

1 Original dataset modeling

Student name: Sam Oliver

Student pace: self-paced

Scheduled project review: 10/12/2022 1:30 PM EST

Instructor name: Abhineet Kulkarni

Preface for this notebook: If not already noted, please reference the README and preferably the STRIP.ipynb notebook (Google Colab notebook). There is information on this project, evaluation methods, notes on the datasets used, etc.

1.1 Dataset description

This notebook will focus on modeling with the [original dataset used in this Kaggle competition \(https://www.kaggle.com/competitions/mayo-clinic-strip-ai/data\)](https://www.kaggle.com/competitions/mayo-clinic-strip-ai/data). This dataset contains 1,158 different files with a size of about 400 GB. Here is a more detailed description found of each file and data field:

1. train - A folder containing images in the TIFF format to be used as training data.
2. test - A folder containing images to be used as test data. The actual test data comprises about 280 images.
3. train.csv Contains annotations for images in the train/ folder.
 - image_id - A unique identifier for this instance having the form {patient_id}_{image_num}. Corresponds to the image {image_id}.tif.
 - center_id - Identifies the medical center where the slide was obtained.
 - patient_id - Identifies the patient from whom the slide was obtained.
 - image_num - Enumerates images of clots obtained from the same patient.
 - label - The etiology of the clot, either CE or LAA. This field is the classification target.
4. test.csv - Annotations for images in the test/ folder. Has the same fields as train.csv excluding label.
5. other.csv - Annotations for images in the other/ folder. Has the same fields as train.csv. The center_id is unavailable for these images however.
 - label - The etiology of the clot, either Unknown or Other.
 - other_specified - The specific etiology, when known, in case the etiology is labeled as Other.
6. sample_submission.csv - A sample submission file in the correct format.

1.2 Import packages and data

```
In [1]: 1 # Imports
2 import pandas as pd
3 import numpy as np
4 from pathlib import Path
5 import glob
6 import os
7 from os import listdir
8 from pathlib import Path
9
10 from skimage.io import imread
11 from PIL import Image
12 import matplotlib.pyplot as plt
13 import seaborn as sns
14 import sklearn
15 import cv2
16
17 %matplotlib inline
18
19 import tensorflow as tf
20 from tensorflow import keras
21 from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_
22 from tensorflow.keras.preprocessing.image import img_to_array
23 from tensorflow.keras import models, layers, optimizers, regularizers
24 from tensorflow.keras import activations
25 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
26 from tensorflow.keras.callbacks import EarlyStopping
27 from tensorflow.keras.applications import VGG16, VGG19
28 from tensorflow.keras.models import Model
29 from tensorflow.keras.layers import Input, Dropout
30 from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
31
32 from sklearn.metrics import confusion_matrix, classification_report
33 from sklearn.metrics import plot_confusion_matrix
34 from sklearn.model_selection import train_test_split
35
36 from keras.models import load_model
37 import keras.applications
38
39 import datetime
40 from tqdm import tqdm
41
42 from tensorflow.random import set_seed
43 set_seed(13)
```

```
In [2]: 1 train_df = pd.read_csv(r'C:\Users\18016\Downloads\train\.csv')
        2 train_df.head()
```

```
Out[2]:
```

	image_id	center_id	patient_id	image_num	label
0	006388_0	11	006388	0	CE
1	008e5c_0	11	008e5c	0	CE
2	00c058_0	11	00c058	0	LAA
3	01adc5_0	11	01adc5	0	LAA
4	026c97_0	4	026c97	0	CE

Each patient has an id, each image has an id, the center id (0-11) is unique identifier for each center that contributed images to the registry. image_num references how many images are represented for the unique patient id. Label references whether the stroke occurred as cardioembolic or large artery atherosclerosis.

Append the paths for the images below.

```
In [3]: 1 train_path = r"C:\\Users\\18016\\Downloads\\train\\"
        2 train_df["file_path"] = train_df["image_id"].apply(lambda x: train_path + x)
        3 train_df.head()
```

```
Out[3]:
```

	image_id	center_id	patient_id	image_num	label	file_path
0	006388_0	11	006388	0	CE	C:\\Users\\18016\\Downloads\\train\\006388_0.tif
1	008e5c_0	11	008e5c	0	CE	C:\\Users\\18016\\Downloads\\train\\008e5c_0.tif
2	00c058_0	11	00c058	0	LAA	C:\\Users\\18016\\Downloads\\train\\00c058_0.tif
3	01adc5_0	11	01adc5	0	LAA	C:\\Users\\18016\\Downloads\\train\\01adc5_0.tif
4	026c97_0	4	026c97	0	CE	C:\\Users\\18016\\Downloads\\train\\026c97_0.tif

OpenSlide package will allow for opening and processing of the Whole Slide images (the .tif images represented by the paths in the train_df file_path column).

