This Kaggle competition is a binary image classification problem where you will identify metastatic cancer in small image patches taken from larger digital pathology scans.

The project has 125 total points. The instructions summarize the criteria you will use to guide your submission and review others' submissions. Note: to receive total points for this section, the learner doesn't need to have a top-performing score on the challenge. As a mini-project to complete as a weekly assignment, we don't expect you to iterate over your project until you have a model capable of winning the challenge. The iterative process takes time, so please start early to get better-quality results and reports. The learner needs to show a score that reasonably reflects that they completed the rubric parts of this project. The grades are more based on the quality and depth of the analysis, not just on a better Kaggle score.

You will submit three deliverables:

Deliverable 1 — A Jupyter notebook with a description of the problem/data, exploratory data analysis (EDA) procedure, analysis (model building and training), result, and discussion/conclusion.

Suppose your work becomes so large that it doesn’t fit into one notebook (or you think it will be less readable by having one large notebook). In that case, you can make several notebooks or scripts in a GitHub repository (as deliverable 3) and submit a report-style notebook or pdf instead.

If your project doesn’t fit into Jupyter notebook format (E.g., you built an app that uses ML), write your approach as a report and submit it in a pdf form.

Deliverable 2 — A public project GitHub repository with your work (please also include the GitHub repo URL in your notebook/report).

Deliverable 3 — A screenshot of your position on the Kaggle competition leaderboard for your top-performing model.

github：https://github.com/zpeople/CNN_cancer_detection

[08/16/2024 Fri 10:56:39] BY Renmin Zhao

Step1

Brief description of the problem and data (5 pts)

Briefly describe the challenge problem and NLP. Describe the size, dimension, structure, etc., of the data

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
from glob import glob
from PIL import Image

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.metrics import roc_curve, auc, accuracy_score


from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [ ]:
```
#load data

labels_df = pd.read_csv("train_labels.csv")

labels_df.shape
labels_df.head()
```

Out[ ]: (220025, 2)

Out[ ]:

	id	label
0	f38a6374c348f90b587e046aac6079959adf3835	0
1	c18f2d887b7ae4f6742ee445113fa1aef383ed77	1
2	755db6279dae599ebb4d39a9123cce439965282d	0
3	bc3f0c64fb968ff4a8bd33af6971ecae77c75e08	0
4	068aba587a4950175d04c680d38943fd488d6a9d	0

In [ ]:
```
labels_df.describe()
```

Out[ ]:

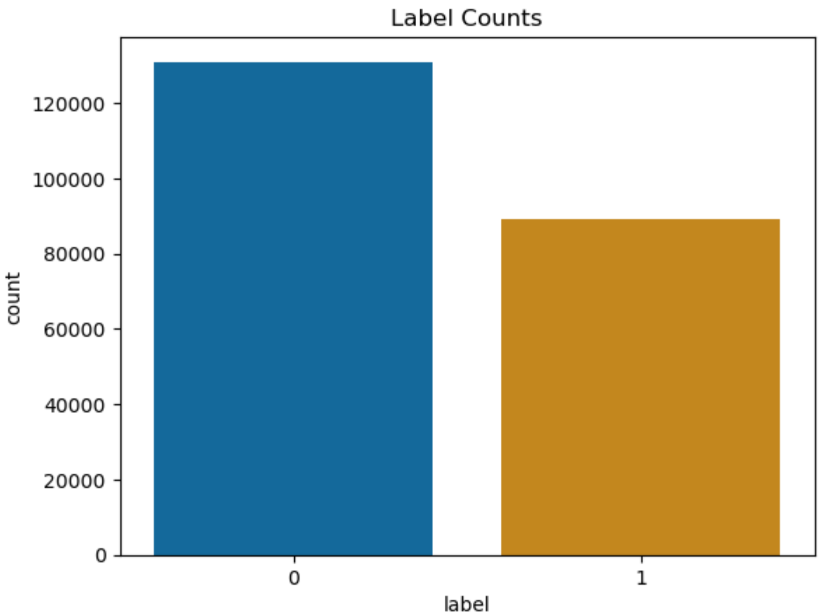
	label
count	220025.000000
mean	0.405031
std	0.490899
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

In [ ]:
```
labels_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220025 entries, 0 to 220024
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   id      220025 non-null  object
 1   label   220025 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 3.4+ MB
```

In [ ]:
```
print(pd.DataFrame(data={'Label Counts': labels_df['label'].value_counts()}))
sns.countplot(x=labels_df['label'], palette='colorblind').set(title='Label Counts');
```

```
       Label Counts
label              
0            130908
1             89117
```



In [ ]:
```
imgs_path_df = pd.DataFrame(glob("train/*.tif"), columns = ["path"])
imgs_path_df["id"] = imgs_path_df["path"].map(lambda x: x.split("\\")[-1].split(".")[0])
imgs_path_df.shape
imgs_path_df.head()
```

Out[ ]: (220025, 2)

Out[ ]:

	path	id
0	train\00001b2b5609af42ab0ab276dd4cd41c3e7745b5...	00001b2b5609af42ab0ab276dd4cd41c3e7745b5
1	train\000020de2aa6193f4c160e398a8edea95b1da598...	000020de2aa6193f4c160e398a8edea95b1da598
2	train\00004aab08381d25d315384d646f5ce413ea24b1...	00004aab08381d25d315384d646f5ce413ea24b1
3	train\0000d563d5cfafc4e68acb7c9829258a298d9b6a...	0000d563d5cfafc4e68acb7c9829258a298d9b6a
4	train\0000da768d06b879e5754c43e2298ce48726f722...	0000da768d06b879e5754c43e2298ce48726f722

In [ ]:
```
#merge data
df = pd.merge(imgs_path_df, labels_df, on = "id", how = "left")
df.shape
df.head()
```

Out[ ]: (220025, 3)

Out[ ]:

	path	id	label
0	train\00001b2b5609af42ab0ab276dd4cd41c3e7745b5...	00001b2b5609af42ab0ab276dd4cd41c3e7745b5	1
1	train\000020de2aa6193f4c160e398a8edea95b1da598...	000020de2aa6193f4c160e398a8edea95b1da598	0
2	train\00004aab08381d25d315384d646f5ce413ea24b1...	00004aab08381d25d315384d646f5ce413ea24b1	0
3	train\0000d563d5cfafc4e68acb7c9829258a298d9b6a...	0000d563d5cfafc4e68acb7c9829258a298d9b6a	0
4	train\0000da768d06b879e5754c43e2298ce48726f722...	0000da768d06b879e5754c43e2298ce48726f722	1

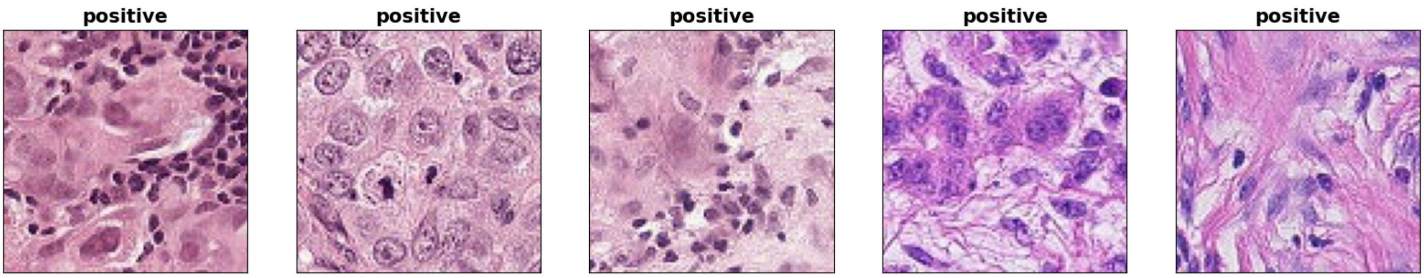
Step2

Exploratory Data Analysis (EDA) — Inspect, Visualize and Clean the Data (15 pts)

Show a few visualizations like histograms. Describe any data cleaning procedures. Based on your EDA, what is your plan of analysis?

In [ ]:
```
#Visualize image
def drawImage(df,title):
    fig, ax = plt.subplots(1, 5, figsize = (20, 5))

    random_num = list(np.random.randint(0, df.shape[0] - 1, size = 5))
    for i,j in enumerate(random_num):
        image_path =df.iloc[j]["path"]
        print(image_path)
        image_data =np.asarray(Image.open(image_path))
        print('Image shape:',image_data.shape) #(batch_size, height, width, channels)
        ax[i].imshow(image_data)
        ax[i].set_xticks([])

        ax[i].set_yticks([])
        ax[i].set_title(title,fontsize = 15, fontweight = "bold")


df_1 =df[df["label"] == 1]
df_0 =df[df["label"] == 0]
drawImage(df_1,'positive')
drawImage(df_0,'negative')
```

```
train\d616059f2df97b4398730901cf1c2f55fe2f4b1b.tif
Image shape: (96, 96, 3)
train\9bdca69670872c68f01d34f2aba4cd4d5bad104f.tif
Image shape: (96, 96, 3)
train\e9615b9803ff845aa44ea0bc8dd6f5afa3287c3e.tif
Image shape: (96, 96, 3)
train\d9b958cb0698e88e1fec349581b4da4b01e9fca1.tif
Image shape: (96, 96, 3)
train\c6b25a357d155861dc4d3de72b13a7c8dfd4fa19.tif
Image shape: (96, 96, 3)
train\8ae95c5ba2299fd0ce43cf5467042ccda9c293aa.tif
Image shape: (96, 96, 3)
train\327856e8014e563211fd8dac354d09142e6cdd79.tif
Image shape: (96, 96, 3)
train\ec9aaf0b32d088671007547bb74285b8d10e4134.tif
Image shape: (96, 96, 3)
train\15831c3315a7cb98621be8c82209823c2159ebc4.tif
Image shape: (96, 96, 3)
train\6ad8ce88201a827295edc2696aef28f11224b927.tif
Image shape: (96, 96, 3)
```



In [ ]:
```
#split train and valid data
index = np.random.permutation(df.shape[0])
df = df.iloc[index]
```

```
train_count =12000 #Adjust the training data for performance
df[:train_count]["label"].value_counts(normalize=True)

train, valid = train_test_split(df[:train_count], test_size = 0.2, stratify = df[:train_count]["label"], random_state = 0)

train.shape
```

Out[ ]:
```
label
0    0.60075
1    0.39925
Name: proportion, dtype: float64
```

Out[ ]:
```
(9600, 3)
```

Step3

DModel Architecture (25 pts)

escribe your model architecture and reasoning for why you believe that specific architecture would be suitable for this problem. Compare multiple architectures and tune hyperparameters.

