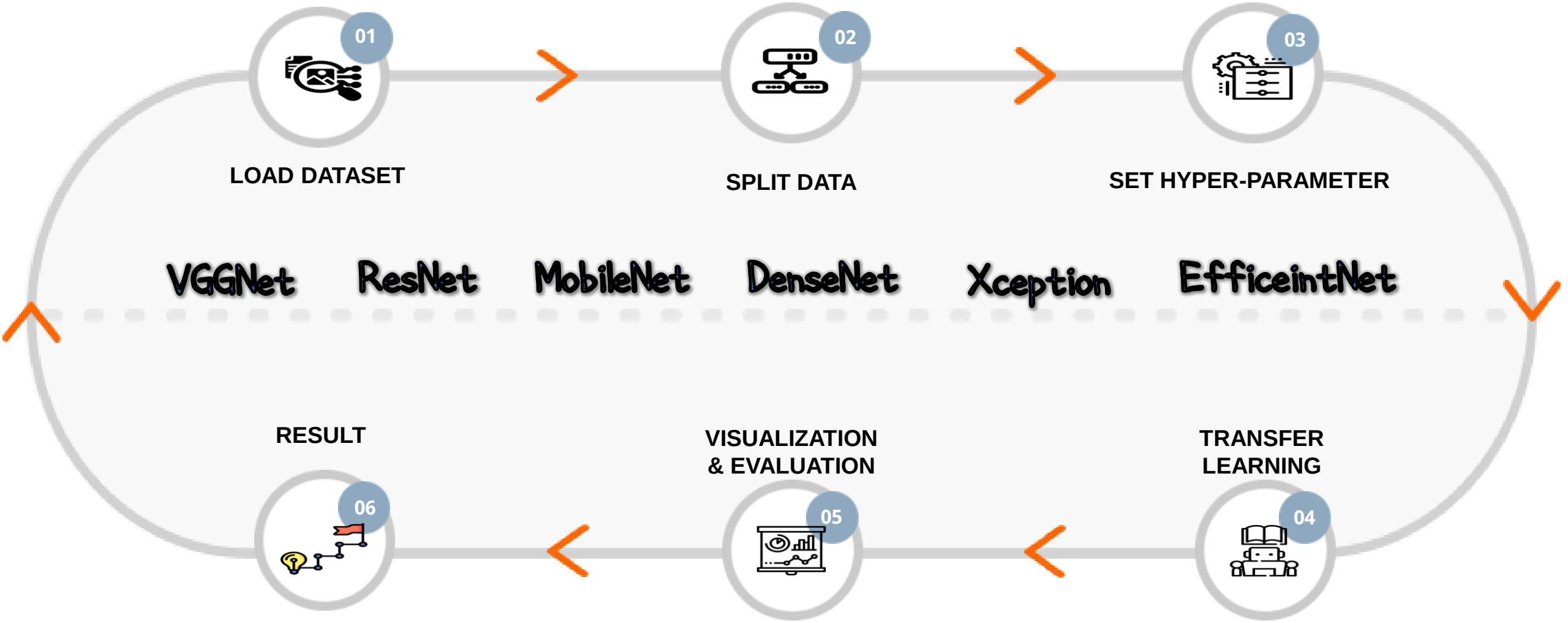
Leukemia Image Classification

- TRANSFER LEARNING
- HYPER PARAMETER TUNING


Outline

Outline


01
LOAD DATASET
02
SPLIT DATA
03
SET HYPER-PARAMETER
VGGNet
ResNet
MobileNet
DenseNet
Xception
EfficeintNet
RESULT
06
VISUALIZATION
& EVALUATION
05
TRANSFER
LEARNING
04