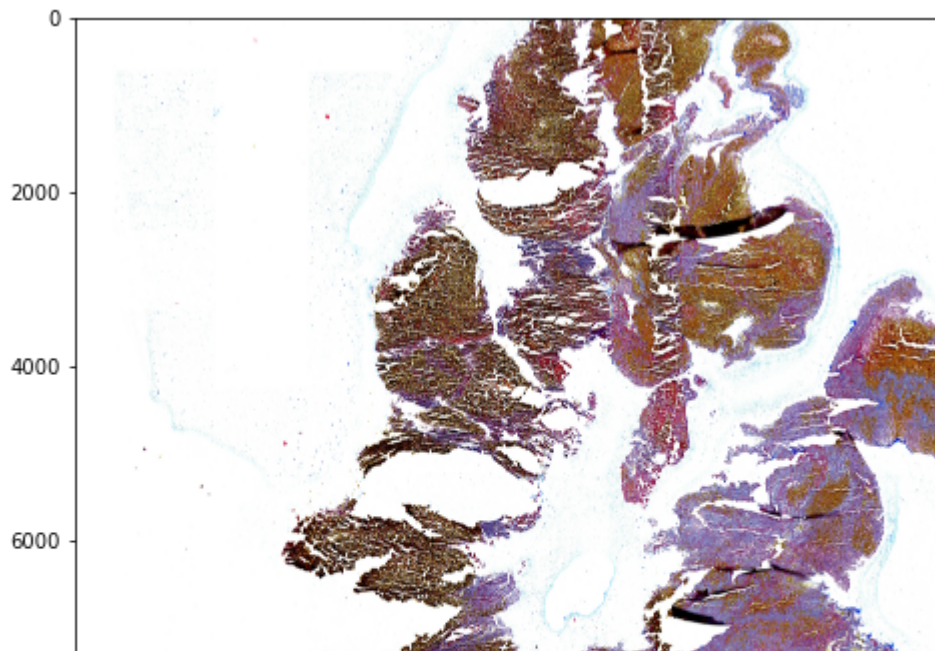
Add openslide path for bin file. Need to add the dll directory file in so OpenSlide builds necessary references.

```
In [4]: 1 OPENSIDE_PATH = r'C:\Users\18016\Desktop\openslide-win64-20220811\bin'
        2
        3 if hasattr(os, 'add_dll_directory'):
        4     # Python >= 3.8 on Windows
        5     with os.add_dll_directory(OPENSIDE_PATH):
        6         import openslide
        7 else:
        8     import openslide
```

```
In [5]: 1 from openslide import OpenSlide
```

1.3 Preview some images

```
In [6]: 1 sample_train = train_df[:5]
        2
        3 for i in range(5):
        4     slide = OpenSlide(sample_train.loc[i, "file_path"])
        5     region = (0, 0)
        6     size = (10000, 10000)
        7     region = slide.read_region(region, 0, size)
        8     plt.figure(figsize=(8, 8))
        9     plt.imshow(region)
       10     plt.show()
```



I wonder if those dark black bands are any indication of a cardioembolic stroke? Also, what are these dark black bands?

The bottom right corner of the last image (LAA type) is interesting as well- does that indicate some issue with how the clot was scanned?

I'm interested in if the center in which the image was taken could be a significant attribute in modeling...

1.4 Clean the dataset

In the STRIP.ipynb notebook, it is noted that there are several images that are blurry and could corrupt future predictions. I will remove these images from this dataset.

```

In [6]: 1 # remove following ids from train_df:
        2 # b894f4_0, 6baf51_0, 7b9aaa_0, 5adc4c_0, bb06a5_0, and e26a04_0
        3 train_df = train_df[(train_df.image_id != 'b894f4_0') &
        4                         (train_df.image_id != '6baf51_0') &
        5                         (train_df.image_id != '7b9aaa_0') &
        6                         (train_df.image_id != '5adc4c_0') &
        7                         (train_df.image_id != 'bb06a5_0') &
        8                         (train_df.image_id != 'e26a04_0')]
        9
       10 # also, reset the index of train_df
       11 train_df = train_df.reset_index(drop=True)
       12
       13 # check changes
       14 train_df

```

```

Out[6]:

```

	image_id	center_id	patient_id	image_num	label	file_path
0	006388_0	11	006388	0	CE	C:\\Users\\18016\\Downloads\\train\\006388_0.tif
1	008e5c_0	11	008e5c	0	CE	C:\\Users\\18016\\Downloads\\train\\008e5c_0.tif
2	00c058_0	11	00c058	0	LAA	C:\\Users\\18016\\Downloads\\train\\00c058_0.tif
3	01adc5_0	11	01adc5	0	LAA	C:\\Users\\18016\\Downloads\\train\\01adc5_0.tif
4	026c97_0	4	026c97	0	CE	C:\\Users\\18016\\Downloads\\train\\026c97_0.tif
...
743	fe9645_0	3	fe9645	0	CE	C:\\Users\\18016\\Downloads\\train\\fe9645_0.tif
744	fe9bec_0	4	fe9bec	0	LAA	C:\\Users\\18016\\Downloads\\train\\fe9bec_0.tif
745	ff14e0_0	6	ff14e0	0	CE	C:\\Users\\18016\\Downloads\\train\\ff14e0_0.tif
746	ffec5c_0	7	ffec5c	0	LAA	C:\\Users\\18016\\Downloads\\train\\ffec5c_0.tif
747	ffec5c_1	7	ffec5c	1	LAA	C:\\Users\\18016\\Downloads\\train\\ffec5c_1.tif

748 rows × 6 columns

Looks good. 748 items is the amount I should have.

1.5 Image transformations

Preprocessing the images because they are very large. Resizing to 512x512 pixels to try to avoid running out of memory.

Also converting images to 3 channels and converting to image array to avoiding OOM. Opening the image with OpenSlide package, which specializes in opening digital pathology Whole Slide images and .tif images.

```
In [7]: 1 def preprocess(image_path):
2         slide = OpenSlide(image_path)      # create OpenSlide image object
3         region = (0, 0)                   # start at bottom left corner
4         size = (10000, 10000)             # size is square 10kx10k pixels starting from
5         image = slide.read_region(region, 0, size)    # read image with params
6         image = image.resize(size=(512,512)).convert('RGB')    # resizing to 512x512
7         image_arr = np.array(image)        # convert image to array
8         return image_arr
```

Reading in each image from the bottom left and going up and to the right by 10k pixels. This technique certainly isn't great for every image, but I think it's probably a decent strategy that still avoids OOM issues.

```
In [ ]: 1 # apply function to images
2
3 # create copy of train_df to apply changes to
4 tdf = train_df.copy()
5 image_arrays = []
6
7 for i in tqdm(tdf['file_path']):
8     img_a = preprocess(i)
9     image_arrays.append(img_a)
10
11 tdf['img_arr'] = image_arrays
12
13 tdf.head()
```

```
3%|| | 24/748 [01:15<41:25, 3.43s/it]
```

Converted images to arrays. Easier on memory this way...