In [ ]:
```
import tensorflow as tf

from tensorflow.keras import layers, models, Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, BatchNormalization, Activation


from tensorflow.keras import optimizers
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras import backend as K

RANDOM_STATE = 42
BATCH_SIZE = 96
TARGET_SIZE =(96,96)   # original image (96, 96)  resize will increases the duration of load
INPUT_SHAPE=(96,96,3)
ROC = tf.keras.metrics.AUC()
EPOCH =20


train["label"] = train["label"].astype(str)
valid["label"] = valid["label"].astype(str)



# Pixel values are normalized to 0-1, while data augmentation methods such as horizontal, vertical, rotation, and scaling transformations are
train_datagen = ImageDataGenerator(
    rescale = 1 / 255,
    vertical_flip = True,
    horizontal_flip = True,
    rotation_range = 90,
    zoom_range = 0.2,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    shear_range = 0.05,
    channel_shift_range = 0.1
)

train_generator = train_datagen.flow_from_dataframe(
    dataframe = train,
    directory = None,
    x_col = "path",
    y_col = "label",
    target_size =TARGET_SIZE,
    class_mode = "binary",
    batch_size = BATCH_SIZE,
    seed = RANDOM_STATE,
    shuffle = True
)


valid_datagen = ImageDataGenerator(
    rescale = 1 / 255
)
valid_generator = valid_datagen.flow_from_dataframe(
    dataframe = valid,
    directory = None,
    x_col = "path",
    y_col = "label",
    target_size = TARGET_SIZE,
    class_mode = "binary",
    batch_size = BATCH_SIZE,
    seed = RANDOM_STATE,
    shuffle = False
)
```

```
Found 9600 validated image filenames belonging to 2 classes.
Found 2400 validated image filenames belonging to 2 classes.
```

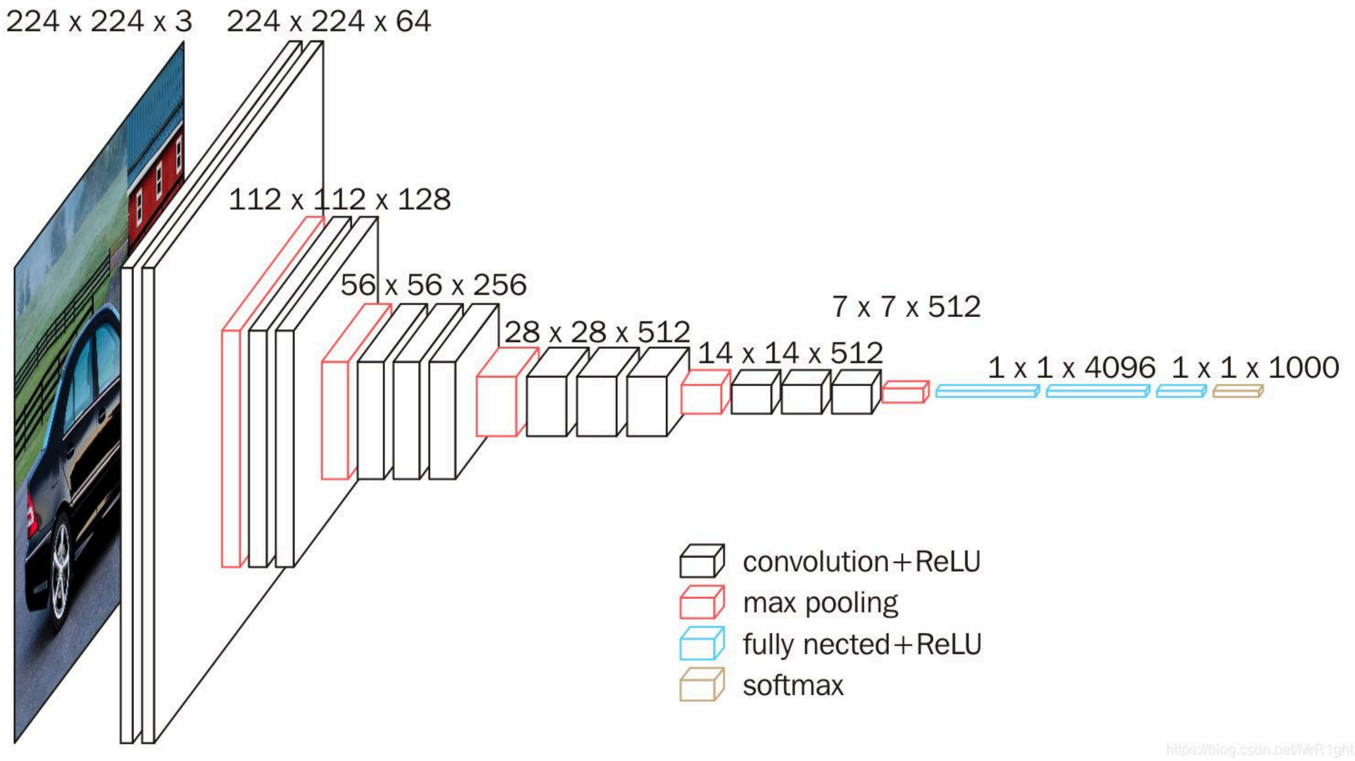
In [ ]:
```python
# Optional: Restrict TensorFlow to only use the first GPU
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) >0:
    print(physical_devices)
    try:

        tf.config.set_visible_devices(devices=physical_devices[0], device_type='GPU')
    except:
        # Invalid device or cannot modify virtual devices once initialized.
        pass

logical_devices = tf.config.list_logical_devices('GPU')
print(f"Physical GPUs: {len(physical_devices)}, Logical GPUs: {len(logical_devices)}")
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
Physical GPUs: 1, Logical GPUs: 1
```

In [ ]:
```python
# common train config
train_step_size = train_generator.n // train_generator.batch_size
valid_step_size = valid_generator.n // valid_generator.batch_size
earlystopper = EarlyStopping(monitor = "val_loss", patience = 5, verbose = 1, restore_best_weights = True)
reducelr = ReduceLROnPlateau(monitor = "val_loss", patience = 3, verbose = 1, factor = 0.1)
```

MY_VGG16

The model have be done by using models.Model. VGG16 is a classical convolutional neural network architecture consisting of 16 layers (13 convolutional layers and 3 fully connected layers).



In [ ]:
```python
#VGG 16
import tensorflow as tf
K.clear_session()
# Define the input

inputs = Input(shape=INPUT_SHAPE)
# Block 1
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2), strides=(2, 2))(x)

# Block 2
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2), strides=(2, 2))(x)

# Block 3
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
```

```
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2), strides=(2, 2))(x)
x = layers.Dropout(0.5)(x)  # Add Dropout after Block 3

# Block 4
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2), strides=(2, 2))(x)

# Block 5
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2), strides=(2, 2))(x)
x = layers.Dropout(0.5)(x)  # Add Dropout after Block 5

# Flatten and Dense layers
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)  # Add Dropout after Dense layer
x = layers.Dense(256, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)  # Add Dropout after Dense layer
outputs = layers.Dense(1, activation = "sigmoid")(x)

# Create the model
my_VGG16 = models.Model(inputs=inputs, outputs=outputs)

# Print the model summary
my_VGG16.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 96, 96, 3)]       0

 conv2d (Conv2D)             (None, 96, 96, 64)        1792

 batch_normalization (BatchN  (None, 96, 96, 64)       256
 ormalization)

 conv2d_1 (Conv2D)           (None, 96, 96, 64)        36928

 batch_normalization_1 (Batc  (None, 96, 96, 64)       256
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 48, 48, 64)       0
 )

 conv2d_2 (Conv2D)           (None, 48, 48, 128)       73856

 batch_normalization_2 (Batc  (None, 48, 48, 128)      512
 hNormalization)

 conv2d_3 (Conv2D)           (None, 48, 48, 128)       147584

 batch_normalization_3 (Batc  (None, 48, 48, 128)      512
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 24, 24, 128)      0
 2D)

 conv2d_4 (Conv2D)           (None, 24, 24, 256)       295168

 batch_normalization_4 (Batc  (None, 24, 24, 256)      1024
 hNormalization)

 conv2d_5 (Conv2D)           (None, 24, 24, 256)       590080

 batch_normalization_5 (Batc  (None, 24, 24, 256)      1024
 hNormalization)

 conv2d_6 (Conv2D)           (None, 24, 24, 256)       590080

 batch_normalization_6 (Batc  (None, 24, 24, 256)      1024
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 dropout (Dropout)           (None, 12, 12, 256)       0

 conv2d_7 (Conv2D)           (None, 12, 12, 512)       1180160

 batch_normalization_7 (Batc  (None, 12, 12, 512)      2048
 hNormalization)

 conv2d_8 (Conv2D)           (None, 12, 12, 512)       2359808

 batch_normalization_8 (Batc  (None, 12, 12, 512)      2048
 hNormalization)

 conv2d_9 (Conv2D)           (None, 12, 12, 512)       2359808

 batch_normalization_9 (Batc  (None, 12, 12, 512)      2048
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 512)        0
 2D)

 conv2d_10 (Conv2D)          (None, 6, 6, 512)         2359808

 batch_normalization_10 (Bat  (None, 6, 6, 512)        2048
 chNormalization)

 conv2d_11 (Conv2D)          (None, 6, 6, 512)         2359808

 batch_normalization_11 (Bat  (None, 6, 6, 512)        2048
 chNormalization)

 conv2d_12 (Conv2D)          (None, 6, 6, 512)         2359808

 batch_normalization_12 (Bat  (None, 6, 6, 512)        2048
 chNormalization)

 max_pooling2d_4 (MaxPooling  (None, 3, 3, 512)        0
 2D)

 dropout_1 (Dropout)         (None, 3, 3, 512)         0

 flatten (Flatten)           (None, 4608)              0
```

```
 dense (Dense)               (None, 256)               1179904

 batch_normalization_13 (Bat  (None, 256)              1024
 chNormalization)

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 batch_normalization_14 (Bat  (None, 256)              1024
 chNormalization)

 dropout_3 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 1)                 257

=================================================================
Total params: 15,979,585
Trainable params: 15,970,113
Non-trainable params: 9,472
_________________________________________________________________
```