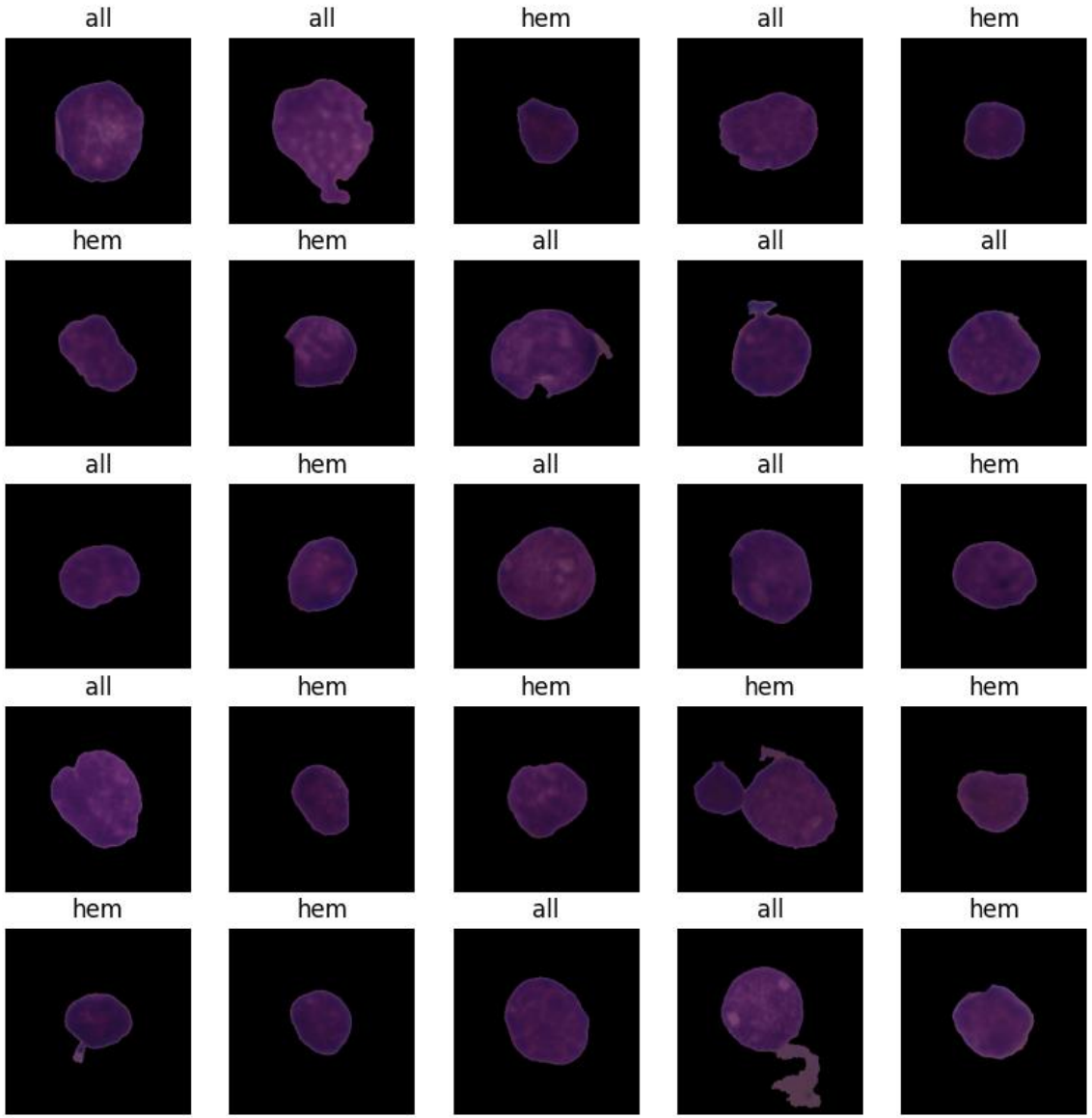

Dataset

1. DATASET :

LEUKEMIA IMAGE CLASSIFICATION

ALL VS HEM

- ALL : Leukemia blasts
- HEM : Normal Blood Cells



ALL (Leukemia blasts) VS HEM (Normal Blood Cells)


Preprocessing

Split Data from Training Dataset:

- Training data : 0.9
- Validation data : 0.05
- Test data : 0.05

```
[ ]  # Split data
     sdir=r'/content/drive/MyDrive/Computer Vision/Image Classification/4.Leukemia Image Classification/C-NMC_Leukemia/training_data'
     trsplit=.9
     vsplit=.05
     train_df, test_df, valid_df= preprocess(sdir,trsplit, vsplit)

     train_df length:  9594   test_df length:  534   valid_df length:  533
     [6644, 3050]
```

Make train data set balance, resize the image

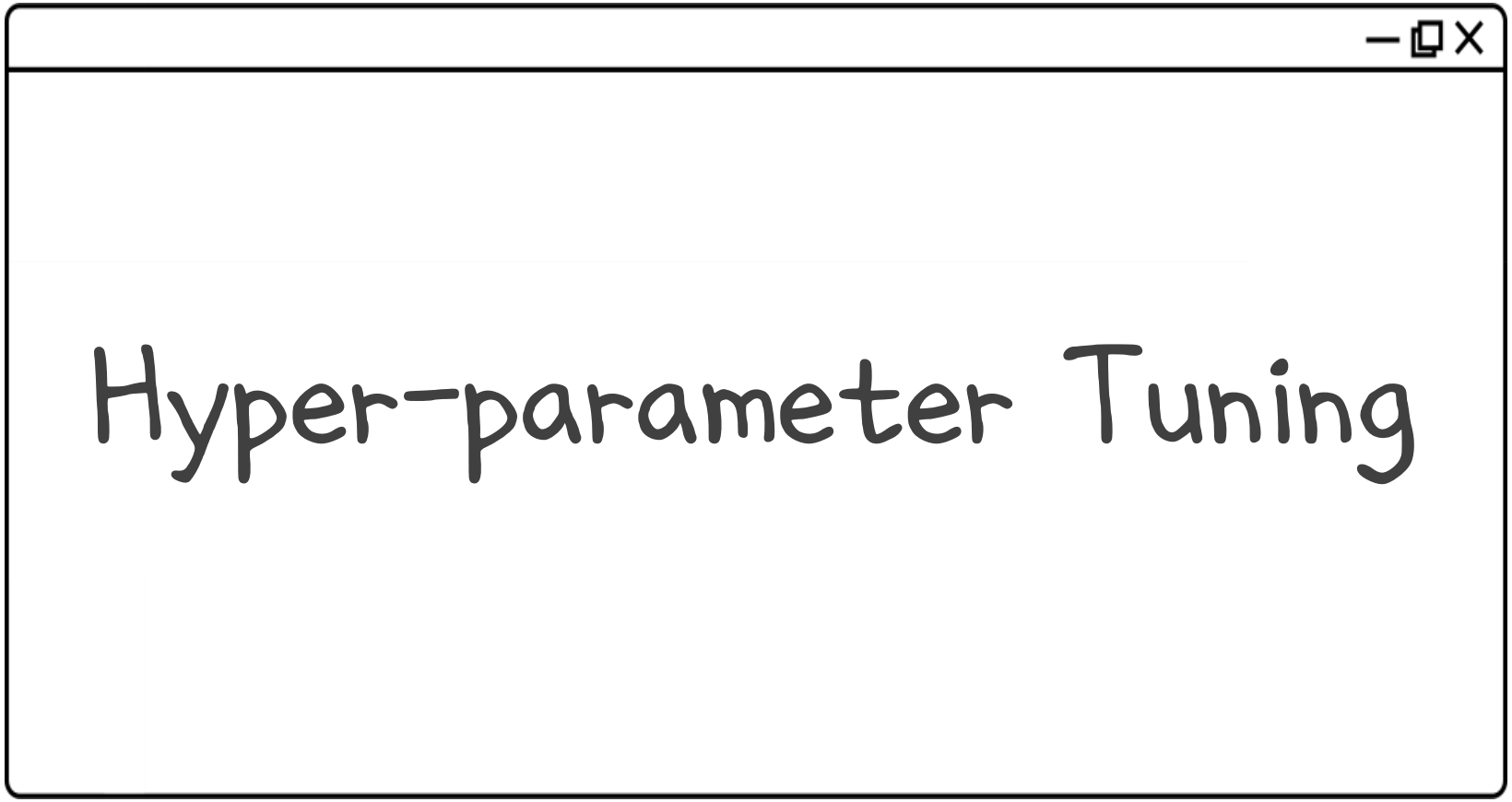
```
[ ]  # The train data set is not balanced. To balance it use the trim function defined above to limit the maximum samples in a class to 3050 samples
     max_samples= 3050
     min_samples=0
     column='labels'
     working_dir = r'./'
     img_size=(224,224)
     train_df=trim(train_df, max_samples, min_samples, column)
     # The train_df dataframe is now balanced with 140 samples per class

     Original Number of classes in dataframe:  2
     [3050, 3050]
```

