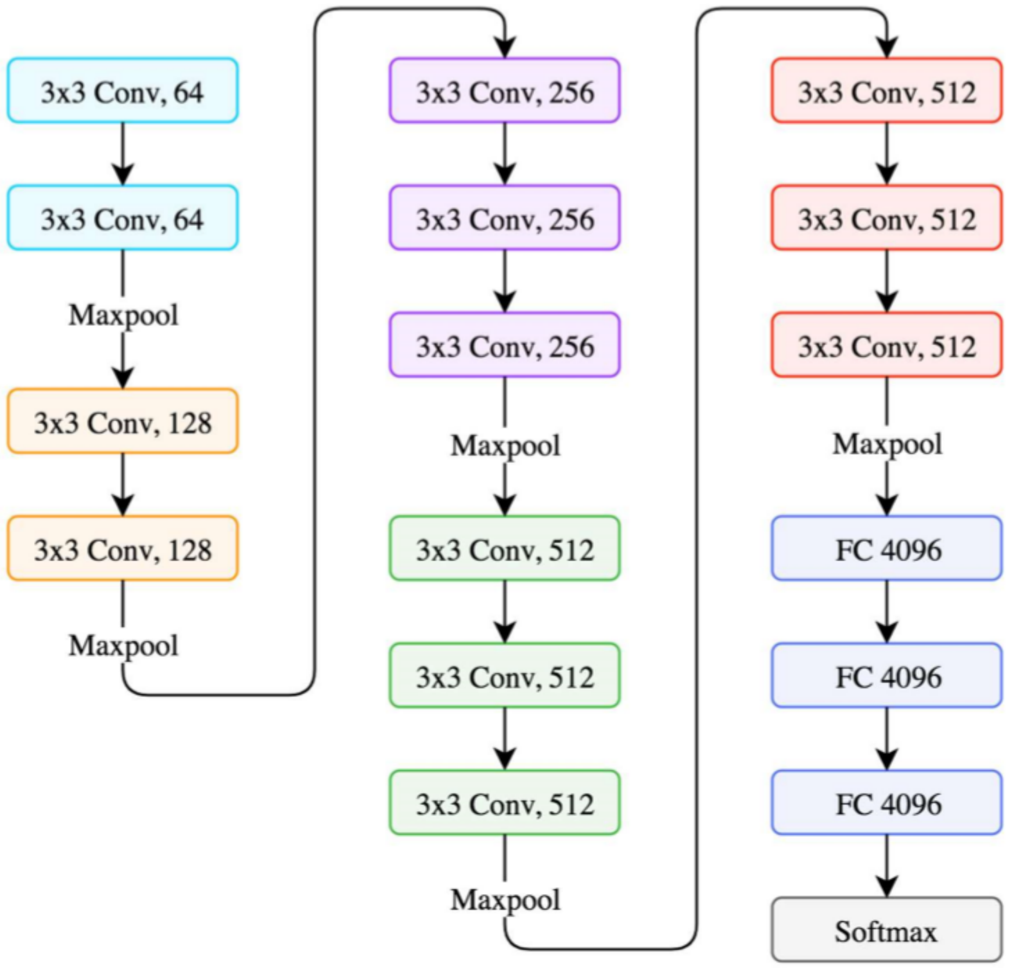
We use different learning rates and image sizes to test our models. There are also three options for the optimizer: Adam, RMSprop and AdamW.

We use `BCEWithLogitsLoss` here as the criterion, since it is combined with a sigmoid unit as well as a `BCELoss` unit. The output from the model will be applied a sigmoid function to serve as probabilities. Probability values are between 0 and 1. That is why sigmoid function is used, rather than softmax or tanh (of course they can be used but sigmoid is more straightforward). And thus this is why `BCEWithLogitsLoss` is used rather than `CrossEntropyLoss`.

VGG [Wang Ao]

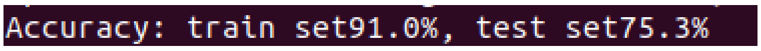
Structure of VGG16

I choose the network structure VGG16 to implement the binary classification. I refer to the VGG16 network structure in the slide of deep learning course.



Above figure show the network structure of VGG16. This network structure has 6 parts. The first five parts are convolution parts and the last part is full connected part. And each convolution part consists of two or three convolution layers and one max pooling layer. Full connected part has three full-connected layers.