Use an ImageDataGenerator to normalize and permute the images.

```
In [ ]: 1 # use a simple kind of ImageDataGenerator
2 permutes = ImageDataGenerator(
3     rescale = 1. / 255,    # normalize the data - all data points btwn 0-1
4     shear_range = 0.2,    # this distorts the image along an axis
5     zoom_range = 0.1,     # range for random zoom
6     horizontal_flip = True # random horizontal flip
7 )
```

▼ 1.6 Train-test split

Split data into train and test dataframes to use for modeling. Using standard split size of 80-20 train to test % composition.

```
In [ ]: 1 # also make a target column because I will need int labels for modeling
        2
        3 # 0 - CE & 1 - LAA
        4 tdf["target"] = tdf["label"].apply(lambda x : 0 if x=="CE" else 1)
```

```
In [ ]: 1 df_tt = tdf.copy()
        2 train, test = train_test_split(df_tt, test_size=0.2, random_state=42,
        3                               shuffle=True)
        4
        5 train
```

Ensure split worked...

```
In [ ]: 1 plt.imshow(train.img_arr[0])
        2 plt.show()
```

▼ 2 Modeling

▼ 2.1 Helper function for evaluation

Print out loss, RMSE, MAE, plots for these metrics, confusion matrix, and classification report.

```

In [21]: 1 def evaluate_model(history, model, test):
2         # print loss and accuracy of the model on the test set
3         test_loss, test_rmse, test_mae = history.model.evaluate(
4             x=np.array(test['img_arr']).to_list()),
5             y=test['target']
6         )
7         print(f'Test Loss: {test_loss}')
8         print(f'Test RMSE: {test_rmse}')
9         print(f'Test MAE: {test_mae}')
10
11         # create plots for accuracy and loss
12         fig, ax = plt.subplots(1, 2, figsize = (12,8))
13         ax[0].plot(history.history['rmse'])
14         ax[0].plot(history.history['val_rmse'])
15         ax[0].set_title('Model RMSE')
16         ax[0].set_xlabel('Epoch')
17         ax[0].set_ylabel('RMSE')
18         ax[0].legend(['Train', 'Test'], loc='upper left')
19
20         ax[1].plot(history.history['loss'])
21         ax[1].plot(history.history['val_loss'])
22         ax[1].set_title('Model Loss')
23         ax[1].set_xlabel('Epochs')
24         ax[1].set_ylabel('Loss')
25         ax[1].legend(['Train', 'Test'], loc='upper left')
26
27         plt.show()
28
29         # plot confusion matrix - create predictions for the model
30         y_hat_tmp = history.model.predict(
31             x=np.array(test['img_arr']).to_list())
32         )
33
34         # classify y_hat as either 0 or 1 based on if val is < or >= to 0.5
35         thresh = 0.5
36         y_hat = (y_hat_tmp > thresh).astype(np.int) # cast 0 or 1 to y_hat
37         y_t = test.target.astype(int)
38         cm_vals = confusion_matrix(y_t, y_hat) # get confusion matrix values
39
40         # plot confusion matrix values
41         sns.heatmap(
42             cm_vals,
43             annot=True,
44             cmap='Blues',
45             fmt='0.5g'
46         )
47         plt.xlabel('Predicted Values')
48         plt.ylabel('True Values')
49         plt.title('Baseline Model Confusion Matrix')
50         plt.show()
51
52         # display classification report
53         print(classification_report(y_t, y_hat))

```


2.2 Baseline model

Going to create simple CNN architecture for the first model.

2.2.1 Define Callbacks

```
In [13]: 1 # Going to use EarlyStopping Callback
2 es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)
3
4 # Instantiate ModelCheckpoint Callback
5 mc = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min',
6                     save_best_only=True)
```

2.2.2 Create and fit the model

Build out simple model.

```
In [ ]: 1 model = models.Sequential()
2 input_shape = (512, 512, 3)
3
4 model.add(Conv2D(filters=32, kernel_size = (3,3), strides =2, padding = 'sa
5                 activation = 'relu', input_shape = input_shape))
6 model.add(Conv2D(filters=64, kernel_size = (3,3), strides =2, padding = 'sa
7                 activation = 'relu'))
8 model.add(Conv2D(filters=32, kernel_size = (3,3), strides =2, padding = 'sa
9                 activation = 'relu'))
10 model.add(Flatten())
11 model.add(Dense(128, activation = 'relu'))
12 model.add(Dropout(0.25))
13
14 model.add(Dense(1))
15
16 model.compile(
17     loss = tf.keras.losses.MeanSquaredError(),
18     metrics = [tf.keras.metrics.RootMeanSquaredError(name="rmse"), 'mae'],
19     optimizer = tf.keras.optimizers.Adam(1e-3)
20 )
21
22 model.summary()
```

```
In [ ]: 1 history = model.fit(
2     x=np.array(train['img_arr'].to_list()),
3     y=train['target'],
4     batch_size=16,
5     epochs=50,
6     callbacks=[es, mc],
7     validation_data=(np.array(test['img_arr'].to_list()), test['target'])
8 )
```



2.2.3 Evaluate the model

In []:

```
1 evaluate_model(history, model, test)
```

Plots for RMSE and loss are not particularly helpful because of the extremely high values on the first model epoch, but I will not choose to correct these failures at the moment because the model did not perform well anyways... like other models so far, this model is almost always choosing CE category. It seems like no signal is detected in this model.



2.3 Dropout regularization

Goal for this model is to try some overfitting reduction strategies.