In [ ]:

```
lr =0.001
my_VGG16.compile(optimizers.Adam(lr), loss = "binary_crossentropy", metrics = ["accuracy",ROC])

vgg_history = my_VGG16.fit(train_generator, steps_per_epoch = train_step_size, epochs = EPOCH,
                                validation_data = valid_generator, validation_steps = valid_step_size,
                                callbacks = [reducelr,earlystopper], verbose = 1)
```

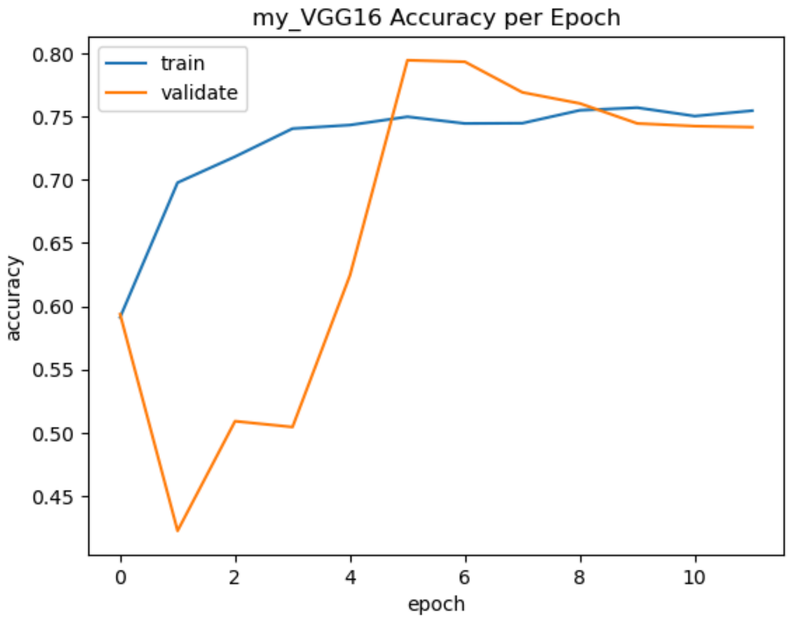
```
Epoch 1/20
100/100 [==============================] - 212s 2s/step - loss: 0.9116 - accuracy: 0.5914 - auc: 0.6292 - val_loss: 0.6750 - val_accuracy: 0.5
938 - val_auc: 0.5394 - lr: 0.0010
Epoch 2/20
100/100 [==============================] - 79s 785ms/step - loss: 0.7230 - accuracy: 0.6978 - auc: 0.7687 - val_loss: 0.8196 - val_accuracy:
0.4225 - val_auc: 0.5143 - lr: 0.0010
Epoch 3/20
100/100 [==============================] - 76s 755ms/step - loss: 0.6871 - accuracy: 0.7183 - auc: 0.7829 - val_loss: 2.0591 - val_accuracy:
0.5092 - val_auc: 0.5611 - lr: 0.0010
Epoch 4/20
100/100 [==============================] - ETA: 0s - loss: 0.6446 - accuracy: 0.7405 - auc: 0.8060
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
100/100 [==============================] - 75s 748ms/step - loss: 0.6446 - accuracy: 0.7405 - auc: 0.8060 - val_loss: 3.1843 - val_accuracy:
0.5046 - val_auc: 0.6635 - lr: 0.0010
Epoch 5/20
100/100 [==============================] - 75s 749ms/step - loss: 0.6172 - accuracy: 0.7433 - auc: 0.8120 - val_loss: 1.3031 - val_accuracy:
0.6246 - val_auc: 0.8428 - lr: 1.0000e-04
Epoch 6/20
100/100 [==============================] - 75s 745ms/step - loss: 0.6092 - accuracy: 0.7499 - auc: 0.8148 - val_loss: 0.5229 - val_accuracy:
0.7946 - val_auc: 0.8740 - lr: 1.0000e-04
Epoch 7/20
100/100 [==============================] - 76s 752ms/step - loss: 0.6097 - accuracy: 0.7446 - auc: 0.8161 - val_loss: 0.4657 - val_accuracy:
0.7933 - val_auc: 0.8670 - lr: 1.0000e-04
Epoch 8/20
100/100 [==============================] - 76s 754ms/step - loss: 0.6228 - accuracy: 0.7448 - auc: 0.8117 - val_loss: 0.5062 - val_accuracy:
0.7692 - val_auc: 0.8562 - lr: 1.0000e-04
Epoch 9/20
100/100 [==============================] - 75s 747ms/step - loss: 0.5967 - accuracy: 0.7550 - auc: 0.8214 - val_loss: 0.5279 - val_accuracy:
0.7604 - val_auc: 0.8532 - lr: 1.0000e-04
Epoch 10/20
100/100 [==============================] - ETA: 0s - loss: 0.5943 - accuracy: 0.7571 - auc: 0.8231
Epoch 10: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
100/100 [==============================] - 75s 747ms/step - loss: 0.5943 - accuracy: 0.7571 - auc: 0.8231 - val_loss: 0.5570 - val_accuracy:
0.7446 - val_auc: 0.8449 - lr: 1.0000e-04
Epoch 11/20
100/100 [==============================] - 75s 747ms/step - loss: 0.5953 - accuracy: 0.7504 - auc: 0.8212 - val_loss: 0.5665 - val_accuracy:
0.7425 - val_auc: 0.8426 - lr: 1.0000e-05
Epoch 12/20
100/100 [==============================] - ETA: 0s - loss: 0.5933 - accuracy: 0.7547 - auc: 0.8223Restoring model weights from the end of the
best epoch: 7.
100/100 [==============================] - 76s 753ms/step - loss: 0.5933 - accuracy: 0.7547 - auc: 0.8223 - val_loss: 0.5697 - val_accuracy:
0.7417 - val_auc: 0.8419 - lr: 1.0000e-05
Epoch 12: early stopping
```

In [ ]:

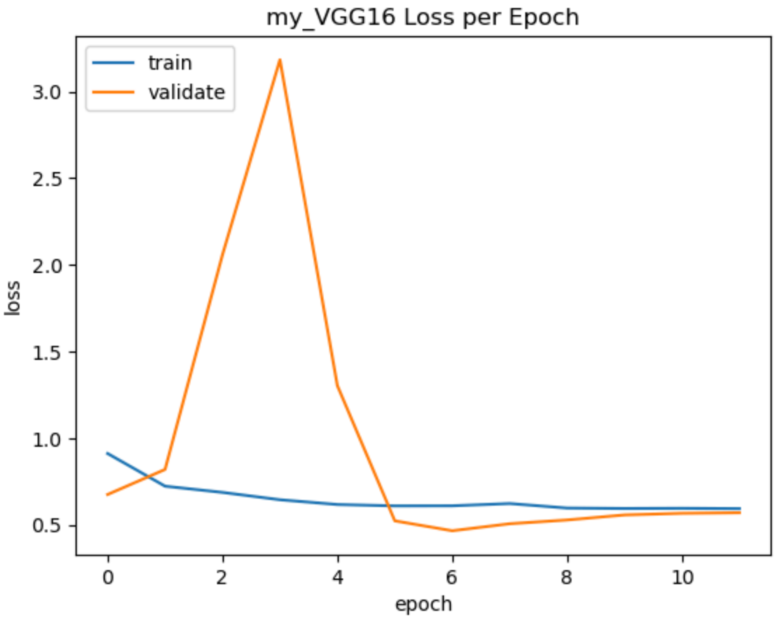
```
# plot model accuracy per epoch
plt.plot(vgg_history.history['accuracy'])
plt.plot(vgg_history.history['val_accuracy'])
plt.title('my_VGG16 Accuracy per Epoch')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc=0)
plt.show()

# plot model loss per epoch
plt.plot(vgg_history.history['loss'])
plt.plot(vgg_history.history['val_loss'])
plt.title('my_VGG16 Loss per Epoch')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc=0)
plt.show()
```

```python
# plot model ROC per epoch
plt.plot(vgg_history.history['auc'])
plt.plot(vgg_history.history['val_auc'])
plt.title('my_VGG16 AUC ROC per Epoch')
plt.ylabel('ROC')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc=0)
plt.show()
```

```
Out[ ]: [<matplotlib.lines.Line2D at 0x23dc247a8f0>]
Out[ ]: [<matplotlib.lines.Line2D at 0x23dc247ab60>]
Out[ ]: Text(0.5, 1.0, 'my_VGG16 Accuracy per Epoch')
Out[ ]: Text(0, 0.5, 'accuracy')
Out[ ]: Text(0.5, 0, 'epoch')
Out[ ]: <matplotlib.legend.Legend at 0x23cabbbc130>
```



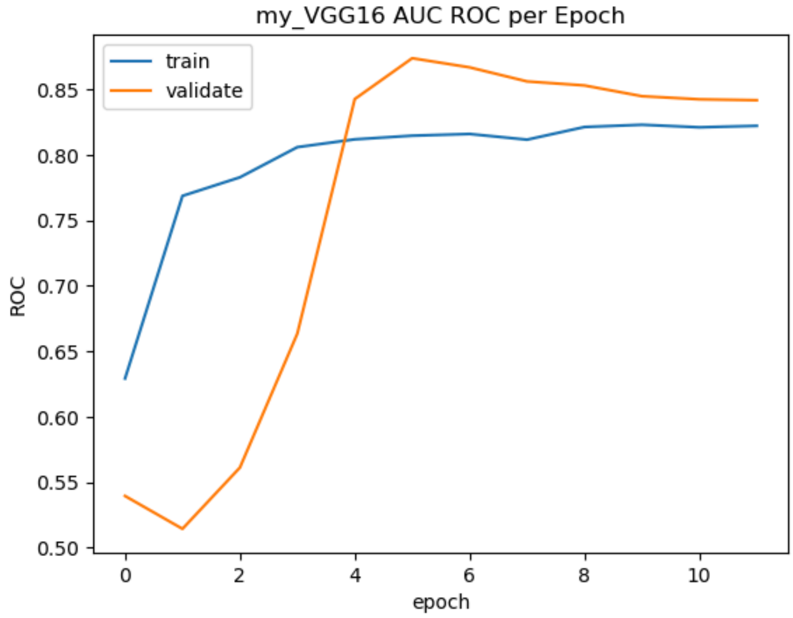

```
Out[ ]: [<matplotlib.lines.Line2D at 0x23dc25300a0>]
Out[ ]: [<matplotlib.lines.Line2D at 0x23dc2530430>]
Out[ ]: Text(0.5, 1.0, 'my_VGG16 Loss per Epoch')
Out[ ]: Text(0, 0.5, 'loss')
Out[ ]: Text(0.5, 0, 'epoch')
Out[ ]: <matplotlib.legend.Legend at 0x23dc24c9cf0>
```




```
Out[ ]: [<matplotlib.lines.Line2D at 0x23dc24ca4a0>]
Out[ ]: [<matplotlib.lines.Line2D at 0x23dc24ca920>]
Out[ ]: Text(0.5, 1.0, 'my_VGG16 AUC ROC per Epoch')
Out[ ]: Text(0, 0.5, 'ROC')
```