Performance of VGG16



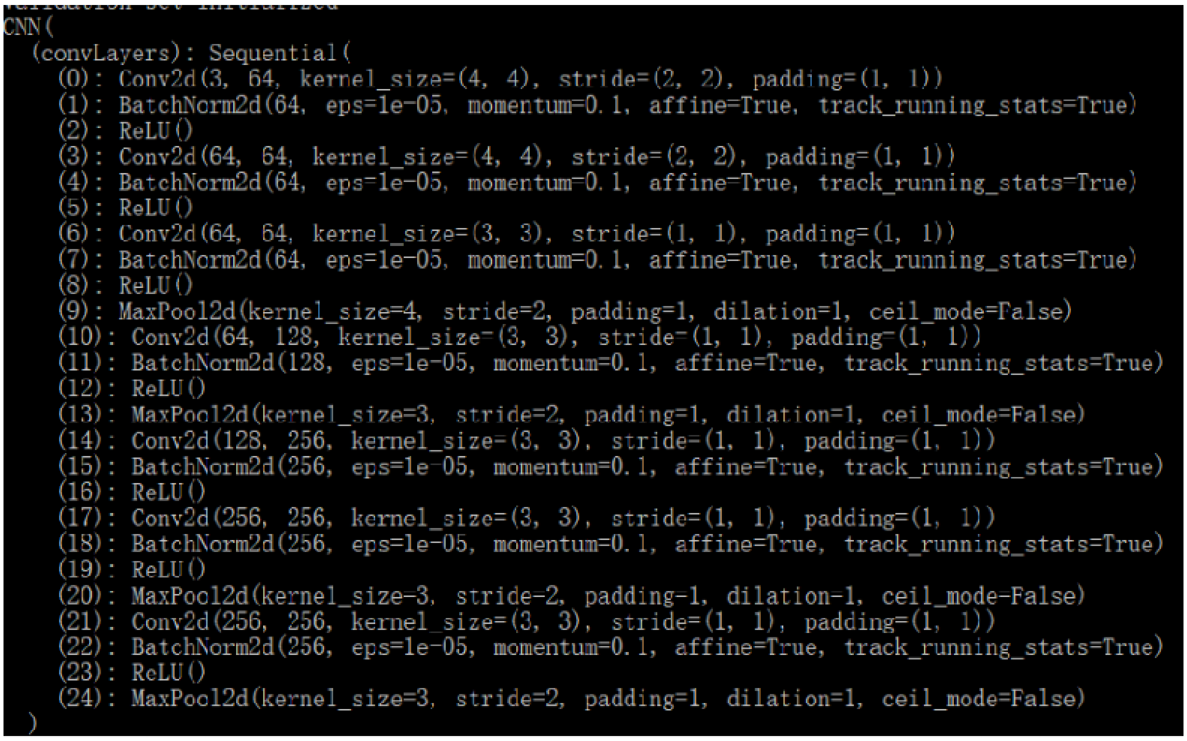
```
Accuracy: train set91.0%, test set75.3%
```

After training process. I found that the accuracy in train set can achieve 90% and the accuracy in test set can achieve 75%. I think there are some problems in my model.

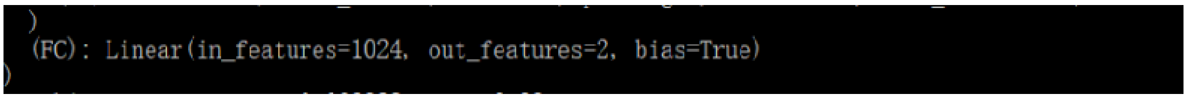
VGG-Like CNN [Wei Yanbin]

Structure of VGG-Like CNN

A VGG-Like CNN model structure has been designed for the binary classification, which is based on what we used for our previous assignment 2. Besides, some modify has occurred so the model structure is as:



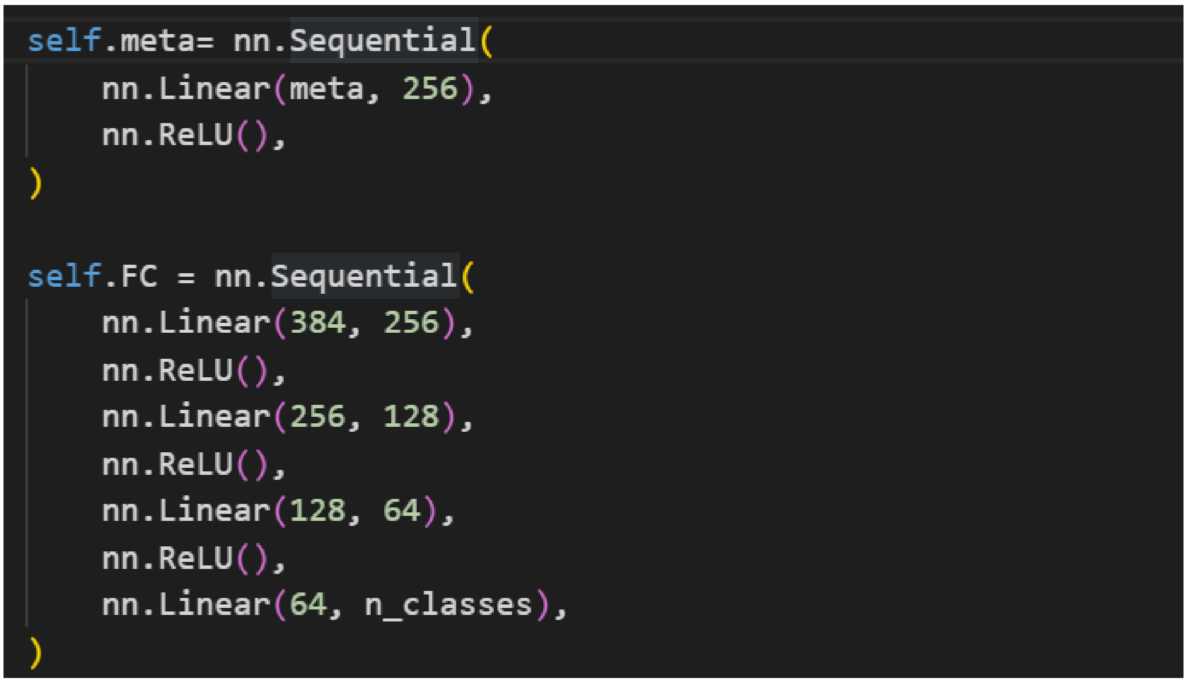
```
CNN(
  (convLayers): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(64, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
    (9): MaxPool2d(kernel_size=4, stride=2, padding=1, dilation=1, ceil_mode=False)
    (10): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU()
    (13): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): ReLU()
    (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (19): ReLU()
    (20): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (21): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (23): ReLU()
    (24): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
```



```
  )
  (FC): Linear(in_features=1024, out_features=2, bias=True)
)
```

The structure of the model has 24 layers for 2D-convolution, max-pooling and ReLU activation, with a fully connected layer to do classification.