2.3.1 Create and fit the model

```

In [ ]: 1 model = models.Sequential()
        2
        3 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
        4                     input_shape=(512, 512, 3)))
        5 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
        6 model.add(MaxPooling2D((2, 2)))
        7
        8 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
        9 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
        10 model.add(MaxPooling2D(2, 2))
        11
        12 model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
        13 model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
        14 model.add(MaxPooling2D((2, 2)))
        15
        16 model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
        17 model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
        18 model.add(MaxPooling2D((2, 2)))
        19
        20 # add dropout regularization
        21 model.add(layers.Flatten())
        22 model.add(layers.Dense(512, activation='relu'))
        23 model.add(layers.Dropout(0.3))
        24 model.add(layers.Dense(512, activation='relu'))
        25 model.add(layers.Dropout(0.3))
        26
        27 model.add(layers.Dense(1))
        28
        29 # Compile the model
        30 model.compile(
        31     loss = tf.keras.losses.MeanSquaredError(),
        32     metrics = [tf.keras.metrics.RootMeanSquaredError(name="rmse"), 'mae'],
        33     optimizer = tf.keras.optimizers.Adam(1e-3)
        34 )
        35
        36 model.summary()

```

```

In [ ]: 1 history = model.fit(
        2     x=np.array(train['img_arr'].to_list()),
        3     y=train['target'],
        4     batch_size=64,
        5     epochs=50,
        6     callbacks=[es, mc],
        7     validation_data=(np.array(test['img_arr'].to_list()), test['target'])
        8 )

```

▼ 2.3.2 Evaluate the model

```

In [ ]: 1 evaluate_model(history, model, test)

```

Model again seems to not have done anything useful.

2.4 EfficientNet model

Kaggle users for this competition have noted that EfficientNet seems to be generating some success. I will go ahead and try this model out.

Background on EfficientNet: EfficientNet is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient. The scaling parameter is compound in that it is a constant ratio between the dimensions.

```
In [7]: 1 from tensorflow.keras.applications import EfficientNetB0 # 224x224
        2 from tensorflow.keras.applications import EfficientNetB7 # 600x600
```

Probably will not use EffNetB7 because of OOM issues.

2.4.1 Image transformations

Need to preprocess the images as 224x224 for EfficientNetB0

```
In [8]: 1 def preprocess(image_path):
        2     slide = OpenSlide(image_path) # create OpenSlide image object
        3     region = (0, 0) # start at bottom left corner
        4     size = (10000, 10000) # size is square 10kx10k pixels starting from
        5     image = slide.read_region(region, 0, size) # read image with params
        6     image = image.resize(size=(224,224)).convert('RGB') # resizing to 224x224
        7     image_arr = np.array(image) # convert image to array
        8     return image_arr
```

Function will convert images to 3 channels and resize to 224x224 px. Then converts images to arrays to avoid OOM.

```
In [9]: 1 # apply function to images
        2
        3 # create copy of train_df to apply changes to
        4 tdf2 = train_df.copy()
        5 image_arrays2 = []
        6
        7 for i in tqdm(train_df['file_path']):
        8     img_a = preprocess(i)
        9     image_arrays2.append(img_a)
       10
       11 tdf2['img_arr'] = image_arrays2
       12
       13 tdf2.head()
```

100%|██████████| 748/748 [38:25<00:00, 3.08s/it]

Out[9]:

	image_id	center_id	patient_id	image_num	label	file_path	i
0	006388_0	11	006388	0	CE	C:\\Users\\18016\\Downloads\\train\\006388_0.tif	
1	008e5c_0	11	008e5c	0	CE	C:\\Users\\18016\\Downloads\\train\\008e5c_0.tif	
2	00c058_0	11	00c058	0	LAA	C:\\Users\\18016\\Downloads\\train\\00c058_0.tif	
3	01adc5_0	11	01adc5	0	LAA	C:\\Users\\18016\\Downloads\\train\\01adc5_0.tif	
4	026c97_0	4	026c97	0	CE	C:\\Users\\18016\\Downloads\\train\\026c97_0.tif	



It does take quite some time to convert the images to arrays.

▼ 2.4.2 Train-test split

```
In [10]: 1 # also make a target column because I will need int labels for modeling
2
3 # 0 - CE & 1 - LAA
4 tdf2["target"] = tdf2["label"].apply(lambda x : 0 if x=="CE" else 1)
5
6 df_tt2 = tdf2.copy()
7 train, test = train_test_split(df_tt2, test_size=0.2, random_state=42,
8                               shuffle=True)
9
10 train
```

Out[10]:

	image_id	center_id	patient_id	image_num	label	file_pa
593	d2c18a_0	11	d2c18a	0	LAA	C:\\Users\\18016\\Downloads\\train\\d2c18a_C
131	2b7304_1	3	2b7304	1	LAA	C:\\Users\\18016\\Downloads\\train\\2b7304_1
44	0d4164_0	11	0d4164	0	CE	C:\\Users\\18016\\Downloads\\train\\0d4164_C
70	15de51_0	4	15de51	0	LAA	C:\\Users\\18016\\Downloads\\train\\15de51_C
670	e9c181_0	10	e9c181	0	CE	C:\\Users\\18016\\Downloads\\train\\e9c181_C
...

	image_id	center_id	patient_id	image_num	label	file_pa
71	162cad_0	11	162cad	0	LAA	C:\\Users\\18016\\Downloads\\train\\162cad_0
106	23d2c1_1	7	23d2c1	1	CE	C:\\Users\\18016\\Downloads\\train\\23d2c1_1
270	56d177_2	7	56d177	2	CE	C:\\Users\\18016\\Downloads\\train\\56d177_2
435	91fee7_0	8	91fee7	0	CE	C:\\Users\\18016\\Downloads\\train\\91fee7_0
102	2268cf_0	11	2268cf	0	CE	C:\\Users\\18016\\Downloads\\train\\2268cf_0

598 rows × 8 columns

▼

2.4.3 Create and fit the model

I will also try some dropout layers to try to avoid overfitting.


```

In [24]: 1 efficient_net = EfficientNetB0(
2         weights='imagenet',
3         input_shape=(224, 224, 3),
4         include_top=False,
5         pooling='max',
6         classes=2
7     )
8
9     model = models.Sequential()
10    model.add(efficient_net)
11    model.add(Dense(units=112, activation='relu'))
12    model.add(Dropout(0.25))
13    model.add(Dense(units=224, activation = 'relu'))
14    model.add(Dropout(0.25))
15    model.add(Dense(1))
16
17    import tensorflow as tf
18
19    model.compile(
20        loss = 'binary_crossentropy',
21        metrics = [tf.keras.metrics.RootMeanSquaredError(name="rmse"), 'mae'],
22        optimizer = tf.keras.optimizers.Adam(1e-6)
23    )
24
25    model.summary()