Out[ ]: Text(0.5, 0, 'epoch')

Out[ ]: <matplotlib.legend.Legend at 0x23dc242a440>



ResNET 34

ResNet-34 model specifically refers to a version of ResNet with 34 layers. It consists of several residual blocks, each containing skip connections that add the input directly to the output of the block. This allows the gradient to flow through the network more easily during backpropagation, facilitating better learning and reducing the degradation problem that arises when adding more layers to a network.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

In [ ]:
```
def residual_block(input_tensor, filters, stride=1):
    x = layers.Conv2D(filters, (3, 3), strides=stride, padding='same')(input_tensor)
    x = layers.BatchNormalization()(x)
    x = layers.ReLU()(x)

    x = layers.Conv2D(filters, (3, 3), strides=1, padding='same')(x)
    x = layers.BatchNormalization()(x)

    if stride != 1 or input_tensor.shape[-1] != filters:
        input_tensor = layers.Conv2D(filters, (1, 1), strides=stride, padding='same')(input_tensor)
        input_tensor = layers.BatchNormalization()(input_tensor)

    x = layers.add([x, input_tensor])
    x = layers.ReLU()(x)
    return x

def build_resnet34(input_shape):
    inputs = layers.Input(shape=input_shape)

    # Initial convolution and pooling layers
    x = layers.Conv2D(64, (7, 7), strides=2, padding='same')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.ReLU()(x)
    x = layers.MaxPooling2D((3, 3), strides=2, padding='same')(x)

    # Residual block configuration (64, 64, 128, 128, 256, 256, 512, 512)
    filter_sizes = [64, 64, 128, 128, 256, 256, 512, 512]
    strides = [1, 1, 2, 1, 2, 1, 2, 1]
    repetitions = [3, 4, 6, 3]
```

```python
    for filters, stride, repeat in zip(filter_sizes, strides, repetitions):
        for _ in range(repeat):
            x = residual_block(x, filters, stride)
            stride = 1

    # Global average pooling and output layer
    x = layers.GlobalAveragePooling2D()(x)
    outputs = layers.Dense(1, activation='sigmoid')(x)

    model = models.Model(inputs, outputs)
    return model
```

In [ ]:

```python
K.clear_session()
resnet = build_resnet34(INPUT_SHAPE)

resnet.summary()
```