After adding metadata, the model structure has a extra meta layer and FC connect them.



```
self.meta= nn.Sequential(
    nn.Linear(meta, 256),
    nn.ReLU(),
)

self.FC = nn.Sequential(
    nn.Linear(384, 256),
    nn.ReLU(),
    nn.Linear(256, 128),
    nn.ReLU(),
    nn.Linear(128, 64),
    nn.ReLU(),
    nn.Linear(64, n_classes),
)
```

Hyper-parameters Tuning

Some hyper-parameters are of importance

```
LEARNING_RATE_DEFAULT = 1e-4

BATCH_SIZE_DEFAULT = 64

MAX_EPOCHS_DEFAULT = 3000

EVAL_FREQ_DEFAULT = 5

USING_GPU_DEFAULT = True

DATA_AUGMENTATION_DEFAULT = False

THREADS_DEFAULT = 8
```

Performance

Model	Validation Set Accuracy	ROC-AUC
Like-VGG CNN	76 % ~ 80 %	81.78 %
Like-VGG CNN(with meta)	73 % ~ 81 %	83.71 %

ResNet/ResNeXt [Peng Weiyuan]

ResNet

A residual learning framework to ease the training of networks that are deeper than those used previously. ResNet archives the SOA performance in 2016, and get the best paper awards in CVPR 2016.

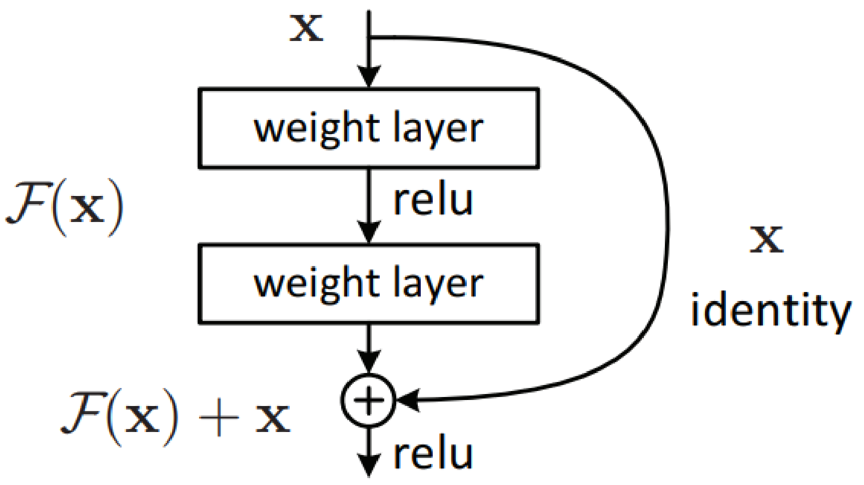


Figure 2. Residual learning: a building block.

ResNeXt

ResNeXt is a simple, highly modularized network architecture for image classification. This network is constructed by repeating a building block that aggregates a set of transformations with the same topology. ResNeXt archives the SOA performance in 2018, so choosing ResNeXt as the basic model may give some performance improvement.