Create train, test and validation generators

```
# Set Hyperparameter
channels = 3
batch_size = 32
img_shape=(img_size[0], img_size[1], channels)
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
test_steps=int(length/test_batch_size)
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps)
```

```
# Set Train, Test, Valid generators
def scalar(img):
    return img  # Network expects pixelsin range 0 to 255 so no scaling is required
trgen=ImageDataGenerator(preprocessing_function=scalar, horizontal_flip=True)
tvgen=ImageDataGenerator(preprocessing_function=scalar)
msg='                                                                    for the train generator'
print(msg, '\r', end='')
train_gen=trgen.flow_from_dataframe( train_df,
                                     x_col='filepaths',
                                     y_col='labels',
                                     target_size=img_size,
                                     class_mode='categorical',
                                     color_mode='rgb',
                                     shuffle=True,
                                     batch_size=batch_size)
msg='                                                                    for the test generator'
print(msg, '\r', end='')
test_gen=tvgen.flow_from_dataframe( test_df,
                                    x_col='filepaths',
                                    y_col='labels',
                                    target_size=img_size,
                                    class_mode='categorical',
                                    color_mode='rgb',
                                    shuffle=False,
                                    batch_size=test_batch_size)
msg='                                                                    for the validation generator'
print(msg, '\r', end='')
valid_gen=tvgen.flow_from_dataframe( valid_df,
                                     x_col='filepaths',
                                     y_col='labels',
                                     target_size=img_size,
                                     class_mode='categorical',
                                     color_mode='rgb',
                                     shuffle=True,
                                     batch_size=batch_size)
```

```
test batch size:  6   test steps:  89
Found 6100 validated image filenames belonging to 2 classes.
Found 534 validated image filenames belonging to 2 classes.
Found 533 validated image filenames belonging to 2 classes.
```


Hyper-parameter Tuning

Hyperparameter	
Epoch	20
Optimizer	Adam
Loss	Binary Cross Entropy
Activation Function	Sigmoid, ReLU
Regularization	L1(0.006), L2(0.016)
Dropout	0.5
Batch Normalization	Momentum(0.99), Epsilon(0.001)
Learning Rate	0.00001