```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
=====		
efficientnetb0 (Functional)	(None, 1280)	4049571
dense_12 (Dense)	(None, 112)	143472
dropout_8 (Dropout)	(None, 112)	0
dense_13 (Dense)	(None, 224)	25312
dropout_9 (Dropout)	(None, 224)	0
dense_14 (Dense)	(None, 1)	225
=====		
Total params: 4,218,580		
Trainable params: 4,176,557		
Non-trainable params: 42,023		

```
In [25]: 1 history = model.fit(
2         x=np.array(train['img_arr'].to_list()),
3         y=train['target'],
4         batch_size=64,
5         epochs=50,
6         #callbacks=[es, mc],
7         validation_data=(np.array(test['img_arr'].to_list()), test['target'])
8     )
```

Epoch 1/50

10/10 [=====] - 57s 5s/step - loss: 5.6984 - rmse: 3.2884 - mae: 2.5250 - val_loss: 4.5564 - val_rmse: 1.2513 - val_mae: 0.9923

Epoch 2/50

10/10 [=====] - 46s 5s/step - loss: 5.6136 - rmse: 3.1062 - mae: 2.3669 - val_loss: 4.7281 - val_rmse: 1.3281 - val_mae: 1.0521

Epoch 3/50

10/10 [=====] - 47s 5s/step - loss: 5.5422 - rmse: 3.2280 - mae: 2.4264 - val_loss: 4.4266 - val_rmse: 1.3824 - val_mae: 1.0894

Epoch 4/50

10/10 [=====] - 46s 5s/step - loss: 5.5358 - rmse: 3.2019 - mae: 2.4313 - val_loss: 4.8046 - val_rmse: 1.3825 - val_mae: 1.1116

Epoch 5/50

10/10 [=====] - 46s 5s/step - loss: 5.9549 - rmse: 3.2062 - mae: 2.4897 - val_loss: 5.1581 - val_rmse: 1.3835 - val_mae: 1.1355

Epoch 6/50

10/10 [=====] - 47s 5s/step - loss: 6.0779 - rmse: 3.1139 - mae: 2.4594 - val_loss: 5.2998 - val_rmse: 1.4236 - val_mae: 1.1717

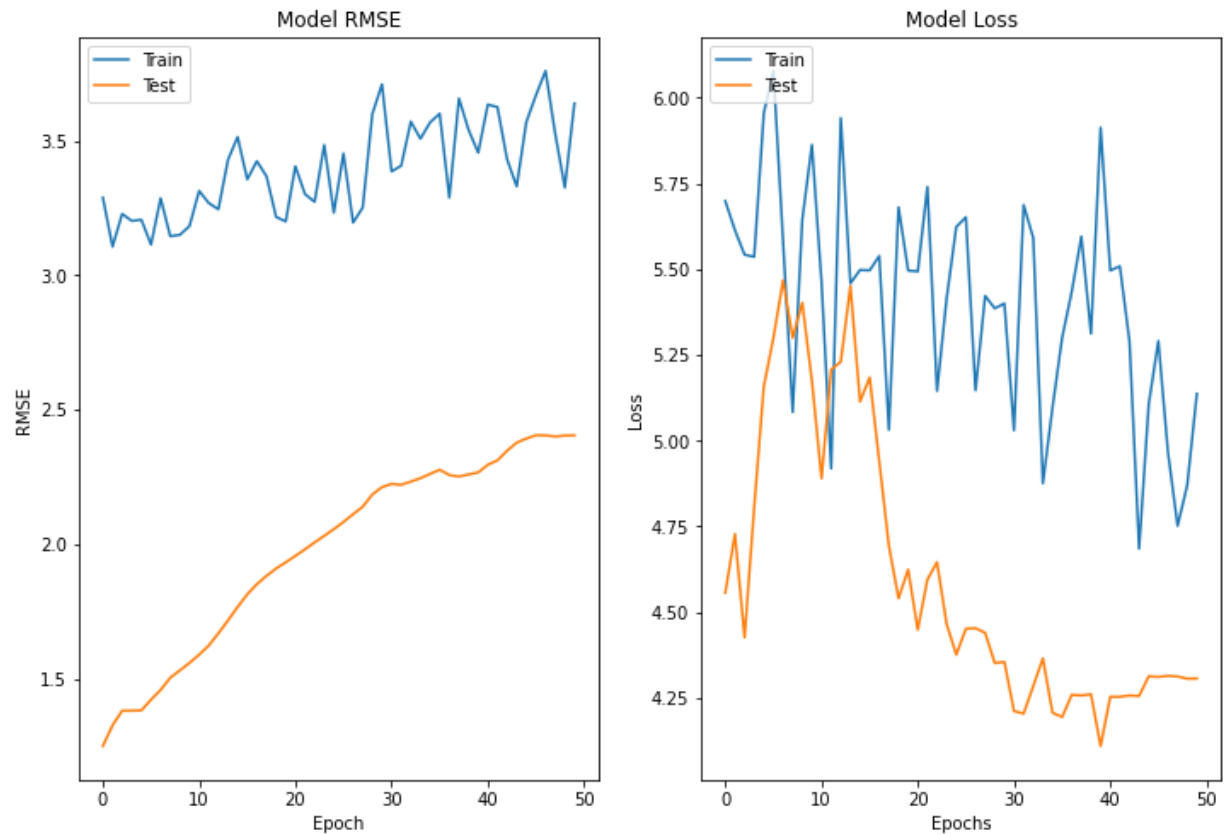
Epoch 7/50

10/10 [=====] - 46s 5s/step - loss: 5.5333 - rmse: 3.1062 - mae: 2.3669 - val_loss: 4.7281 - val_rmse: 1.3281 - val_mae: 1.0521