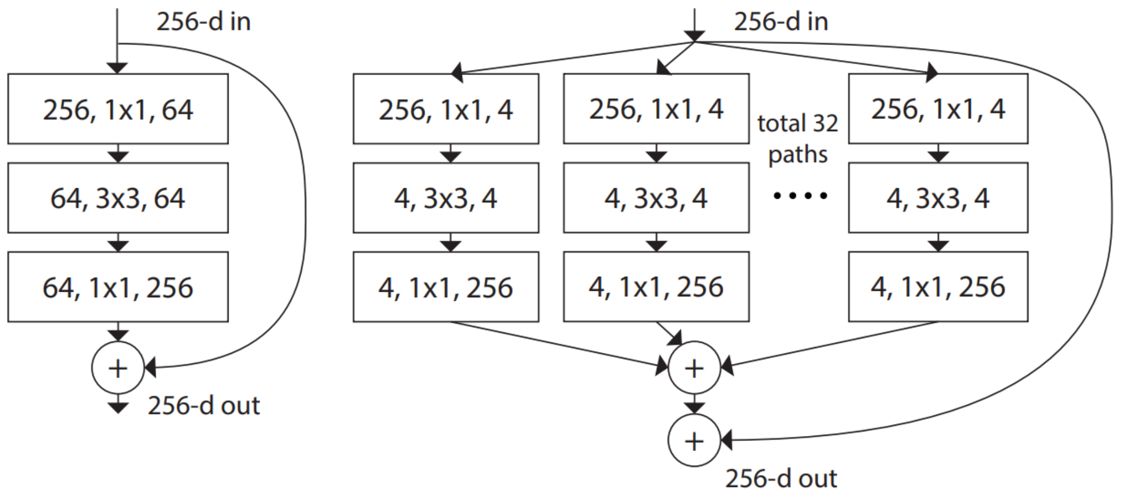


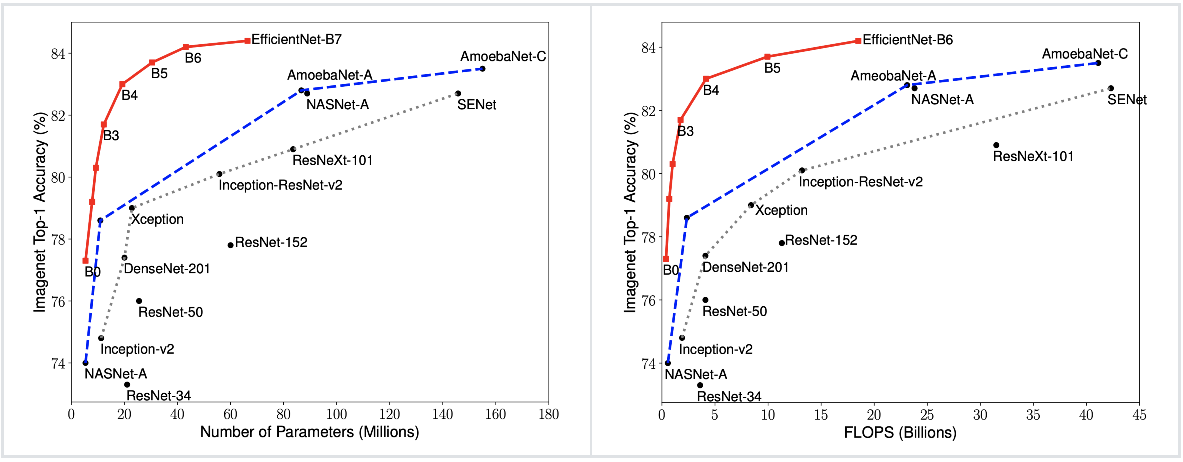
Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Basic Model Selection

Because the dataset is not as big as hundreds of GBs like the original Melanoma dataset. The too big model will cause overfitting and training too slow. Small models like ResNet-50 and ResNeXt-50 are good chooses, they are not too big but still have good performance in testing.

EfficientNet [Peter S]

EfficientNets are a family of image classification models, which achieve state-of-the-art accuracy, yet being order-of-magnitude-smaller and faster-than-previous models. The EfficientNets are developed based on AutoML and Compound Scaling.



From the figures shown above, we see that the training accuracy of EfficientNets seem to achieve much better results than other classification models (such as ResNet/ResNeXt) and the outcome is more obvious as the number of parameters or the number of FLOPS increases. Generally speaking, when picking models for image classification as a start, EfficientNet seems to always be a nice choice.

For this project, considering the GPU resources we can consume, only EfficientNet-b1 with image input size as 240 is used. Additionally, from the Kaggle Notebook of the project, there are certain articles suggesting that a pooling layer as well as a dropout layer are used.

The model is constructed simply as follow:

```
EfficientNet-b1
Adaptive_Avg_Pool2d(output_size=1)
Dropout(p=0.2)
Linear(x, 1)
```

If the metadata is used, it will go through these layers:

```
Linear(meta_len, meta_features * 2)
ReLU()
Dropout(p=0.2)
Linear(meta_features * 2, meta_features)
BatchNorm1d(meta_features)
ReLU()
Dropout(p=0.2)
```

The two outputs will be concatenated and passed through these layers:

```
Linear(efnet_features + meta_features, final_fc_features)
ReLU()
Dropout(p=0.2)
Linear(final_fc_features, 1)
```

The metadata layers and the final fully-connected layers are designed in a very simple way, since EfficientNet is already huge for training.

Training/Evaluation Process [Peter S]

The training-validation process is as follow:

- At each epoch, the model is trained on the training set once. The training accuracy under threshold = 0.5 and the general training loss are computed
- In the meantime, at each evaluation epoch, the model will accept data from the validation set. The validation accuracy under the computed best threshold as well as the ROC AUC score are calculated. The model under current status will be saved to local as the best model as long as the following conditions are satisfied:
    - The training accuracy for this epoch is not below 90%
    - The ROC AUC score for the validation set is better than historical record

The evaluation process is as follow:

Evaluate using saved best model (Note that this is a Test API)

1. Re-evaluate on the validation set
2. Find the ROC AUC score
3. Plot the ROC curve
4. Find the optimal threshold for the model/network
5. List the misclassified images in the validation set
6. Use the optimal threshold to predict the test set

Results

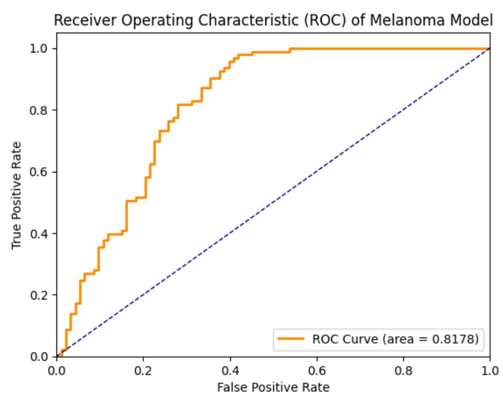
The results below all use AdamW with learning rate as 0.0001 and are recorded as the training results become relatively steady. Note that the ROC AUC score and the accuracy are all on the validation set:

Model	ROC AUC	Accuracy
VGG	75.30%	70%~75%
VGG-Like CNN	81.78%	76%~80%
VGG-Like CNN (with meta)	83.71%	73%~81%
ResNeXt50	90.44%	80%~87%
ResNeXt50 (with meta)	89.39%	74%~85%
EfficientNet-b0	92.05%	Around 83%
EfficientNet-b0 (with meta)	**92.13%**	75%~88%

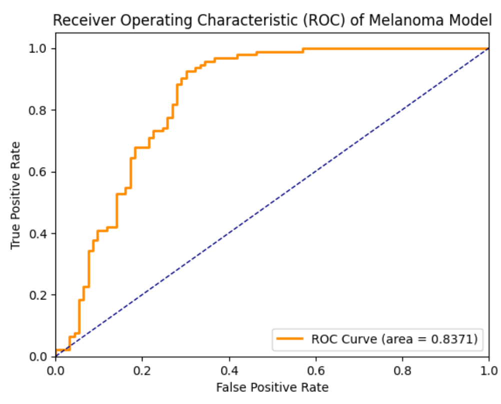
From the results we find that the model with the best ROC AUC score is the EfficientNet with metadata ensemble. The result reaches 92.13% which seems good. For the best threshold set as 0.3955, the validation accuracy reaches 88.71% which is so far the highest. The final `test.csv` is thus predicted using this model.