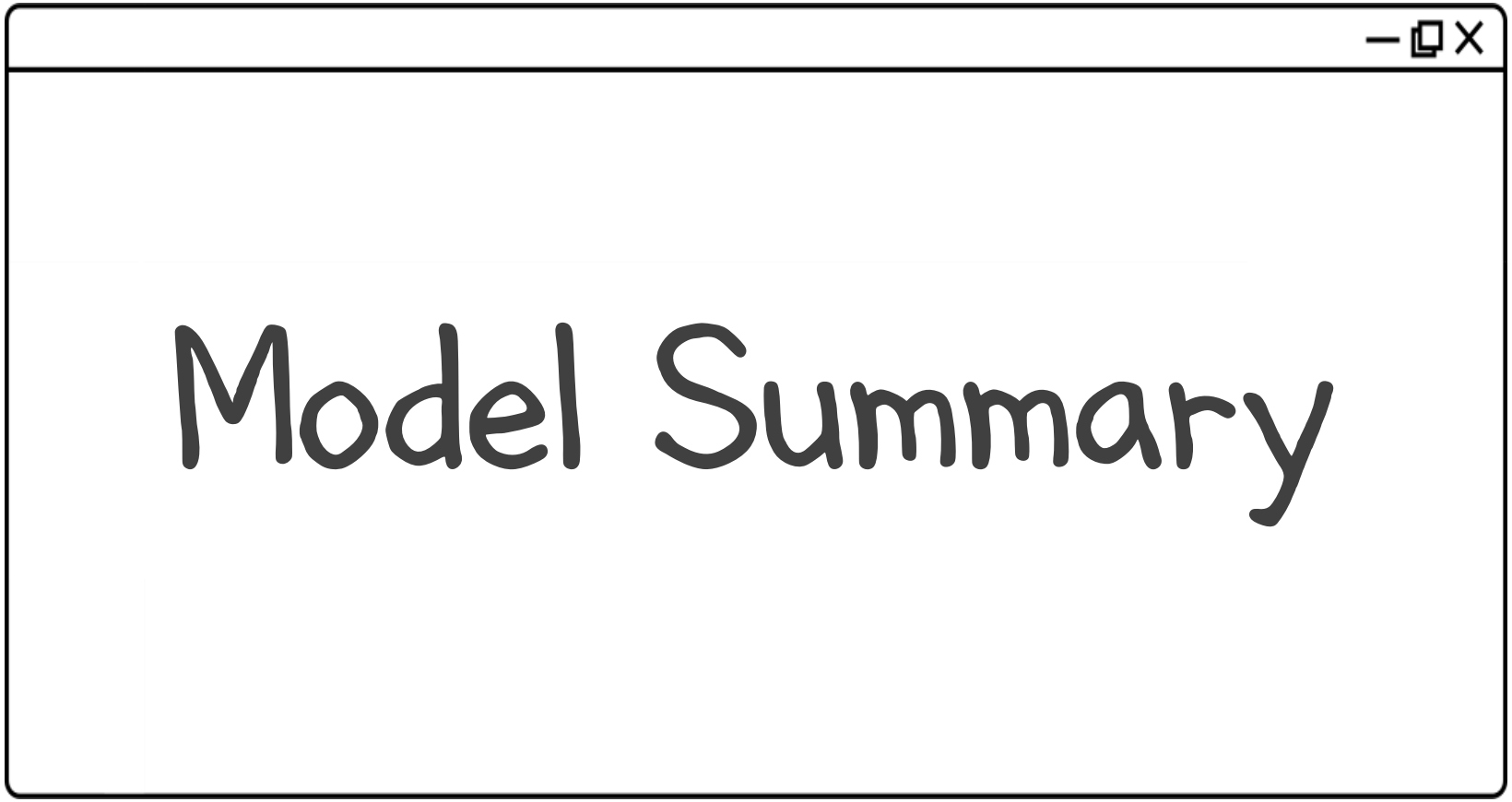
Hyper-parameter

Define Callbacks

```
[ ] class LRA(keras.callbacks.Callback):
        def __init__(self,model, base_model, patience,stop_patience, threshold, factor, dwell, batches, initial_epoch,epochs, ask_epoch):
            super(LRA, self).__init__()
            self.model=model
            self.base_model=base_model
            self.patience=patience # specifies how many epochs without improvement before learning rate is adjusted
            self.stop_patience=stop_patience # specifies how many times to adjust lr without improvement to stop training
            self.threshold=threshold # specifies training accuracy threshold when lr will be adjusted based on validation loss
            self.factor=factor # factor by which to reduce the learning rate
            self.dwell=dwell
            self.batches=batches # number of training batch to runn per epoch
            self.initial_epoch=initial_epoch
            self.epochs=epochs
            self.ask_epoch=ask_epoch
            self.ask_epoch_initial=ask_epoch # save this value to restore if restarting training

            # callback variables
            self.count=0 # how many times lr has been reduced without improvement
            self.stop_count=0
            self.best_epoch=1   # epoch with the lowest loss
            self.initial_lr=float(tf.keras.backend.get_value(model.optimizer.lr)) # get the initiallearning rate and save it
            self.highest_tracc=0.0 # set highest training accuracy to 0 initially
            self.lowest_vloss=np.inf # set lowest validation loss to infinity initially
            self.best_weights=self.model.get_weights() # set best weights to model's initial weights
            self.initial_weights=self.model.get_weights()   # save initial weights if they have to get restored
```

```
# Set Callback Option
patience= 10
stop_patience =3
threshold=.9
factor=.5
dwell=True
freeze=False
ask_epoch=20
batches=train_steps

callbacks=[LRA(model=model6,base_model= base_model,patience=patience,stop_patience=stop_patience, threshold=threshold,
                   factor=factor,dwell=dwell, batches=batches,initial_epoch=0,epochs=epochs, ask_epoch=ask_epoch )]
```

Callbacks


Model Summary

2. Model Summary

```
Model: "VGGNet19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080
...
Total params: 20,092,354
Trainable params: 20,091,330
Non-trainable params: 1,024
_________________________________________________________________
```

VGGNet : VGGNet19

```
Model: "ResNet50"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==================================================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3       0           []
                             )]

 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)       0           ['input_3[0][0]']

 conv1_conv (Conv2D)         (None, 112, 112, 64       9472        ['conv1_pad[0][0]']
                             )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64    256         ['conv1_conv[0][0]']
                             )

 conv1_relu (Activation)     (None, 112, 112, 64       0           ['conv1_bn[0][0]']
                             )

 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64       0           ['conv1_relu[0][0]']
                             )

 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)        0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2D)  (None, 56, 56, 64)      4160        ['pool1_pool[0][0]']

...
Total params: 23,858,434
Trainable params: 23,801,218
Non-trainable params: 57,216
__________________________________________________________________________________________________
```

ResNet : ResNet50

2. Model Summary

```
Model: "MobileNet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv1 (Conv2D)              (None, 112, 112, 32)      864

 conv1_bn (BatchNormalizatio  (None, 112, 112, 32)     128
 n)

 conv1_relu (ReLU)           (None, 112, 112, 32)      0

 conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)     288

 conv_dw_1_bn (BatchNormaliz  (None, 112, 112, 32)     128
 ation)

 conv_dw_1_relu (ReLU)       (None, 112, 112, 32)      0

 conv_pw_1 (Conv2D)          (None, 112, 112, 64)      2048

 conv_pw_1_bn (BatchNormaliz  (None, 112, 112, 64)     256
 ation)

...
Total params: 3,364,418
Trainable params: 3,340,482
Non-trainable params: 23,936
_________________________________________________________________
```