Model: "model"

```
 Layer (type)                   Output Shape          Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 96, 96, 3)]  0           []

 conv2d (Conv2D)                (None, 48, 48, 64)   9472        ['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 48, 48, 64)  256         ['conv2d[0][0]']
 alization)

 Layer (type)                   Output Shape          Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 96, 96, 3)]  0           []

 conv2d (Conv2D)                (None, 48, 48, 64)   9472        ['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 48, 48, 64)  256         ['conv2d[0][0]']
 alization)

 re_lu (ReLU)                   (None, 48, 48, 64)   0           ['batch_normalization[0][0]']

 max_pooling2d (MaxPooling2D)   (None, 24, 24, 64)   0           ['re_lu[0][0]']

 conv2d_1 (Conv2D)              (None, 24, 24, 64)   36928       ['max_pooling2d[0][0]']

 batch_normalization_1 (BatchNo  (None, 24, 24, 64)  256         ['conv2d_1[0][0]']
 rmalization)

 re_lu_1 (ReLU)                 (None, 24, 24, 64)   0           ['batch_normalization_1[0][0]']

 conv2d_2 (Conv2D)              (None, 24, 24, 64)   36928       ['re_lu_1[0][0]']

 batch_normalization_2 (BatchNo  (None, 24, 24, 64)  256         ['conv2d_2[0][0]']
 rmalization)

 add (Add)                      (None, 24, 24, 64)   0           ['batch_normalization_2[0][0]',
                                                                  'max_pooling2d[0][0]']

 re_lu_2 (ReLU)                 (None, 24, 24, 64)   0           ['add[0][0]']

 conv2d_3 (Conv2D)              (None, 24, 24, 64)   36928       ['re_lu_2[0][0]']

 batch_normalization_3 (BatchNo  (None, 24, 24, 64)  256         ['conv2d_3[0][0]']
 rmalization)

 re_lu_3 (ReLU)                 (None, 24, 24, 64)   0           ['batch_normalization_3[0][0]']

 conv2d_4 (Conv2D)              (None, 24, 24, 64)   36928       ['re_lu_3[0][0]']

 batch_normalization_4 (BatchNo  (None, 24, 24, 64)  256         ['conv2d_4[0][0]']
 rmalization)

 add_1 (Add)                    (None, 24, 24, 64)   0           ['batch_normalization_4[0][0]',
                                                                  're_lu_2[0][0]']

 re_lu_4 (ReLU)                 (None, 24, 24, 64)   0           ['add_1[0][0]']

 conv2d_5 (Conv2D)              (None, 24, 24, 64)   36928       ['re_lu_4[0][0]']

 batch_normalization_5 (BatchNo  (None, 24, 24, 64)  256         ['conv2d_5[0][0]']
 rmalization)

 re_lu_5 (ReLU)                 (None, 24, 24, 64)   0           ['batch_normalization_5[0][0]']

 conv2d_6 (Conv2D)              (None, 24, 24, 64)   36928       ['re_lu_5[0][0]']

 batch_normalization_6 (BatchNo  (None, 24, 24, 64)  256         ['conv2d_6[0][0]']
 rmalization)

 add_2 (Add)                    (None, 24, 24, 64)   0           ['batch_normalization_6[0][0]',
                                                                  're_lu_4[0][0]']

 re_lu_6 (ReLU)                 (None, 24, 24, 64)   0           ['add_2[0][0]']

 conv2d_7 (Conv2D)              (None, 24, 24, 64)   36928       ['re_lu_6[0][0]']

 batch_normalization_7 (BatchNo  (None, 24, 24, 64)  256         ['conv2d_7[0][0]']
 rmalization)

 re_lu_7 (ReLU)                 (None, 24, 24, 64)   0           ['batch_normalization_7[0][0]']

 conv2d_8 (Conv2D)              (None, 24, 24, 64)   36928       ['re_lu_7[0][0]']

 batch_normalization_8 (BatchNo  (None, 24, 24, 64)  256         ['conv2d_8[0][0]']
 rmalization)

 add_3 (Add)                    (None, 24, 24, 64)   0           ['batch_normalization_8[0][0]',
                                                                  're_lu_6[0][0]']

 re_lu_8 (ReLU)                 (None, 24, 24, 64)   0           ['add_3[0][0]']
```

```
 conv2d_9 (Conv2D)              (None, 24, 24, 64)   36928       ['re_lu_8[0][0]']

 batch_normalization_9 (BatchNo  (None, 24, 24, 64)  256         ['conv2d_9[0][0]']
 rmalization)

 re_lu_9 (ReLU)                 (None, 24, 24, 64)   0           ['batch_normalization_9[0][0]']

 conv2d_10 (Conv2D)             (None, 24, 24, 64)   36928       ['re_lu_9[0][0]']

 batch_normalization_10 (BatchN  (None, 24, 24, 64)  256         ['conv2d_10[0][0]']
 ormalization)

 add_4 (Add)                    (None, 24, 24, 64)   0           ['batch_normalization_10[0][0]',
                                                                  're_lu_8[0][0]']

 re_lu_10 (ReLU)                (None, 24, 24, 64)   0           ['add_4[0][0]']

 conv2d_11 (Conv2D)             (None, 24, 24, 64)   36928       ['re_lu_10[0][0]']

 batch_normalization_11 (BatchN  (None, 24, 24, 64)  256         ['conv2d_11[0][0]']
 ormalization)

 re_lu_11 (ReLU)                (None, 24, 24, 64)   0           ['batch_normalization_11[0][0]']

 conv2d_12 (Conv2D)             (None, 24, 24, 64)   36928       ['re_lu_11[0][0]']

 batch_normalization_12 (BatchN  (None, 24, 24, 64)  256         ['conv2d_12[0][0]']
 ormalization)

 add_5 (Add)                    (None, 24, 24, 64)   0           ['batch_normalization_12[0][0]',
                                                                  're_lu_10[0][0]']

 re_lu_12 (ReLU)                (None, 24, 24, 64)   0           ['add_5[0][0]']

 conv2d_13 (Conv2D)             (None, 24, 24, 64)   36928       ['re_lu_12[0][0]']

 batch_normalization_13 (BatchN  (None, 24, 24, 64)  256         ['conv2d_13[0][0]']
 ormalization)

 re_lu_13 (ReLU)                (None, 24, 24, 64)   0           ['batch_normalization_13[0][0]']

 conv2d_14 (Conv2D)             (None, 24, 24, 64)   36928       ['re_lu_13[0][0]']

 batch_normalization_14 (BatchN  (None, 24, 24, 64)  256         ['conv2d_14[0][0]']
 ormalization)

 add_6 (Add)                    (None, 24, 24, 64)   0           ['batch_normalization_14[0][0]',
                                                                  're_lu_12[0][0]']

 re_lu_14 (ReLU)                (None, 24, 24, 64)   0           ['add_6[0][0]']

 conv2d_15 (Conv2D)             (None, 12, 12, 128)  73856       ['re_lu_14[0][0]']

 batch_normalization_15 (BatchN  (None, 12, 12, 128)  512        ['conv2d_15[0][0]']
 ormalization)

 re_lu_15 (ReLU)                (None, 12, 12, 128)  0           ['batch_normalization_15[0][0]']

 conv2d_16 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_15[0][0]']

 conv2d_17 (Conv2D)             (None, 12, 12, 128)  8320        ['re_lu_14[0][0]']

 batch_normalization_16 (BatchN  (None, 12, 12, 128)  512        ['conv2d_16[0][0]']
 ormalization)

 batch_normalization_17 (BatchN  (None, 12, 12, 128)  512        ['conv2d_17[0][0]']
 ormalization)

 add_7 (Add)                    (None, 12, 12, 128)  0           ['batch_normalization_16[0][0]',
                                                                  'batch_normalization_17[0][0]']

 re_lu_16 (ReLU)                (None, 12, 12, 128)  0           ['add_7[0][0]']

 conv2d_18 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_16[0][0]']

 batch_normalization_18 (BatchN  (None, 12, 12, 128)  512        ['conv2d_18[0][0]']
 ormalization)

 re_lu_17 (ReLU)                (None, 12, 12, 128)  0           ['batch_normalization_18[0][0]']

 conv2d_19 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_17[0][0]']

 batch_normalization_19 (BatchN  (None, 12, 12, 128)  512        ['conv2d_19[0][0]']
 ormalization)

 add_8 (Add)                    (None, 12, 12, 128)  0           ['batch_normalization_19[0][0]',
                                                                  're_lu_16[0][0]']

 re_lu_18 (ReLU)                (None, 12, 12, 128)  0           ['add_8[0][0]']
```

```
 conv2d_20 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_18[0][0]']

 batch_normalization_20 (BatchN  (None, 12, 12, 128)  512        ['conv2d_20[0][0]']
 ormalization)

 re_lu_19 (ReLU)                (None, 12, 12, 128)  0           ['batch_normalization_20[0][0]']

 conv2d_21 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_19[0][0]']

 batch_normalization_21 (BatchN  (None, 12, 12, 128)  512        ['conv2d_21[0][0]']
 ormalization)

 add_9 (Add)                    (None, 12, 12, 128)  0           ['batch_normalization_21[0][0]',
                                                                  're_lu_18[0][0]']

 re_lu_20 (ReLU)                (None, 12, 12, 128)  0           ['add_9[0][0]']

 conv2d_22 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_20[0][0]']

 batch_normalization_22 (BatchN  (None, 12, 12, 128)  512        ['conv2d_22[0][0]']
 ormalization)

 re_lu_21 (ReLU)                (None, 12, 12, 128)  0           ['batch_normalization_22[0][0]']

 conv2d_23 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_21[0][0]']

 batch_normalization_23 (BatchN  (None, 12, 12, 128)  512        ['conv2d_23[0][0]']
 ormalization)

 add_10 (Add)                   (None, 12, 12, 128)  0           ['batch_normalization_23[0][0]',
                                                                  're_lu_20[0][0]']

 re_lu_22 (ReLU)                (None, 12, 12, 128)  0           ['add_10[0][0]']

 conv2d_24 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_22[0][0]']

 batch_normalization_24 (BatchN  (None, 12, 12, 128)  512        ['conv2d_24[0][0]']
 ormalization)

 re_lu_23 (ReLU)                (None, 12, 12, 128)  0           ['batch_normalization_24[0][0]']

 conv2d_25 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_23[0][0]']

 batch_normalization_25 (BatchN  (None, 12, 12, 128)  512        ['conv2d_25[0][0]']
 ormalization)

 add_11 (Add)                   (None, 12, 12, 128)  0           ['batch_normalization_25[0][0]',
                                                                  're_lu_22[0][0]']

 re_lu_24 (ReLU)                (None, 12, 12, 128)  0           ['add_11[0][0]']

 conv2d_26 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_24[0][0]']

 batch_normalization_26 (BatchN  (None, 12, 12, 128)  512        ['conv2d_26[0][0]']
 ormalization)

 re_lu_25 (ReLU)                (None, 12, 12, 128)  0           ['batch_normalization_26[0][0]']

 conv2d_27 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_25[0][0]']

 batch_normalization_27 (BatchN  (None, 12, 12, 128)  512        ['conv2d_27[0][0]']
 ormalization)

 add_12 (Add)                   (None, 12, 12, 128)  0           ['batch_normalization_27[0][0]',
                                                                  're_lu_24[0][0]']

 re_lu_26 (ReLU)                (None, 12, 12, 128)  0           ['add_12[0][0]']

 conv2d_28 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_26[0][0]']

 batch_normalization_28 (BatchN  (None, 12, 12, 128)  512        ['conv2d_28[0][0]']
 ormalization)

 re_lu_27 (ReLU)                (None, 12, 12, 128)  0           ['batch_normalization_28[0][0]']

 conv2d_29 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_27[0][0]']

 batch_normalization_29 (BatchN  (None, 12, 12, 128)  512        ['conv2d_29[0][0]']
 ormalization)

 add_13 (Add)                   (None, 12, 12, 128)  0           ['batch_normalization_29[0][0]',
                                                                  're_lu_26[0][0]']

 re_lu_28 (ReLU)                (None, 12, 12, 128)  0           ['add_13[0][0]']

 conv2d_30 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_28[0][0]']

 batch_normalization_30 (BatchN  (None, 12, 12, 128)  512        ['conv2d_30[0][0]']
 ormalization)

 re_lu_29 (ReLU)                (None, 12, 12, 128)  0           ['batch_normalization_30[0][0]']
```

```
 conv2d_31 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_29[0][0]']

 batch_normalization_31 (BatchN  (None, 12, 12, 128)  512        ['conv2d_31[0][0]']
 ormalization)

 add_14 (Add)                   (None, 12, 12, 128)  0           ['batch_normalization_31[0][0]',
                                                                  're_lu_28[0][0]']

 re_lu_30 (ReLU)                (None, 12, 12, 128)  0           ['add_14[0][0]']

 conv2d_32 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_30[0][0]']

 batch_normalization_32 (BatchN  (None, 12, 12, 128)  512        ['conv2d_32[0][0]']
 ormalization)

 re_lu_31 (ReLU)                (None, 12, 12, 128)  0           ['batch_normalization_32[0][0]']

 conv2d_33 (Conv2D)             (None, 12, 12, 128)  147584      ['re_lu_31[0][0]']

 batch_normalization_33 (BatchN  (None, 12, 12, 128)  512        ['conv2d_33[0][0]']
 ormalization)

 add_15 (Add)                   (None, 12, 12, 128)  0           ['batch_normalization_33[0][0]',
                                                                  're_lu_30[0][0]']

 re_lu_32 (ReLU)                (None, 12, 12, 128)  0           ['add_15[0][0]']

 global_average_pooling2d (Glob  (None, 128)         0           ['re_lu_32[0][0]']
 alAveragePooling2D)

 dense (Dense)                  (None, 1)            129         ['global_average_pooling2d[0][0]'
                                                                 ]

==================================================================================================
Total params: 3,131,265
Trainable params: 3,124,481
Non-trainable params: 6,784
__________________________________________________________________________________________________
```