ROC curves

- VGG-Like CNN

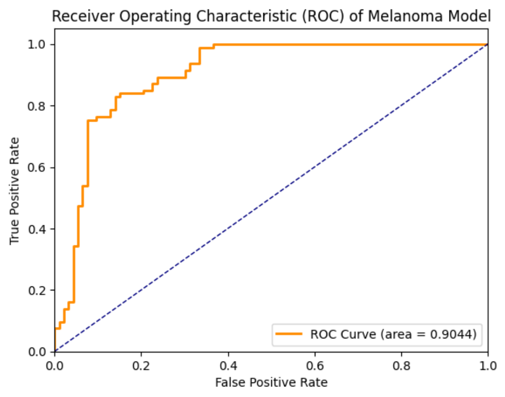


VGG-Like CNN

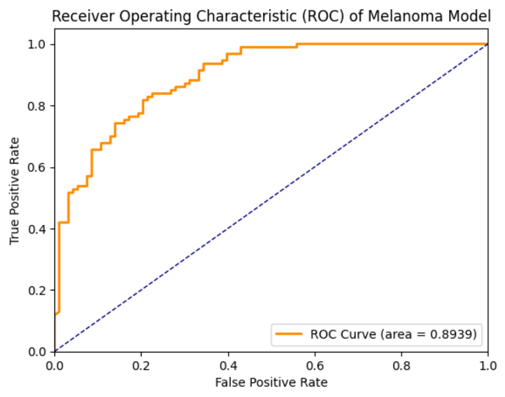


VGG-Like CNN (With meta)

- ResNeXt

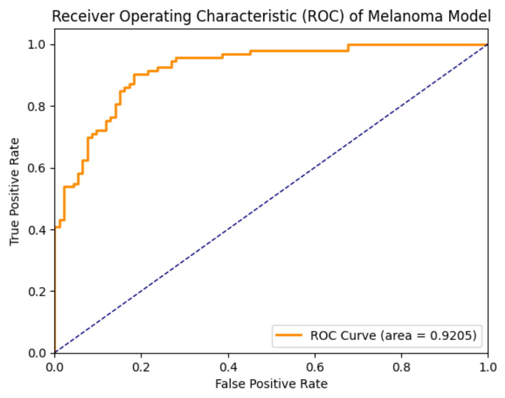


ResNeXt

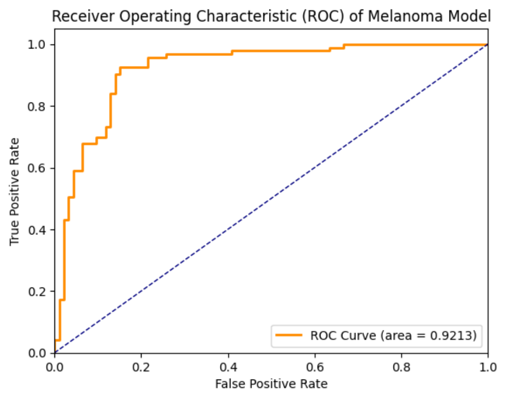


ResNeXt (With meta)

- EfficientNet



EfficientNet



EfficientNet (With meta)

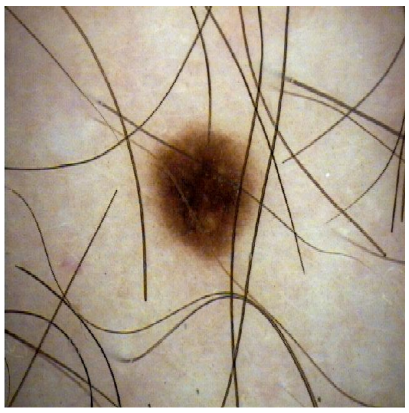
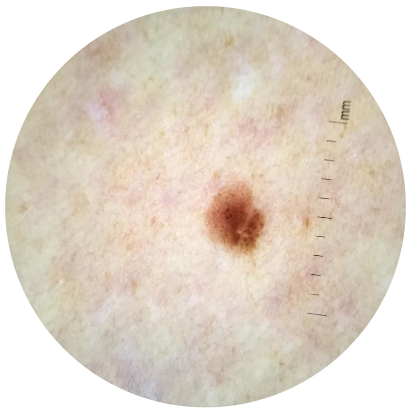
As we can see, ResNeXt and EfficientNet tend to have higher true positive rates when the false positive rates are low. From the curves, EfficientNet with metadata obtains the largest AUC among all.

Problems & Solutions

- **[Peter S] For the metadata, there are some empty values in the csv files. How to deal with them?**

  There are two probable ways to handle empty values for these csv files. The first approach is to ignore them. For the metadata ensemble tensor, if empty values occur, the tensor will not receive any 1s. The second approach is to balance the data, by applying meaningful values that show up with the smallest frequency.