MobileNet : MobileNet

```
Model: "DenseNet121"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3  0          []
                             )]

 zero_padding2d (ZeroPadding2D)  (None, 230, 230, 3)  0          ['input_1[0][0]']

 conv1/conv (Conv2D)         (None, 112, 112, 64  9408        ['zero_padding2d[0][0]']
                             )

 conv1/bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1/conv[0][0]']
                             )

 conv1/relu (Activation)     (None, 112, 112, 64  0           ['conv1/bn[0][0]']
                             )

 zero_padding2d_1 (ZeroPadding2  (None, 114, 114, 64  0          ['conv1/relu[0][0]']
 D)                          )

 pool1 (MaxPooling2D)        (None, 56, 56, 64)   0           ['zero_padding2d_1[0][0]']

 conv2_block1_0_bn (BatchNormal  (None, 56, 56, 64)  256         ['pool1[0][0]']
 ization)
...
Total params: 7,173,058
Trainable params: 7,087,362
Non-trainable params: 85,696
__________________________________________________________________________________________________
```

DenseNet : DenseNet121

2. Model Summary

```
Model: "Xception"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_5 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 block1_conv1 (Conv2D)          (None, 111, 111, 32  864         ['input_5[0][0]']
                                )

 block1_conv1_bn (BatchNormaliz  (None, 111, 111, 32  128        ['block1_conv1[0][0]']
 ation)                         )

 block1_conv1_act (Activation)  (None, 111, 111, 32  0           ['block1_conv1_bn[0][0]']
                                )

 block1_conv2 (Conv2D)          (None, 109, 109, 64  18432       ['block1_conv1_act[0][0]']
                                )

 block1_conv2_bn (BatchNormaliz  (None, 109, 109, 64  256        ['block1_conv2[0][0]']
 ation)                         )

 block1_conv2_act (Activation)  (None, 109, 109, 64  0           ['block1_conv2_bn[0][0]']
                                )

...
Total params: 21,132,202
Trainable params: 21,073,578
Non-trainable params: 58,624
__________________________________________________________________________________________________
```

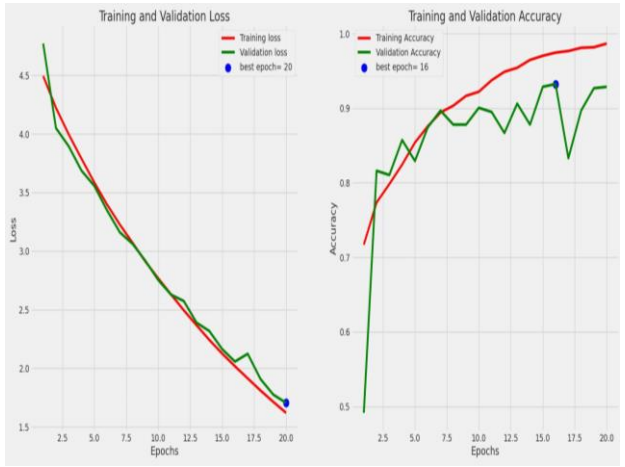
Xception : Xception

```
Model: "EfficientNetB1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 rescaling (Rescaling)          (None, 224, 224, 3)  0           ['input_2[0][0]']

 normalization (Normalization)  (None, 224, 224, 3)  7           ['rescaling[0][0]']

 rescaling_1 (Rescaling)        (None, 224, 224, 3)  0           ['normalization[0][0]']

 stem_conv_pad (ZeroPadding2D)  (None, 225, 225, 3)  0           ['rescaling_1[0][0]']

 stem_conv (Conv2D)             (None, 112, 112, 32  864         ['stem_conv_pad[0][0]']
                                )

 stem_bn (BatchNormalization)   (None, 112, 112, 32  128         ['stem_conv[0][0]']
                                )

 stem_activation (Activation)   (None, 112, 112, 32  0           ['stem_bn[0][0]']
                                )

 block1a_dwconv (DepthwiseConv2  (None, 112, 112, 32  288        ['stem_activation[0][0]']
...
Total params: 6,744,585
Trainable params: 6,679,970
Non-trainable params: 64,615
__________________________________________________________________________________________________
```