▼ 2.4.4 Evaluate the model

```
In [26]: 1 evaluate_model(history, model, test)
```

```
5/5 [=====] - 3s 525ms/step - loss: 4.3069 - rmse: 2.4051 - mae: 1.9174  
Test Loss: 4.306916236877441  
Test RMSE: 2.405116558074951  
Test MAE: 1.9173504114151
```

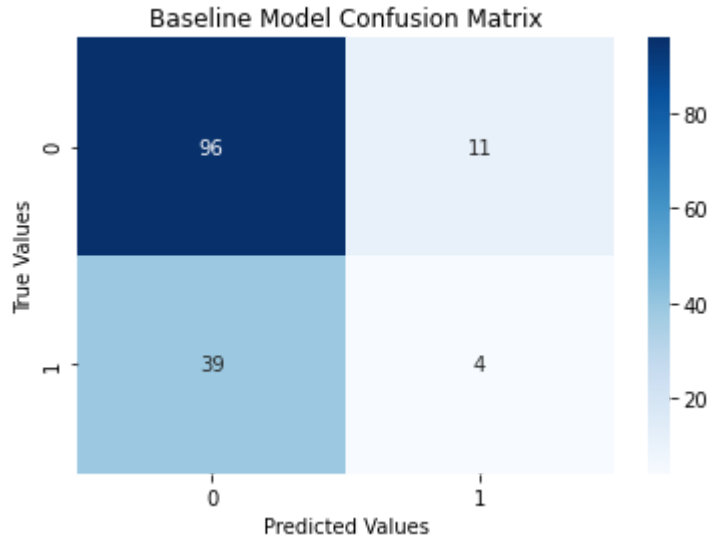


5/5 [=====] - 4s 525ms/step

<ipython-input-21-9e032a6d7b88>:36: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations> (<https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>)

```
y_hat = (y_hat_tmp > thresh).astype(np.int) # cast 0 or 1 to y_hat values
```



	precision	recall	f1-score	support
0	0.71	0.90	0.79	107
1	0.27	0.09	0.14	43
accuracy			0.67	150
macro avg	0.49	0.50	0.47	150
weighted avg	0.58	0.67	0.61	150

Just from CM, it would be intuited that the model is a little bit worse than random guessing. Again, I'm mainly paying attention to the loss function, but it's important to keep track of what the model is doing and where it is classifying images.



2.5 Xception Model

After speaking to my instructor (Abhineet), he recommended that I try Xception. This kind of CNN represents an "intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution)." This quote is from a Francois Chollet [article \(https://arxiv.org/abs/1610.02357\)](https://arxiv.org/abs/1610.02357) describing this model.

In [9]:

```
1 # import the Xception model from Keras
2 from tensorflow.keras.applications.xception import Xception
```



2.5.1 Create and fit the model

```

In [25]: 1 xception_model = Xception(
          2     weights='imagenet',
          3     input_shape=(224, 224, 3),
          4     include_top=False,
          5     pooling='max',
          6     classes=2
          7 )
          8
          9 model = models.Sequential()
         10 model.add(xception_model)
         11 model.add(Dense(units=112, activation='relu'))
         12 model.add(Dropout(0.25))
         13 model.add(Dense(units=224, activation = 'relu'))
         14 model.add(Dropout(0.25))
         15
         16 # sigmoid activation with binary crossentropy loss
         17 model.add(Dense(1, activation = 'sigmoid'))
         18
         19 model.compile(
         20     loss = 'binary_crossentropy',
         21     metrics = [tf.keras.metrics.RootMeanSquaredError(name="rmse"), 'mae'],
         22     optimizer = tf.keras.optimizers.Adam(1e-6)
         23 )
         24
         25 model.summary()

```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
=====		
xception (Functional)	(None, 2048)	20861480
dense_9 (Dense)	(None, 112)	229488
dropout_6 (Dropout)	(None, 112)	0
dense_10 (Dense)	(None, 224)	25312
dropout_7 (Dropout)	(None, 224)	0
dense_11 (Dense)	(None, 1)	225
=====		
Total params: 21,116,505		
Trainable params: 21,061,977		
Non-trainable params: 54,528		

```
In [26]: 1 history = model.fit(  
2         x=np.array(train['img_arr'].to_list()),  
3         y=train['target'],  
4         batch_size=64,  
5         epochs=150,  
6         callbacks=[es, mc],  
7         validation_data=(np.array(test['img_arr'].to_list()), test['target'])  
8     )
```

Epoch 32/150

10/10 [=====] - 99s 10s/step - loss: 0.5657 - rmse: 0.4359 - mae: 0.3734 - val_loss: 0.6266 - val_rmse: 0.4625 - val_mae: 0.4009

Epoch 33/150

10/10 [=====] - 98s 10s/step - loss: 0.5593 - rmse: 0.4312 - mae: 0.3695 - val_loss: 0.6275 - val_rmse: 0.4627 - val_mae: 0.4005

Epoch 34/150

10/10 [=====] - 99s 10s/step - loss: 0.5572 - rmse: 0.4329 - mae: 0.3713 - val_loss: 0.6282 - val_rmse: 0.4629 - val_mae: 0.4006

Epoch 35/150

10/10 [=====] - 97s 10s/step - loss: 0.5478 - rmse: 0.4297 - mae: 0.3690 - val_loss: 0.6286 - val_rmse: 0.4630 - val_mae: 0.4004

Epoch 36/150

10/10 [=====] - 99s 10s/step - loss: 0.5603 - rmse: 0.4346 - mae: 0.3721 - val_loss: 0.6290 - val_rmse: 0.4631 - val_mae: 0.4002