- **[Peter S] Some images have their own noise features. For example, in the training set there are some images that contain lots of hairs (e.g., `img_150.jpg`), which become certain noises. There are also some images retrieved using some sort of telescopic methods (e.g., `img_195.jpg`), whose surrounding white areas are needless. How to utilize the model to learn about these "extra" features (distinguish from these noises)?**

 img_150.jpg	 img_195.jpg

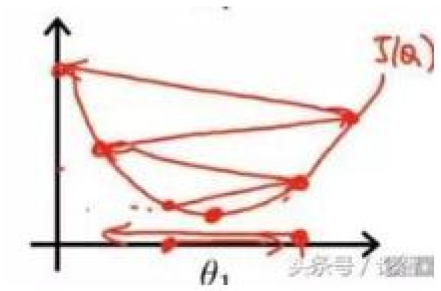
```
Use certain data augmentation skills. For hair noises, use `DrawHair`
function which draws a random number of pseudo-hairs in the sample image.
For telescope images, use `CenterCrop` or `RandomCrop` to crop out the white
regions.
```

- **[Peter S] When considering accuracy on the validation set, it matters which value (threshold) we choose to split the predictions of positive or negative samples. How to compute the best threshold, which can give us the highest accuracy?**

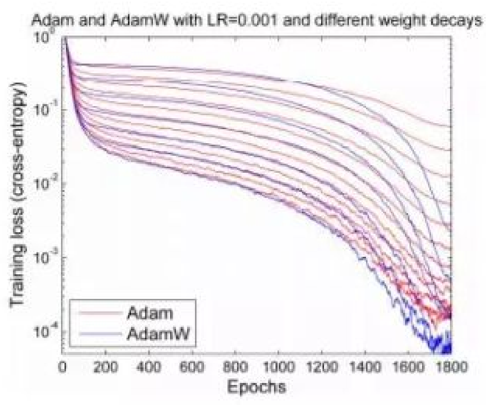
  There are lots of standards to compute the best threshold on. Simply, we use accuracy, which is the number of correctly predicted samples divided by total number of samples. Of course, we can also use precision (I have successfully identified lots of melanomas) or recall (My claims on the melanomas are mostly correct).

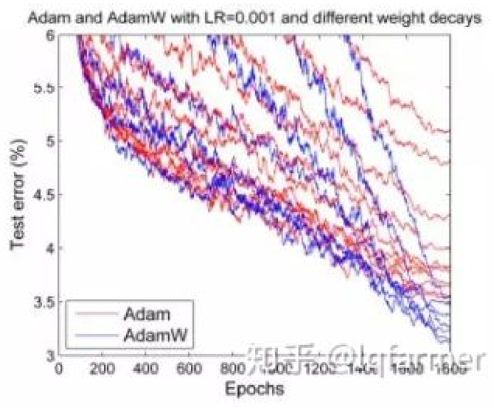
- **[Peter S] Why does EfficientNet achieve a much better performance after the learning rate is changed from 0.001 to 0.0001 and the AdamW optimizer is used instead of Adam or RMSprop?**

  For the learning rate, it seems that 0.001 is too large for the model and it keeps wandering around the theoretically best status. Its behavior is like the figure shown below:

  AdamW acts partially as a simpler version of Adam and it corrects the wrong implementations of Adam on multiple platforms. From the figure below we can see that AdamW performs better than Adam on both training loss and test error. The effect gets even more obvious as the epoch increases:





As for RMSprop, it seems to keep the training process more steady (with less fluctuations of accuracy values). However, Adam or AdamW have momentum to control this as well. The reason that RMSprop was better than Adam in the tests before might be that the learning rate 0.001 is too high so that the model jumps around more easily.

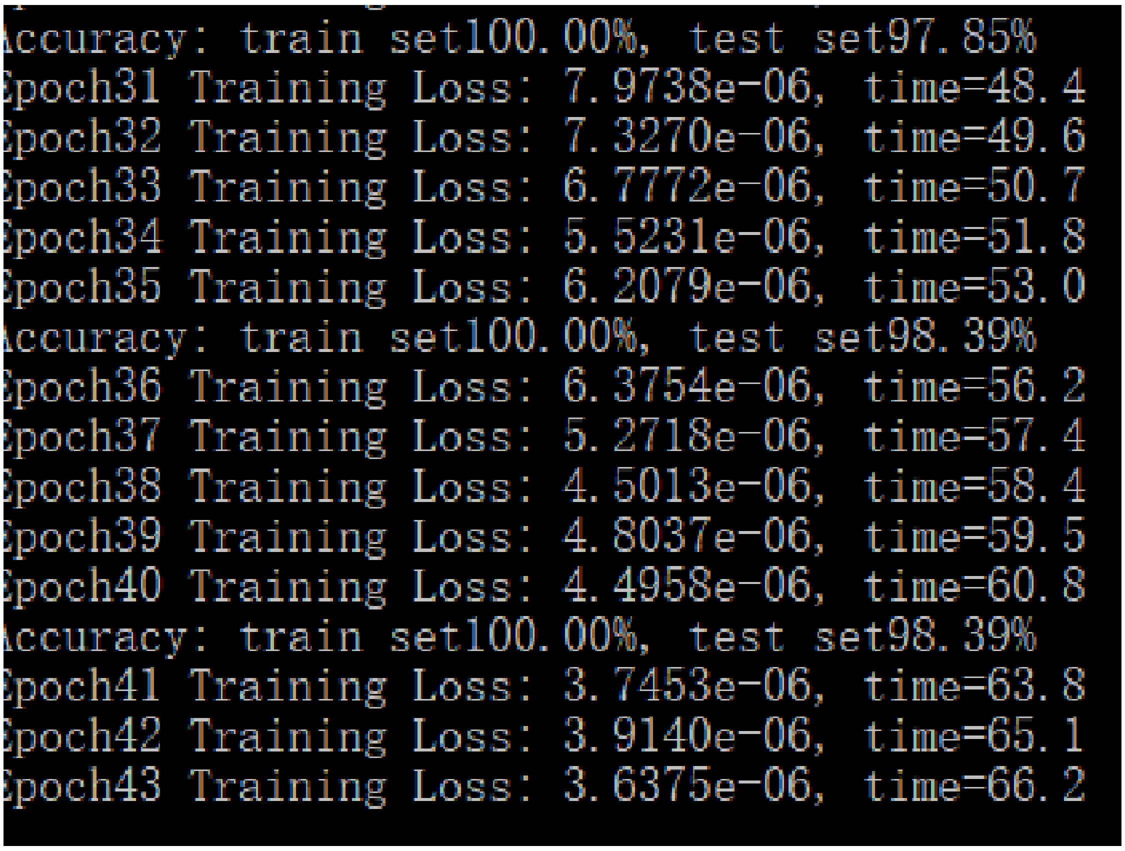
- **[Wang Ao] GPU resources are not enough when training VGG16?**

  When I tried to train the model on the server. There was always exception that GPU out of memory. So I have to select some network layers and delete them. And I think this may lead to the bad performance of VGG16.