In [ ]:

```
lr =0.01
resnet.compile(optimizers.Adam(lr), loss = "binary_crossentropy", metrics = ["accuracy",ROC])


resnet_history = resnet.fit(train_generator, steps_per_epoch = train_step_size, epochs = EPOCH,
                                validation_data = valid_generator, validation_steps = valid_step_size,
                                callbacks = [reducelr,earlystopper], verbose = 1)
```

```
Epoch 1/20
100/100 [==============================] - 48s 425ms/step - loss: 0.6207 - accuracy: 0.7574 - auc: 0.8408 - val_loss: 197.2018 - val_accuracy: 0.6008 - val_auc: 0.5000 - lr: 0.0100
Epoch 2/20
100/100 [==============================] - 39s 390ms/step - loss: 0.4308 - accuracy: 0.8108 - auc: 0.8747 - val_loss: 4.9183 - val_accuracy: 0.6008 - val_auc: 0.3656 - lr: 0.0100
Epoch 3/20
100/100 [==============================] - 42s 422ms/step - loss: 0.4181 - accuracy: 0.8188 - auc: 0.8828 - val_loss: 3.7287 - val_accuracy: 0.6000 - val_auc: 0.4777 - lr: 0.0100
Epoch 4/20
100/100 [==============================] - 41s 403ms/step - loss: 0.3999 - accuracy: 0.8290 - auc: 0.8929 - val_loss: 1.3311 - val_accuracy: 0.6079 - val_auc: 0.6833 - lr: 0.0100
Epoch 5/20
100/100 [==============================] - 41s 402ms/step - loss: 0.3807 - accuracy: 0.8386 - auc: 0.9042 - val_loss: 0.7352 - val_accuracy: 0.6779 - val_auc: 0.8252 - lr: 0.0100
Epoch 6/20
100/100 [==============================] - 39s 388ms/step - loss: 0.3663 - accuracy: 0.8464 - auc: 0.9108 - val_loss: 0.6701 - val_accuracy: 0.6708 - val_auc: 0.8038 - lr: 0.0100
Epoch 7/20
100/100 [==============================] - 39s 386ms/step - loss: 0.3572 - accuracy: 0.8494 - auc: 0.9154 - val_loss: 0.4423 - val_accuracy: 0.7992 - val_auc: 0.8796 - lr: 0.0100
Epoch 8/20
100/100 [==============================] - 39s 384ms/step - loss: 0.3582 - accuracy: 0.8476 - auc: 0.9146 - val_loss: 0.5434 - val_accuracy: 0.6954 - val_auc: 0.8123 - lr: 0.0100
Epoch 9/20
100/100 [==============================] - 39s 388ms/step - loss: 0.3487 - accuracy: 0.8548 - auc: 0.9192 - val_loss: 0.9197 - val_accuracy: 0.7129 - val_auc: 0.7364 - lr: 0.0100
Epoch 10/20
100/100 [==============================] - ETA: 0s - loss: 0.3478 - accuracy: 0.8574 - auc: 0.9198
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0009999999776482583.
100/100 [==============================] - 39s 390ms/step - loss: 0.3478 - accuracy: 0.8574 - auc: 0.9198 - val_loss: 1.3343 - val_accuracy: 0.6350 - val_auc: 0.7821 - lr: 0.0100
Epoch 11/20
100/100 [==============================] - 41s 406ms/step - loss: 0.3231 - accuracy: 0.8621 - auc: 0.9308 - val_loss: 0.4740 - val_accuracy: 0.7742 - val_auc: 0.9146 - lr: 1.0000e-03
Epoch 12/20
100/100 [==============================] - 41s 407ms/step - loss: 0.3035 - accuracy: 0.8741 - auc: 0.9391 - val_loss: 0.4375 - val_accuracy: 0.8046 - val_auc: 0.8825 - lr: 1.0000e-03
Epoch 13/20
100/100 [==============================] - 44s 433ms/step - loss: 0.3081 - accuracy: 0.8717 - auc: 0.9373 - val_loss: 0.7308 - val_accuracy: 0.7417 - val_auc: 0.7586 - lr: 1.0000e-03
Epoch 14/20
100/100 [==============================] - 48s 472ms/step - loss: 0.3008 - accuracy: 0.8755 - auc: 0.9399 - val_loss: 0.3791 - val_accuracy: 0.8267 - val_auc: 0.9306 - lr: 1.0000e-03
Epoch 15/20
100/100 [==============================] - 58s 573ms/step - loss: 0.3042 - accuracy: 0.8747 - auc: 0.9385 - val_loss: 0.4949 - val_accuracy: 0.8054 - val_auc: 0.8800 - lr: 1.0000e-03
Epoch 16/20
100/100 [==============================] - 50s 495ms/step - loss: 0.2923 - accuracy: 0.8774 - auc: 0.9440 - val_loss: 0.4415 - val_accuracy: 0.8046 - val_auc: 0.9175 - lr: 1.0000e-03
Epoch 17/20
100/100 [==============================] - ETA: 0s - loss: 0.2973 - accuracy: 0.8766 - auc: 0.9412
Epoch 17: ReduceLROnPlateau reducing learning rate to 9.999999310821295e-05.
100/100 [==============================] - 60s 596ms/step - loss: 0.2973 - accuracy: 0.8766 - auc: 0.9412 - val_loss: 0.5866 - val_accuracy: 0.7721 - val_auc: 0.8749 - lr: 1.0000e-03
Epoch 18/20
100/100 [==============================] - 47s 463ms/step - loss: 0.2815 - accuracy: 0.8866 - auc: 0.9474 - val_loss: 0.2932 - val_accuracy: 0.8763 - val_auc: 0.9424 - lr: 1.0000e-04
Epoch 19/20
100/100 [==============================] - 46s 459ms/step - loss: 0.2868 - accuracy: 0.8814 - auc: 0.9455 - val_loss: 0.2961 - val_accuracy: 0.8725 - val_auc: 0.9476 - lr: 1.0000e-04
Epoch 20/20
100/100 [==============================] - 47s 469ms/step - loss: 0.2852 - accuracy: 0.8824 - auc: 0.9462 - val_loss: 0.3904 - val_accuracy: 0.8254 - val_auc: 0.9403 - lr: 1.0000e-04
```

In [ ]:

```python
# plot model accuracy per epoch
plt.plot(resnet_history.history['accuracy'])
plt.plot(resnet_history.history['val_accuracy'])
plt.title('ResNET Accuracy per Epoch')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc=0)
plt.show()

# plot model loss per epoch
plt.plot(resnet_history.history['loss'])
plt.plot(resnet_history.history['val_loss'])
plt.title('ResNET Loss per Epoch')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc=0)
plt.show()

# plot model ROC per epoch

plt.plot(resnet_history.history['auc'])
plt.plot(resnet_history.history['val_auc'])
plt.title('ResNET AUC ROC per Epoch')
plt.ylabel('ROC')
plt.xlabel('epoch')
```

```
plt.legend(['train', 'validate'], loc=0)
plt.show()
```

Out[ ]: [<matplotlib.lines.Line2D at 0x16cd2015810>]

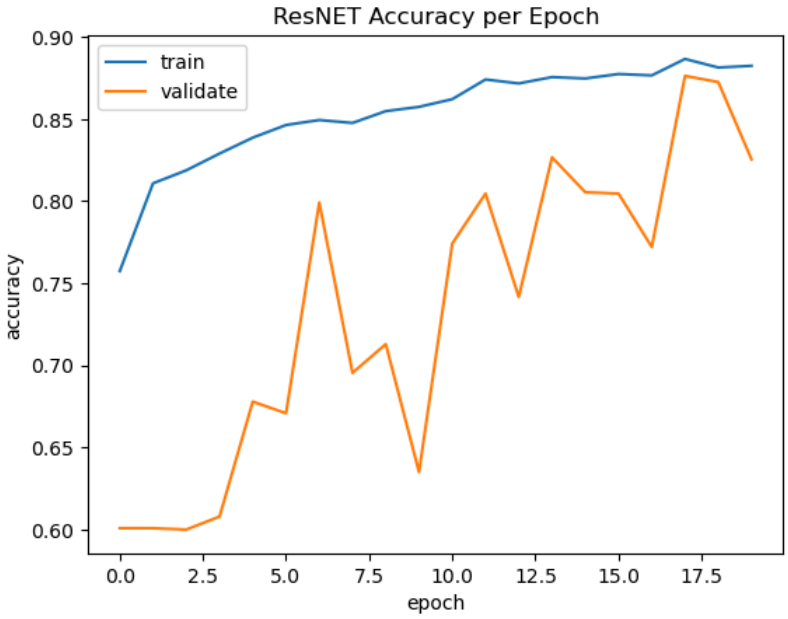
Out[ ]: [<matplotlib.lines.Line2D at 0x16cd2015b70>]

Out[ ]: Text(0.5, 1.0, 'ResNET Accuracy per Epoch')

Out[ ]: Text(0, 0.5, 'accuracy')

Out[ ]: Text(0.5, 0, 'epoch')

Out[ ]: <matplotlib.legend.Legend at 0x16cca40eb00>



Out[ ]: [<matplotlib.lines.Line2D at 0x16cd207b7f0>]

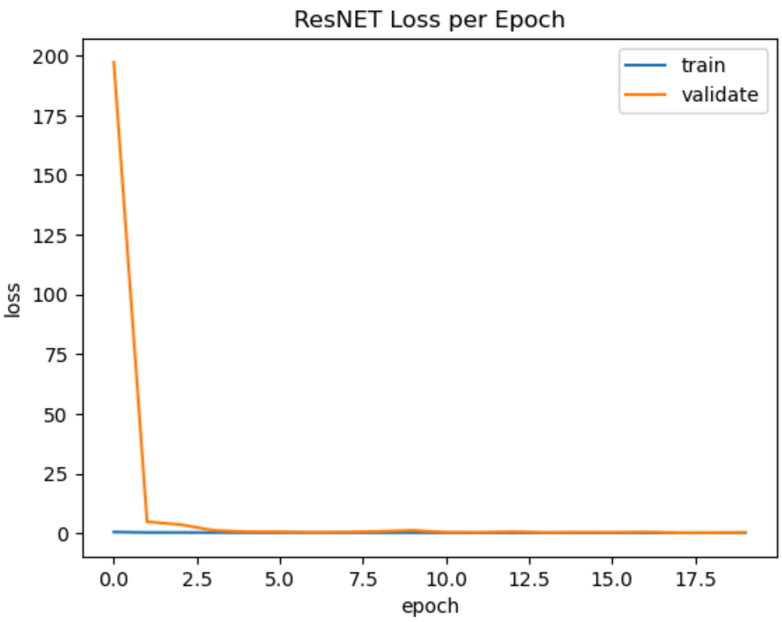
Out[ ]: [<matplotlib.lines.Line2D at 0x16cd207a800>]

Out[ ]: Text(0.5, 1.0, 'ResNET Loss per Epoch')

Out[ ]: Text(0, 0.5, 'loss')

Out[ ]: Text(0.5, 0, 'epoch')

Out[ ]: <matplotlib.legend.Legend at 0x16ccbde7a30>



Out[ ]: [<matplotlib.lines.Line2D at 0x16cd322df00>]

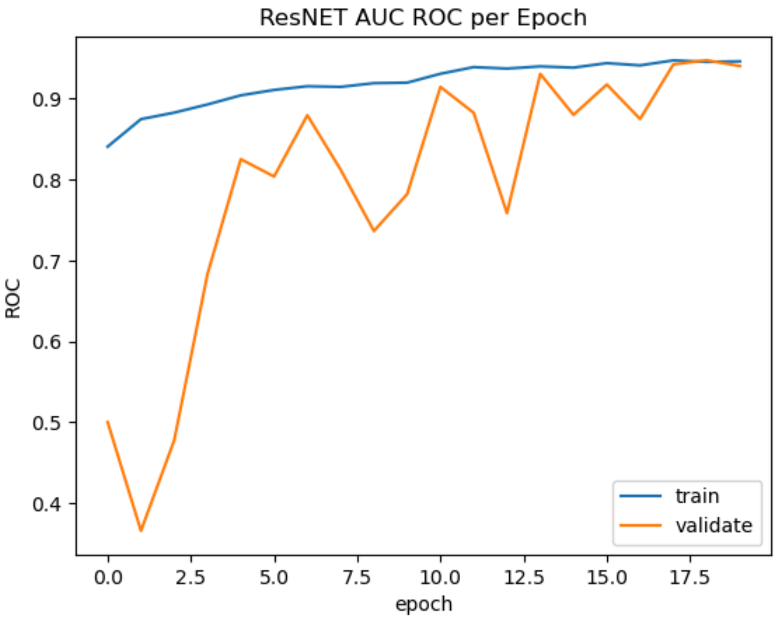
Out[ ]: [<matplotlib.lines.Line2D at 0x16cd322e170>]

Out[ ]: Text(0.5, 1.0, 'ResNET AUC ROC per Epoch')

Out[ ]: Text(0, 0.5, 'ROC')

Out[ ]: Text(0.5, 0, 'epoch')

Out[ ]: <matplotlib.legend.Legend at 0x16cd31fbbb0>



TRANSFER LEARNING VGG16

The model is initialized using weights pre-trained on the ImageNet dataset.Only the convolutional base part is used, which contains the weights for feature extraction

In [ ]:
```
# VGG16 trained on the imagenet dataset, with the top fully connected network removed
vgg_conv_base = VGG16(weights = "imagenet", include_top = False, input_shape =INPUT_SHAPE)
vgg = Sequential()
#conv
vgg.add(vgg_conv_base)
vgg.add(Flatten())
#fc
vgg.add(Dense(4096, use_bias = False))
vgg.add(BatchNormalization())
vgg.add(Activation("relu"))
vgg.add(Dropout(0.5))
#fc
vgg.add(Dense(4096, use_bias = False))
vgg.add(BatchNormalization())
vgg.add(Activation("relu"))
vgg.add(Dropout(0.5))
#fc
vgg.add(Dense(1, activation = "sigmoid"))
```

Feature reuse:

The first few layers of VGG (including the ones before block5_conv1) usually learn more general image features such as edges, textures, etc. These features are useful for many vision tasks, so keeping the pre-trained weights of these layers can be a good starting point for subsequent tasks.