Epoch 37/150

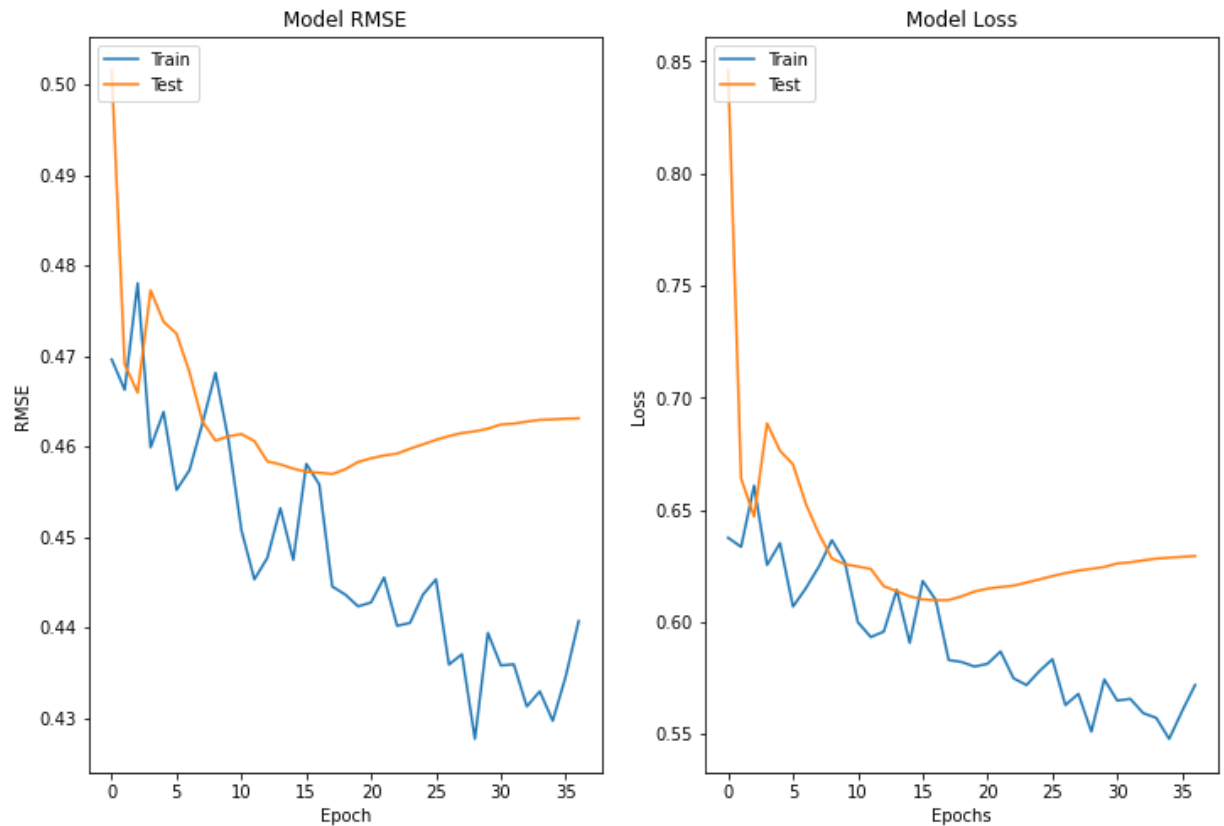
10/10 [=====] - 100s 10s/step - loss: 0.5719 - rmse: 0.4407 - mae: 0.3807 - val_loss: 0.6294 - val_rmse: 0.4631 - val_mae: 0.3999

Epoch 37: early stopping

▼ 2.5.2 Evaluate the model

```
In [27]: 1 evaluate_model(history, model, test)
```

```
5/5 [=====] - 6s 1s/step - loss: 0.6294 - rmse: 0.4631  
- mae: 0.3999  
Test Loss: 0.6293601989746094  
Test RMSE: 0.46309494972229004  
Test MAE: 0.3998658359050751
```

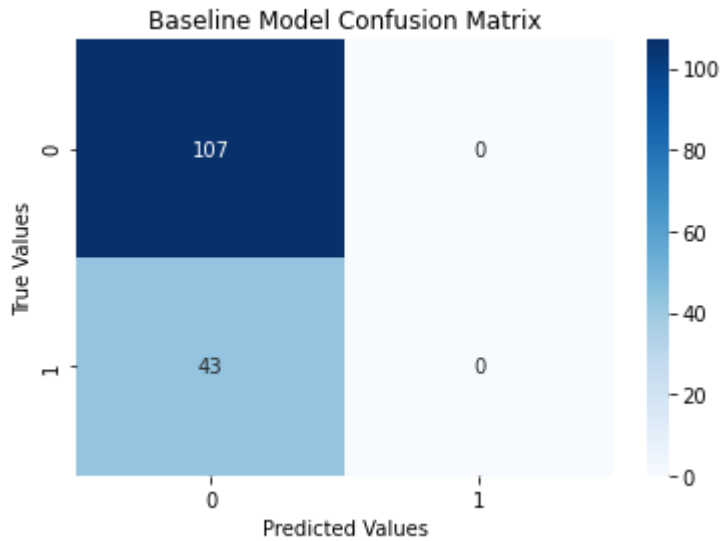


5/5 [=====] - 6s 1s/step

<ipython-input-10-9e032a6d7b88>:36: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations> (<https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>)

```
y_hat = (y_hat_tmp > thresh).astype(np.int) # cast 0 or 1 to y_hat values
```



	precision	recall	f1-score	support
0	0.71	1.00	0.83	107
1	0.00	0.00	0.00	43
accuracy			0.71	150
macro avg	0.36	0.50	0.42	150
weighted avg	0.51	0.71	0.59	150

C:\Users\18016\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics_classification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

```
_warn_prf(average, modifier, msg_start, len(result))
```

The loss is better than baseline, but the model is just predicting CE category for every image. The bottom or valley of the loss and RMSE is very clear from the graphs.