- **[Wei Yanbin] Information Leakage Hazard**

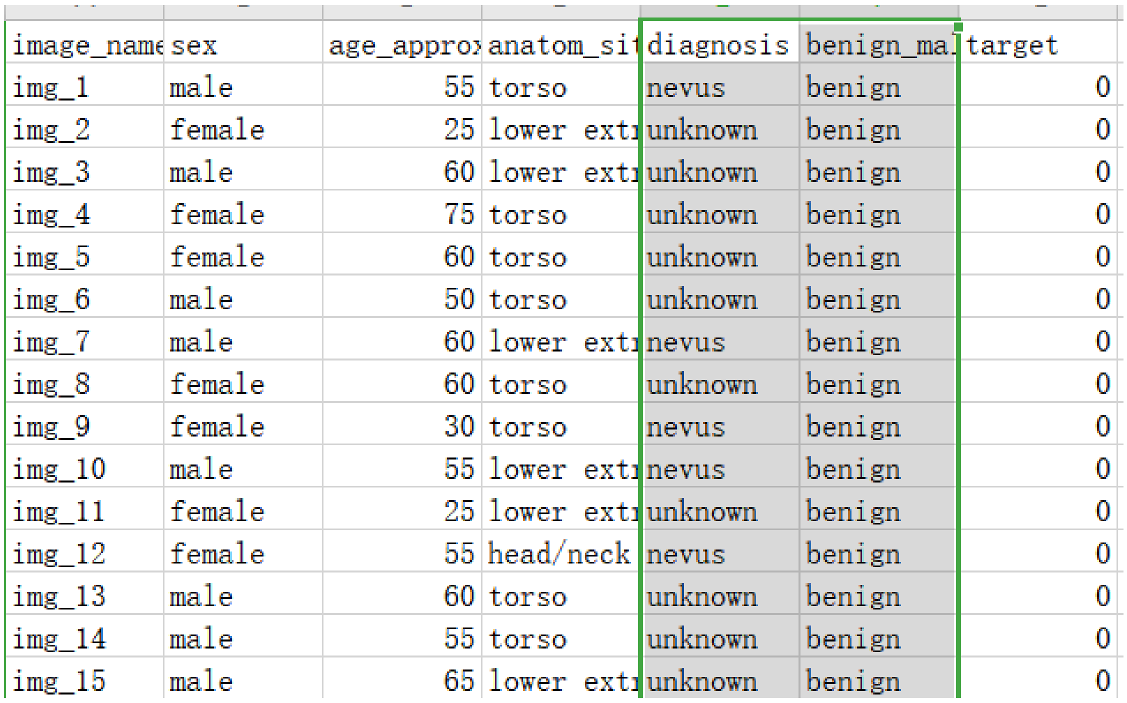
  There is a problem need to be noticed carefully during the metadata processing. That is when we use metadata csv given, some columns of them directly reveals the target of dataset, such as diagnosis, benign, may cause information leakage hazard.

  Problem Phenomenon: Model can perform similar perfect on validation set. But it is because of Information leakage !



```
ccuracy: train set100.00%, test set97.85%
poch31 Training Loss: 7.9738e-06, time=48.4
poch32 Training Loss: 7.3270e-06, time=49.6
poch33 Training Loss: 6.7772e-06, time=50.7
poch34 Training Loss: 5.5231e-06, time=51.8
poch35 Training Loss: 6.2079e-06, time=53.0
ccuracy: train set100.00%, test set98.39%
poch36 Training Loss: 6.3754e-06, time=56.2
poch37 Training Loss: 5.2718e-06, time=57.4
poch38 Training Loss: 4.5013e-06, time=58.4
poch39 Training Loss: 4.8037e-06, time=59.5
poch40 Training Loss: 4.4958e-06, time=60.8
ccuracy: train set100.00%, test set98.39%
poch41 Training Loss: 3.7453e-06, time=63.8
poch42 Training Loss: 3.9140e-06, time=65.1
poch43 Training Loss: 3.6375e-06, time=66.2
```

Analysis: The abnormal performance is caused by the two columns in csv of the meta_data, which reveal the target appearently.



image_name	sex	age_approx	anatom_sit	diagnosis	benign_ma	target
img_1	male	55	torso	nevus	benign	0
img_2	female	25	lower ext	unknown	benign	0
img_3	male	60	lower ext	unknown	benign	0
img_4	female	75	torso	unknown	benign	0
img_5	female	60	torso	unknown	benign	0
img_6	male	50	torso	unknown	benign	0
img_7	male	60	lower ext	nevus	benign	0
img_8	female	60	torso	unknown	benign	0
img_9	female	30	torso	nevus	benign	0
img_10	male	55	lower ext	nevus	benign	0
img_11	female	25	lower ext	unknown	benign	0
img_12	female	55	head/neck	nevus	benign	0
img_13	male	60	torso	unknown	benign	0
img_14	male	55	torso	unknown	benign	0
img_15	male	65	lower ext	unknown	benign	0

Solution: To avoid the Information leakage. Only use sex age_approx and anatom.. Then the performance looks normal.