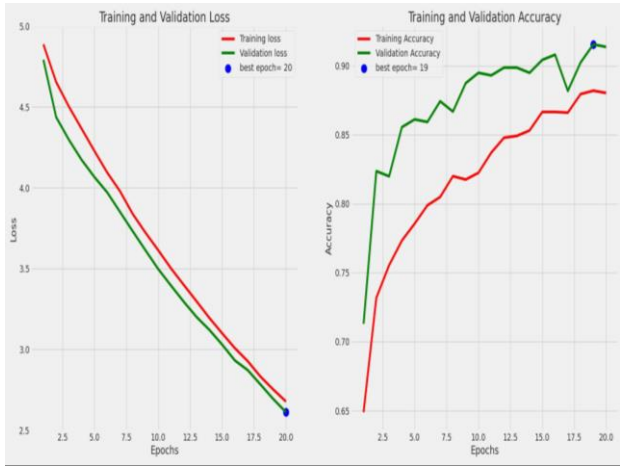
EfficientNet : EfficientNetB0


Results

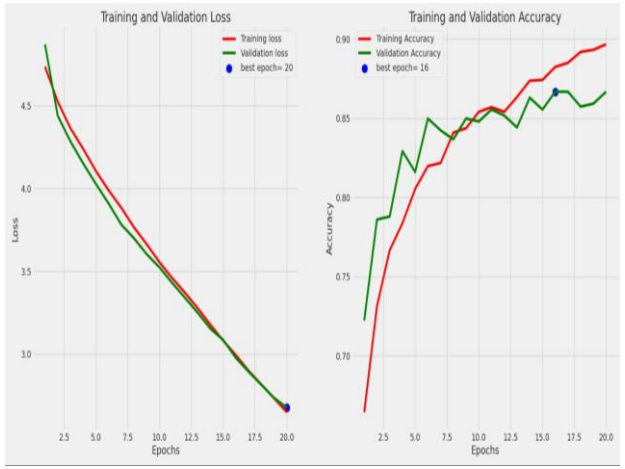
3. Results : Loss and Accuracy



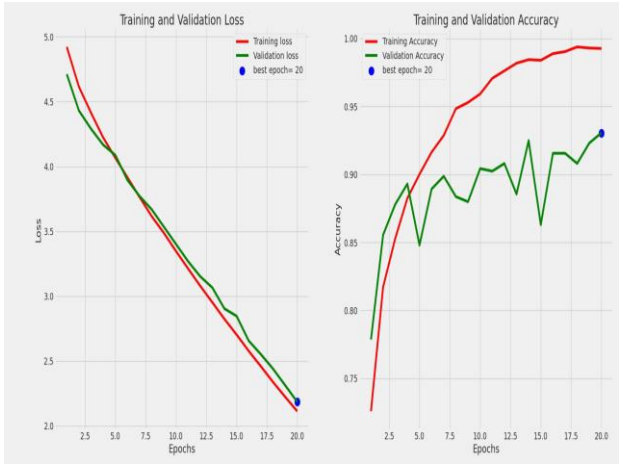
DenseNet121



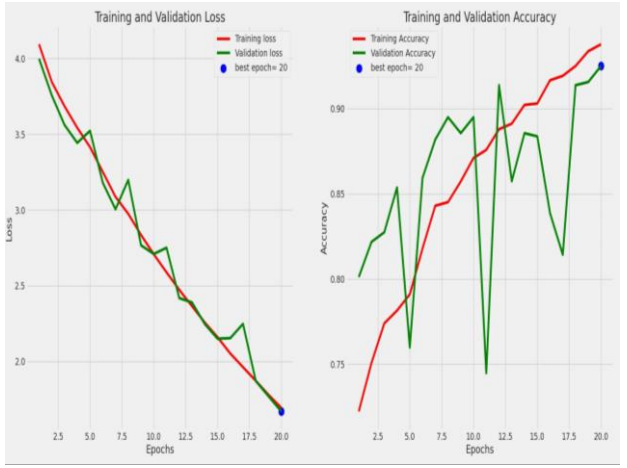
EfficientNetB1



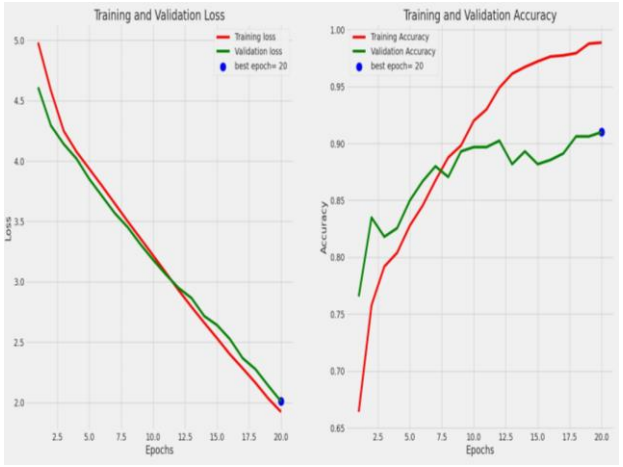
MobileNet



ResNet50



VGG19



Xception

3. Results : Classification Report

Classification Report:

	precision	recall	f1-score	support
all	0.9394	0.9368	0.9381	364
hem	0.8655	0.8706	0.8680	170
accuracy			0.9157	534
macro avg	0.9024	0.9037	0.9031	534
weighted avg	0.9159	0.9157	0.9158	534

DenseNet121

Classification Report:

	precision	recall	f1-score	support
all	0.9345	0.9011	0.9175	364
hem	0.8033	0.8647	0.8329	170
accuracy			0.8895	534
macro avg	0.8689	0.8829	0.8752	534
weighted avg	0.8927	0.8895	0.8905	534

EfficientNetB1

Classification Report:

	precision	recall	f1-score	support
all	0.9032	0.9231	0.9130	364
hem	0.8272	0.7882	0.8072	170
accuracy			0.8801	534
macro avg	0.8652	0.8557	0.8601	534
weighted avg	0.8790	0.8801	0.8794	534

MobileNet

Classification Report:

	precision	recall	f1-score	support
all	0.9339	0.9698	0.9515	364
hem	0.9295	0.8529	0.8896	170
accuracy			0.9326	534
macro avg	0.9317	0.9114	0.9205	534
weighted avg	0.9325	0.9326	0.9318	534

ResNet50

Classification Report:

	precision	recall	f1-score	support
all	0.9489	0.9176	0.9330	364
hem	0.8352	0.8941	0.8636	170
accuracy			0.9101	534
macro avg	0.8920	0.9059	0.8983	534
weighted avg	0.9127	0.9101	0.9109	534

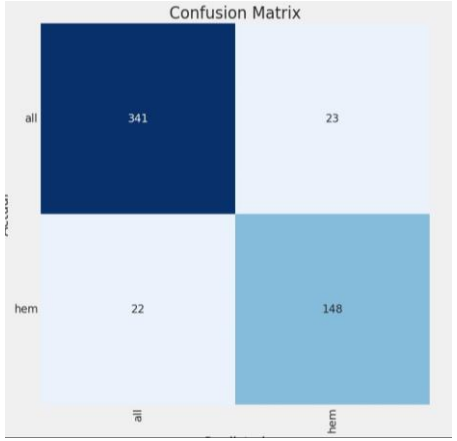
VGG19

Classification Report:

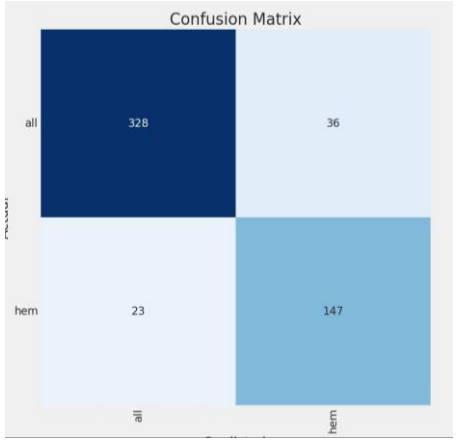
	precision	recall	f1-score	support
all	0.9096	0.9396	0.9243	364
hem	0.8608	0.8000	0.8293	170
accuracy			0.8951	534
macro avg	0.8852	0.8698	0.8768	534
weighted avg	0.8940	0.8951	0.8941	534

Xception

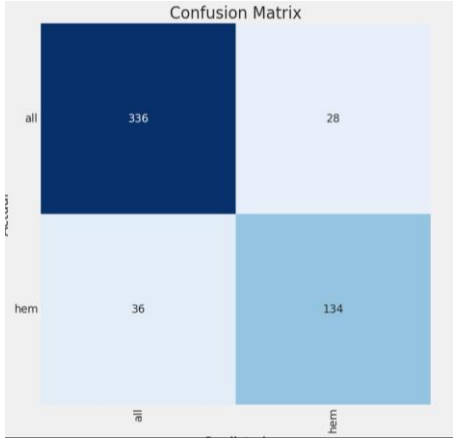
3. Results : Confusion Matrix



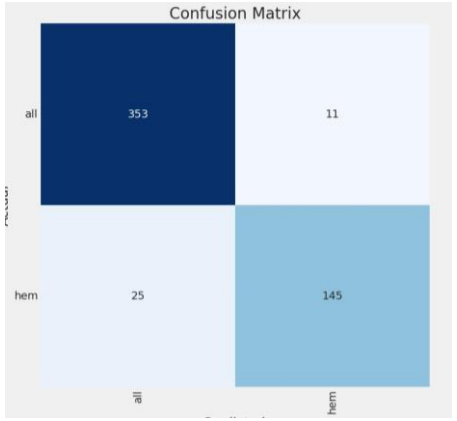
DenseNet121



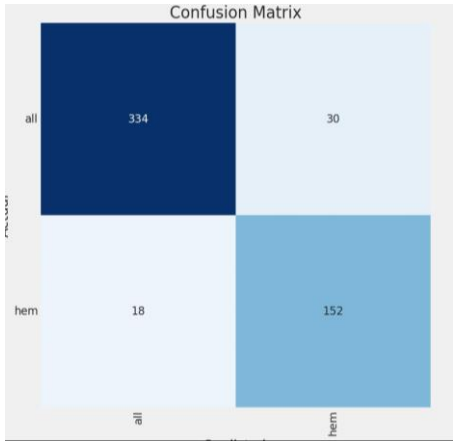
EfficientNEtB1



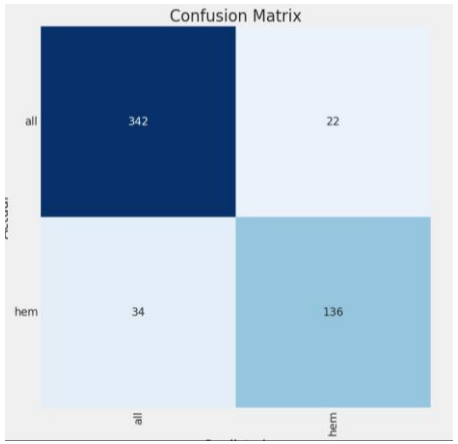
MobileNet



ResNet50



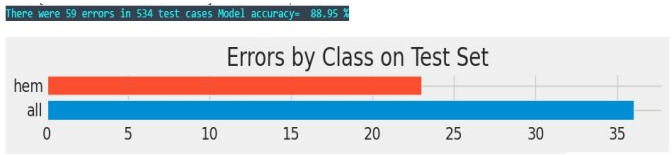
VGG19



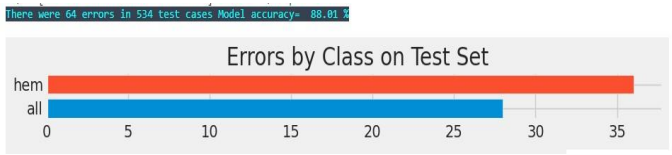
Xception

3. Results : Number of Errors

DenseNet121

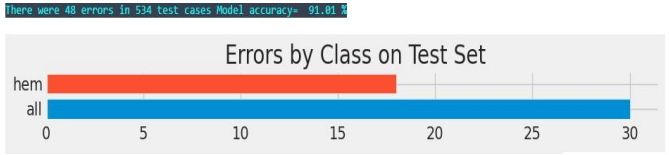


EfficientNEtB1



MobileNet

ResNet50



VGG19

Xception


Analyzation

4. Analyzation : Loss and Accuracy

- Result Analysis 1 : Loss and Accuracy
- Training Loss and Accuracy, Evaluation Loss and Accuracy for each model
- Top 3 Models : ResNet50, DenseNet121, VGGNet19

	Train_Loss	Eval_Loss	Train_Accuracy	Eval_Accuracy
DenseNet121	1.6205	1.7057	0.9867	0.9287
MobileNet	2.6480	2.6754	0.8967	0.8668
EfficientNetB1	2.6775	2.6113	0.8803	0.9137
ResNet50	2.1126	2.1845	0.9926	0.9306
VGGNet19	1.6934	1.6692	0.9377	0.9250
Xception	1.9231	2.0096	0.9887	0.9099