In [ ]:
```
vgg_conv_base.Trainable = True
set_trainable = False
#The first few layers of VGG (including the layers before block5_conv1) usually learn general image features such as edges, textures, etc.
for layer in vgg_conv_base.layers:
    if layer.name == "block5_conv1":
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

vgg_conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 96, 96, 3)]       0         
                                                                 
 block1_conv1 (Conv2D)       (None, 96, 96, 64)        1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 96, 96, 64)        36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 48, 48, 64)        0         
                                                                 
 block2_conv1 (Conv2D)       (None, 48, 48, 128)       73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 48, 48, 128)       147584    
                                                                 
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 96, 96, 3)]       0         
                                                                 
 block1_conv1 (Conv2D)       (None, 96, 96, 64)        1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 96, 96, 64)        36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 48, 48, 64)        0         
                                                                 
 block2_conv1 (Conv2D)       (None, 48, 48, 128)       73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 48, 48, 128)       147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 24, 24, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 24, 24, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 24, 24, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 24, 24, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 12, 12, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 12, 12, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 12, 12, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 12, 12, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 6, 6, 512)         0         
                                                                 
 block5_conv1 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 3, 3, 512)         0         
                                                                 
=================================================================
Total params: 14,714,688
Trainable params: 7,079,424
Non-trainable params: 7,635,264
_________________________________________________________________
```

In [ ]:

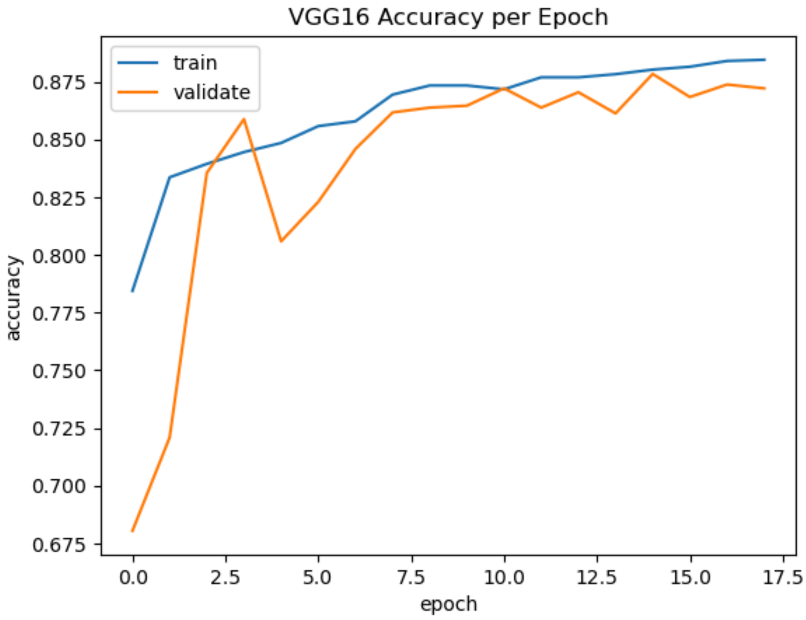
```
%%time
K.clear_session()
lr =0.001
vgg.compile(optimizers.Adam(lr), loss = "binary_crossentropy", metrics = ["accuracy",ROC])



vgg_history = vgg.fit(train_generator, steps_per_epoch = train_step_size, epochs = EPOCH,
                                validation_data = valid_generator, validation_steps = valid_step_size,
                                callbacks = [reducelr,earlystopper], verbose = 1)
```

```
Epoch 1/20
100/100 [==============================] - 48s 451ms/step - loss: 0.6016 - accuracy: 0.7844 - auc: 0.8594 - val_loss: 1.1090 - val_accuracy:
0.6804 - val_auc: 0.7513 - lr: 0.0010
Epoch 2/20
100/100 [==============================] - 45s 451ms/step - loss: 0.3933 - accuracy: 0.8335 - auc: 0.9002 - val_loss: 0.8003 - val_accuracy:
0.7208 - val_auc: 0.8047 - lr: 0.0010
Epoch 3/20
100/100 [==============================] - 46s 455ms/step - loss: 0.3786 - accuracy: 0.8394 - auc: 0.9077 - val_loss: 0.4003 - val_accuracy:
0.8354 - val_auc: 0.9169 - lr: 0.0010
Epoch 4/20
100/100 [==============================] - 50s 495ms/step - loss: 0.3747 - accuracy: 0.8445 - auc: 0.9102 - val_loss: 0.3416 - val_accuracy:
0.8587 - val_auc: 0.9292 - lr: 0.0010
Epoch 5/20
100/100 [==============================] - 53s 533ms/step - loss: 0.3541 - accuracy: 0.8484 - auc: 0.9186 - val_loss: 0.4644 - val_accuracy:
0.8058 - val_auc: 0.9324 - lr: 0.0010
Epoch 6/20
100/100 [==============================] - 45s 444ms/step - loss: 0.3448 - accuracy: 0.8557 - auc: 0.9219 - val_loss: 0.4210 - val_accuracy:
0.8229 - val_auc: 0.9281 - lr: 0.0010
Epoch 7/20
100/100 [==============================] - ETA: 0s - loss: 0.3403 - accuracy: 0.8578 - auc: 0.9239
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
100/100 [==============================] - 44s 439ms/step - loss: 0.3403 - accuracy: 0.8578 - auc: 0.9239 - val_loss: 0.3697 - val_accuracy:
0.8458 - val_auc: 0.9333 - lr: 0.0010
Epoch 8/20
100/100 [==============================] - 45s 448ms/step - loss: 0.3087 - accuracy: 0.8694 - auc: 0.9367 - val_loss: 0.3308 - val_accuracy:
0.8617 - val_auc: 0.9378 - lr: 1.0000e-04
Epoch 9/20
100/100 [==============================] - 45s 447ms/step - loss: 0.3054 - accuracy: 0.8733 - auc: 0.9380 - val_loss: 0.3162 - val_accuracy:
0.8637 - val_auc: 0.9357 - lr: 1.0000e-04
Epoch 10/20
100/100 [==============================] - 44s 442ms/step - loss: 0.3038 - accuracy: 0.8733 - auc: 0.9390 - val_loss: 0.3295 - val_accuracy:
0.8646 - val_auc: 0.9403 - lr: 1.0000e-04
Epoch 11/20
100/100 [==============================] - 45s 445ms/step - loss: 0.3004 - accuracy: 0.8717 - auc: 0.9396 - val_loss: 0.2982 - val_accuracy:
0.8721 - val_auc: 0.9419 - lr: 1.0000e-04
Epoch 12/20
100/100 [==============================] - 45s 443ms/step - loss: 0.3006 - accuracy: 0.8769 - auc: 0.9398 - val_loss: 0.3222 - val_accuracy:
0.8637 - val_auc: 0.9408 - lr: 1.0000e-04
Epoch 13/20
100/100 [==============================] - 44s 440ms/step - loss: 0.2930 - accuracy: 0.8769 - auc: 0.9425 - val_loss: 0.2978 - val_accuracy:
0.8704 - val_auc: 0.9439 - lr: 1.0000e-04
Epoch 14/20
100/100 [==============================] - 45s 445ms/step - loss: 0.2886 - accuracy: 0.8782 - auc: 0.9448 - val_loss: 0.3252 - val_accuracy:
0.8612 - val_auc: 0.9409 - lr: 1.0000e-04
Epoch 15/20
100/100 [==============================] - 45s 444ms/step - loss: 0.2905 - accuracy: 0.8802 - auc: 0.9443 - val_loss: 0.3018 - val_accuracy:
0.8783 - val_auc: 0.9434 - lr: 1.0000e-04
Epoch 16/20
100/100 [==============================] - ETA: 0s - loss: 0.2904 - accuracy: 0.8815 - auc: 0.9437
Epoch 16: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
100/100 [==============================] - 45s 448ms/step - loss: 0.2904 - accuracy: 0.8815 - auc: 0.9437 - val_loss: 0.3148 - val_accuracy:
0.8683 - val_auc: 0.9387 - lr: 1.0000e-04
Epoch 17/20
100/100 [==============================] - 45s 444ms/step - loss: 0.2859 - accuracy: 0.8840 - auc: 0.9458 - val_loss: 0.3060 - val_accuracy:
0.8737 - val_auc: 0.9438 - lr: 1.0000e-05
Epoch 18/20
100/100 [==============================] - ETA: 0s - loss: 0.2796 - accuracy: 0.8845 - auc: 0.9475Restoring model weights from the end of the
best epoch: 13.
100/100 [==============================] - 45s 445ms/step - loss: 0.2796 - accuracy: 0.8845 - auc: 0.9475 - val_loss: 0.3022 - val_accuracy:
0.8721 - val_auc: 0.9436 - lr: 1.0000e-05
Epoch 18: early stopping
CPU times: total: 19min 18s
Wall time: 14min 19s
```