Extra Contents

ROC & AUC

This part is already discussed in Results.

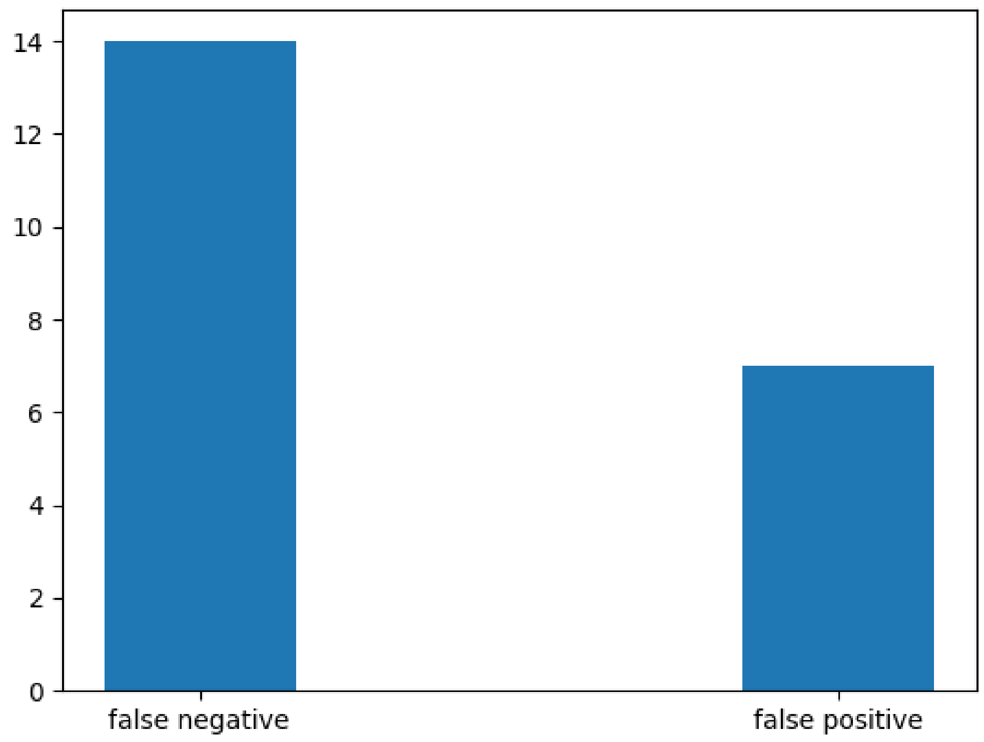
Misclassified Images in the Validation Set [Peng Weiyuan]

In the final validate process, 21 (of 186) images in validation set are misclassified.

Confusion Matrix

In these 21 photos, 14 are false-negative and 7 are false-positive.

- **False-Negative**: malignant sample been misclassified as benign.
- **False-Positive**: benign sample been misclassified as malignant.



We can see the number of false-negative is 2x of false positive, which means that the threshold of the model is still too high, predicting a large number of negative samples as positive.

If we want to improve the performance of the model, we can consider decreasing the threshold in model classification.

However, this threshold (0.3955 here) is calculated through the ROC analysis in the training process. Theoretically, the best threshold is reached on the training set.

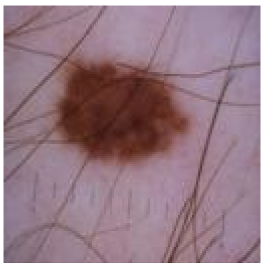
So, a possible reason is that the **data set is too small**, there is still a big difference between the training set and the validation set, and it is difficult to achieve the same result. Or some other accidental factors caused by the insufficient data set.

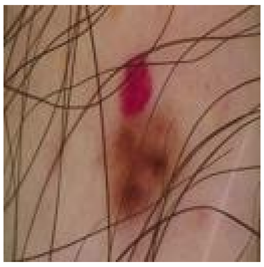
Unclear image

By seeing there misclassified images one by one, I found some of them are unclear image,

Hair

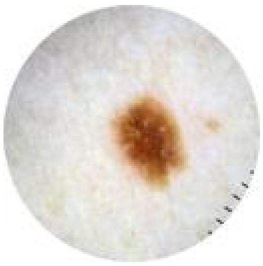
For example:



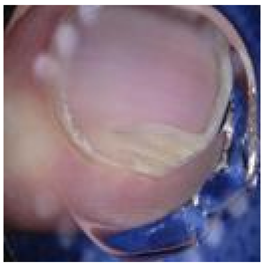


In these 2 images, they both have very obvious hairs and the number of hairs is much bigger the fake hair we draw in data augmentation (0 to 2 hairs). These kinds of sample are very hard to classify.

Other Unclear Image



The background of this sample is round, not square like other samples. We tried to add such samples by cropping the image in data augmentation but maybe there are still not enough samples like this, which caused the model classification error.



This image is too different from other samples. I think this is because of a wrong shot. The model does not have the ability to classify this kind of image, it can only guess.

Conclusion

The threshold greatly affects the final accuracy of the model. A too high threshold may cause many positive samples to be classified as negative. Using ROC to find the best threshold can solve this problem. However, while the dataset is too small, the best threshold on the training set and validation set may be different.

Some of the images in the dataset are unclear and they may be misclassified very easily. For example, Some images have lots of hairs, and others may be a round background.

We can reduce the error rate of predicting those unclear images by doing data augmentation. If we can simulate those unclear images and add them to the training set, the model will correctly classify those unclear images.