Results : Loss and Accuracy

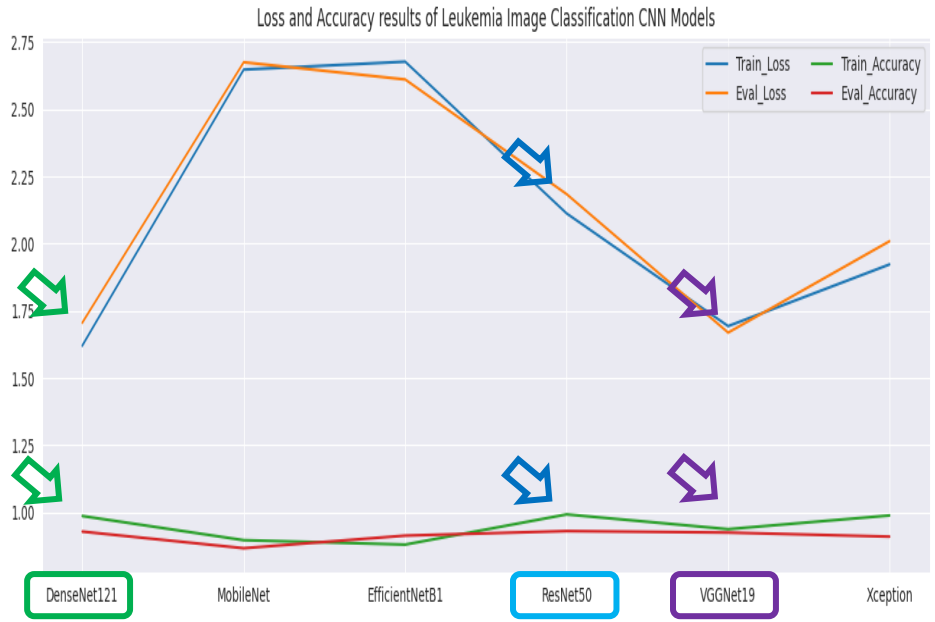
- Result Analysis 2 : Precision, Recall, F1-Score
- Precision, Recall, F1-Score of ALL vs Precision, Recall, F1-Score of HEM
- Top 3 Models : ResNet50, DenseNet121, VGGNet19

	Precision(ALL)	Recall(ALL)	F1-Score(ALL)	Precision(HEM)	Recall(HEM)	F1-Score(HEM)
DenseNet121	0.9394	0.9368	0.9381	0.8655	0.8706	0.8680
MobileNet	0.9032	0.9231	0.9130	0.8272	0.7882	0.8072
EfficientNetB1	0.9345	0.9011	0.9175	0.8033	0.8647	0.8329
ResNet50	0.9339	0.9698	0.9515	0.9295	0.8529	0.8896
VGGNet19	0.9489	0.9176	0.9330	0.8352	0.8941	0.8636
Xception	0.9096	0.9396	0.9243	0.8608	0.8000	0.8293

Results : Precision, Recall, F1-score (ALL & HEM)

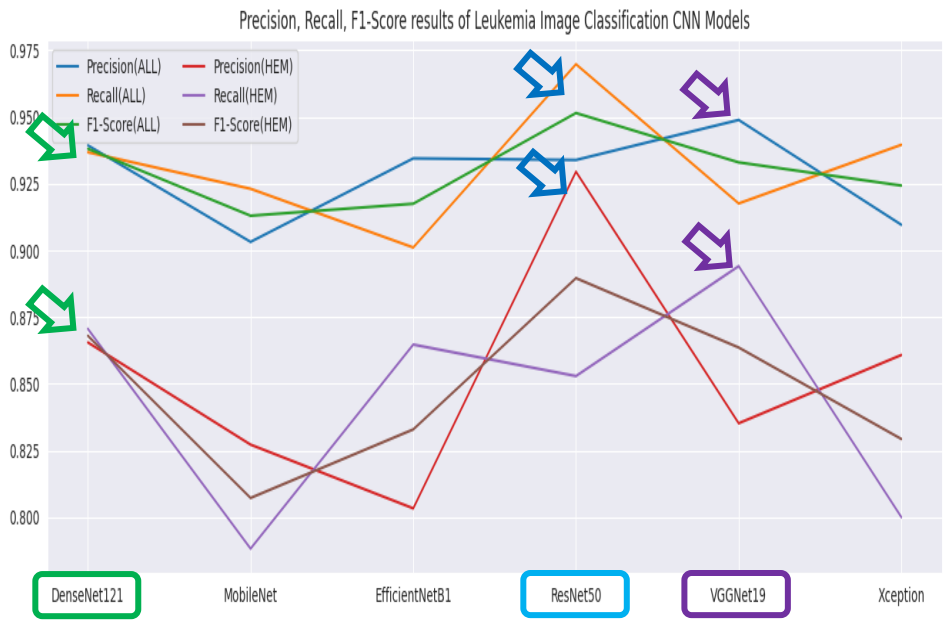
4. Analyzation : Loss and Accuracy

- Result Analysis 1 : Loss and Accuracy
- Top 3 Models : ResNet50, DenseNet121, VGGNet19



Results : Loss and Accuracy

- Result Analysis 2 : Precision, Recall, F1-Score
- Top 3 Models : ResNet50, DenseNet121, VGGNet19



Results : Precision, Recall, F1-score

THANK YOU

Contacts

- E-mail : johnnyworld9278@gmail.com
- GitHub : https://github.com/jpjp92