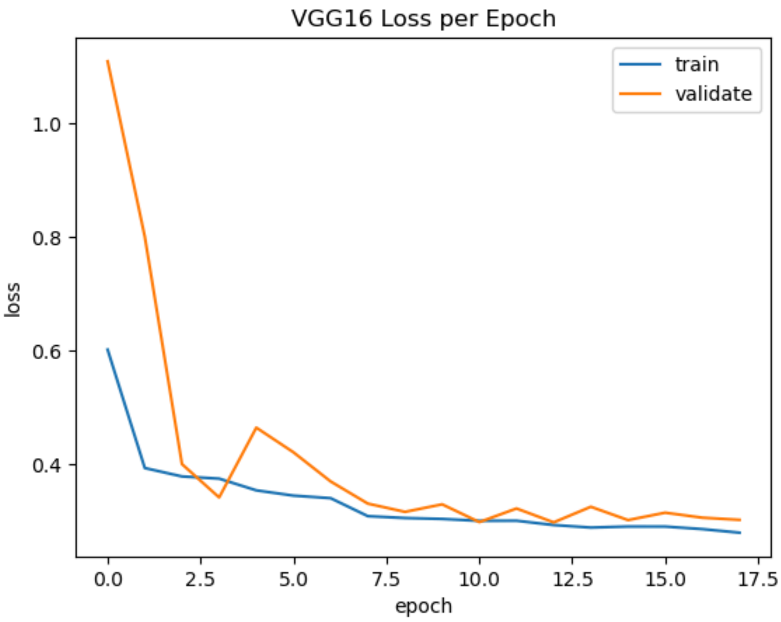
In [ ]:

```
# plot model accuracy per epoch
plt.plot(vgg_history.history['accuracy'])
plt.plot(vgg_history.history['val_accuracy'])
plt.title('VGG16 Accuracy per Epoch')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc=0)
plt.show()

# plot model loss per epoch
plt.plot(vgg_history.history['loss'])
plt.plot(vgg_history.history['val_loss'])
plt.title('VGG16 Loss per Epoch')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc=0)
plt.show()

# plot model ROC per epoch
plt.plot(vgg_history.history['auc'])
plt.plot(vgg_history.history['val_auc'])
plt.title('VGG16 AUC ROC per Epoch')
plt.ylabel('ROC')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc=0)
plt.show()
```

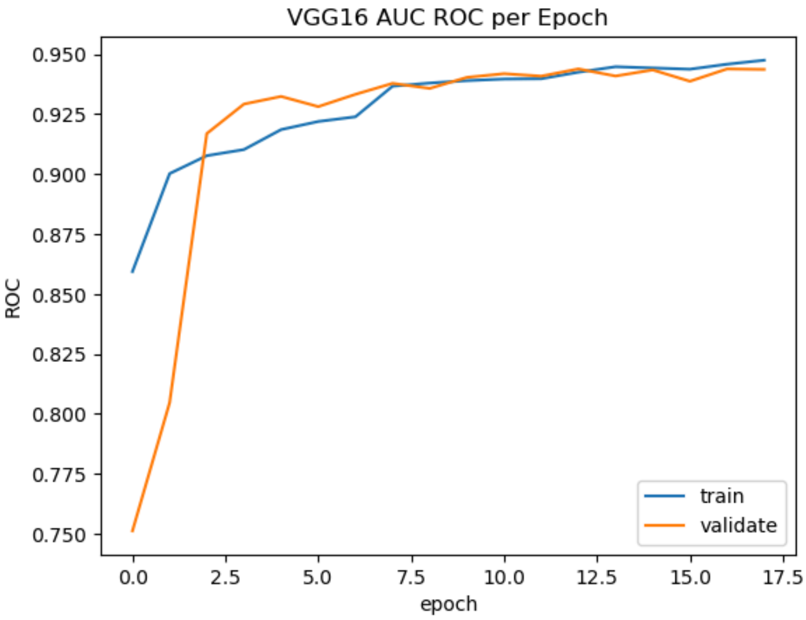
```
Out[ ]: [<matplotlib.lines.Line2D at 0x16cc97f6740>]
Out[ ]: [<matplotlib.lines.Line2D at 0x16cc97f44c0>]
Out[ ]: Text(0.5, 1.0, 'VGG16 Accuracy per Epoch')
Out[ ]: Text(0, 0.5, 'accuracy')
Out[ ]: Text(0.5, 0, 'epoch')
Out[ ]: <matplotlib.legend.Legend at 0x16cca0759c0>
```




```
Out[ ]: [<matplotlib.lines.Line2D at 0x16cb0490340>]
Out[ ]: [<matplotlib.lines.Line2D at 0x16cb0492770>]
Out[ ]: Text(0.5, 1.0, 'VGG16 Loss per Epoch')
Out[ ]: Text(0, 0.5, 'loss')
Out[ ]: Text(0.5, 0, 'epoch')
Out[ ]: <matplotlib.legend.Legend at 0x16cc9190e20>
```




```
Out[ ]: [<matplotlib.lines.Line2D at 0x16cc98d9db0>]
Out[ ]: [<matplotlib.lines.Line2D at 0x16cc98dafe0>]
Out[ ]: Text(0.5, 1.0, 'VGG16 AUC ROC per Epoch')
Out[ ]: Text(0, 0.5, 'ROC')
Out[ ]: Text(0.5, 0, 'epoch')
Out[ ]: <matplotlib.legend.Legend at 0x16cc9ff7160>
```



Step4

Results and Analysis (35 pts)

Run hyperparameter tuning, try different architectures for comparison, apply techniques to improve training or performance, and discuss what helped.

Includes results with tables and figures. There is an analysis of why or why not something worked well, troubleshooting, and a hyperparameter optimization procedure summary.

In [ ]:
```
test_imgs_path_df = pd.DataFrame(glob("test/*.tif"), columns = ["path"])
test_imgs_path_df["id"] = test_imgs_path_df["path"].map(lambda x: x.split("\\")[-1].split(".")[0])
test_imgs_path_df.shape
test_imgs_path_df.head()
```

Out[ ]: (57458, 2)

Out[ ]:

	path	id
0	test\00006537328c33e284c973d7b39d340809f7271b.tif	00006537328c33e284c973d7b39d340809f7271b
1	test\0000ec92553fda4ce39889f9226ace43cae3364e.tif	0000ec92553fda4ce39889f9226ace43cae3364e
2	test\00024a6dee61f12f7856b0fc6be20bc7a48ba3d2.tif	00024a6dee61f12f7856b0fc6be20bc7a48ba3d2
3	test\000253dfaa0be9d0d100283b22284ab2f6b643f6.tif	000253dfaa0be9d0d100283b22284ab2f6b643f6
4	test\000270442cc15af719583a8172c87cd2bd9c7746.tif	000270442cc15af719583a8172c87cd2bd9c7746

In [ ]:
```
# save results data to csv
def save_data(y_prob,filename):
    '''
    predicted data is output as csv
    '''

    labels =(y_prob > 0.5) * 1
    print(type(y_prob),labels)
    submission=pd.DataFrame ( {
    "id" : test_imgs_path_df [ "id" ],

    "label" :labels
    })
    submission.to_csv ( filename+'_result.csv',index=False)
```

In [ ]:
```
datagen_test = ImageDataGenerator(rescale=1./255.)

test_generator = datagen_test.flow_from_dataframe(
    dataframe = test_imgs_path_df,
    directory = None,
    x_col = "path",
    y_col = 'id',
    target_size = TARGET_SIZE,
    class_mode = None,
    batch_size = BATCH_SIZE,
    seed = RANDOM_STATE,
    shuffle = False
    )
```

Found 57458 validated image filenames.

In [ ]:
```
my_vgg_prob = my_VGG16.predict(test_generator, verbose=1)
my_vgg_prob=np.transpose(my_vgg_prob)[0]
```

```
save_data(my_vgg_prob,'my_vgg16')
```

```
599/599 [==============================] - 153s 255ms/step
<class 'numpy.ndarray'> [0 0 0 ... 0 1 0]
```

my_vgg16_result.csv Complete (after deadline) · 1d ago	0.7226	0.7717

In [ ]:
```
resnet_prob = resnet.predict(test_generator, verbose=1)
resnet_prob=np.transpose(resnet_prob)[0]
save_data(resnet_prob,'resnet')
```

```
599/599 [==============================] - 66s 109ms/step
<class 'numpy.ndarray'> [1 1 0 ... 0 1 0]
```

resnet_result.csv Complete (after deadline) · now	0.7827	0.8122

In [ ]:
```
vgg_prob = vgg.predict(test_generator, verbose=1)

vgg_prob=np.transpose(vgg_prob)[0]

vgg_prob.shape

save_data(vgg_prob,'vgg16')
```

Improve performance:

- Reducing the size of the training dataset,due to hardware constraints, using a smaller dataset might be necessary to train a model at all
- Adjust the size of batchsize and epoch

But, Increasing the size of batchsize and the number of epochs can improve the accuracy of the model

vgg16_result.csv Complete (after deadline) · 15h ago	0.8158	0.8292	☐
vgg16_result.csv Complete (after deadline) · 16h ago	0.8210	0.8284	☐
vgg16_result.csv Complete (after deadline) · 18h ago	0.8047	0.8190	☐
vgg16_result.csv Complete (after deadline) · 18h ago	0.8028	0.8161	☐
vgg16_result.csv Complete (after deadline) · 19h ago	0.7989	0.8266	☐
vgg16_result.csv Complete (after deadline) · 19h ago	0.7935	0.8212	☐

batchsize ->96 -> 128 ->196

A smaller batch size means the model updates more frequently, which can lead to faster convergence but might result in less stable convergence.

epoch ->10 ->20

If the model is not given enough epochs, it might not converge to an optimal solution, resulting in underfitting, where the model is too simple to capture the underlying patterns in the data.

Step5

Conclusion (15 pts)

Discuss and interpret results as well as learnings and takeaways. What did and did not help improve the performance of your models? What improvements could you try in the future?

Although the ResNet model is deeper, its parameter utilization is higher due to the existence of residual connections, and the model can still maintain a relatively small number of parameters and higher computational efficiency in the case of deeper. Training a model from scratch requires more computation, but the parameters can be adjusted. The benefit of transfer learning is that the pre-trained weights can be reused, and only a small amount of training is needed to adapt to the new task, thus greatly reducing the training time. And with transfer learning, the knowledge of the pre-trained model can make up for the problem of insufficient data, so that it can also achieve good performance on small datasets.