3 Import png image dataset

I think this dataset will be much more manageable to work with, and hopefully higher resolution of each image will be allowed in model iterations involving this dataset. The goal is for higher resolution to capture features within the data.

```
In [3]: 1 # load in the dataset
        2 train_path = r"C:\\Users\\18016\\Desktop\\train_images_cleaned\\"
        3 train_df["file_path"] = train_df["image_id"].apply(lambda x: train_path + x)
        4 train_df.head()
```

```
Out[3]:
```

	image_id	center_id	patient_id	image_num	label	file_path
0	006388_0	11	006388	0	CE	C:\\Users\\18016\\Desktop\\train_images_cleane...
1	008e5c_0	11	008e5c	0	CE	C:\\Users\\18016\\Desktop\\train_images_cleane...
2	00c058_0	11	00c058	0	LAA	C:\\Users\\18016\\Desktop\\train_images_cleane...
3	01adc5_0	11	01adc5	0	LAA	C:\\Users\\18016\\Desktop\\train_images_cleane...
4	026c97_0	4	026c97	0	CE	C:\\Users\\18016\\Desktop\\train_images_cleane...

3.1 Clean the dataset

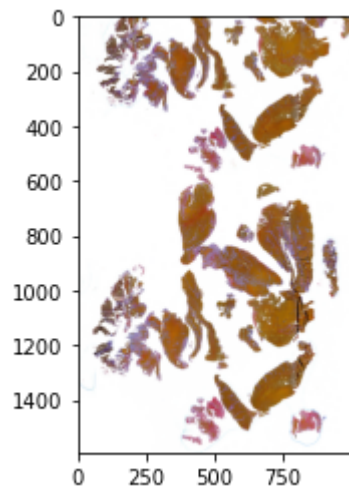
```
In [5]: 1 # follow same process as in 1.3
        2
        3 # remove following ids from train_df:
        4 # b894f4_0, 6baf51_0, 7b9aaa_0, 5adc4c_0, bb06a5_0, and e26a04_0
        5 train_df = train_df[(train_df.image_id != 'b894f4_0') &
        6                       (train_df.image_id != '6baf51_0') &
        7                       (train_df.image_id != '7b9aaa_0') &
        8                       (train_df.image_id != '5adc4c_0') &
        9                       (train_df.image_id != 'bb06a5_0') &
        10                      (train_df.image_id != 'e26a04_0')]
        11
        12 # also, reset the index of train_df
        13 train_df = train_df.reset_index(drop=True)
```

These images are removed due to issues with them being distorted, blurry, or having some other issue. These were mainly identified by Kaggle users.

3.2 Preview an image

```
In [7]: 1 img = Image.open(train_df.file_path[0])  
        2 plt.imshow(img)
```

Out[7]: <matplotlib.image.AxesImage at 0x22c1539e790>



▼ 3.3 Preprocess images

Normalize and permute with ImageDataGenerator, perform train-test-split, and fit datagenerator to train and test set.

```

In [8]: 1 # use ImageDataGenerator
        2 permutes = ImageDataGenerator(
        3     rescale = 1. / 255,      # normalize values btwn 0-1
        4     shear_range = 0.2,      # this distorts the image along an axis
        5     zoom_range = 0.2,       # range for random zoom
        6     horizontal_flip = True  # random horizontal flip
        7 )
        8
        9 # 80-20 standard split
       10 df_tt = train_df.copy()
       11 train, test = train_test_split(df_tt, test_size=0.2, random_state=42,
       12                                 shuffle=True)
       13
       14
       15 # fit the image modifier/permuter
       16 # use target size of 299x299 pixels. That is the default input image size j
       17 # Xception. Maybe it will perform better with this size, but I imagine that
       18 # larger (more resolute) images would perform better in general.
       19 train_gen = permutes.flow_from_dataframe(dataframe=train, x_col='file_path',
       20                                           y_col='label', target_size=(299,299),
       21                                           class_mode='binary', batch_size=16,
       22                                           seed=42)
       23
       24 test_gen = permutes.flow_from_dataframe(dataframe=test, x_col='file_path',
       25                                         y_col='label', target_size=(299,299),
       26                                         class_mode='binary', batch_size=16,
       27                                         seed=42)

```

Found 598 validated image filenames belonging to 2 classes.
 Found 150 validated image filenames belonging to 2 classes.

Image size is 299x299 because Xception defaults to this size. I want to try this size... It's so specific that maybe there is some optimization for it, but I imagine more resolute images work better in general.

▼ 3.4 Xception Model

▼ 3.4.1 Create and fit model

Use the default input size of 299x299 pixels for Xception.

```

In [12]: 1 xception_model = Xception(
          2     weights='imagenet',
          3     input_shape=(299, 299, 3),
          4     include_top=False,
          5     pooling='max',
          6     classes=2
          7 )
          8
          9 # using dropout of 0.5 in hopes of decreasing overfitting.
         10
         11 model = models.Sequential()
         12 model.add(xception_model)
         13 model.add(Dense(units=229, activation='relu'))
         14 model.add(Dropout(0.5))
         15 model.add(Dense(units=458, activation = 'relu'))
         16 model.add(Dropout(0.5))
         17
         18 # sigmoid activation with binary crossentropy loss
         19 model.add(Dense(1, activation = 'sigmoid'))
         20
         21 model.compile(
         22     loss = 'binary_crossentropy',
         23     metrics = [tf.keras.metrics.RootMeanSquaredError(name="rmse"), 'mae'],
         24     optimizer = tf.keras.optimizers.Adam(1e-6)
         25 )
         26
         27 model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
xception (Functional)	(None, 2048)	20861480
dense (Dense)	(None, 229)	469221
dropout (Dropout)	(None, 229)	0
dense_1 (Dense)	(None, 458)	105340
dropout_1 (Dropout)	(None, 458)	0
dense_2 (Dense)	(None, 1)	459
=====		
Total params: 21,436,500		
Trainable params: 21,381,972		
Non-trainable params: 54,528		
=====		

```
In [13]: 1 history = model.fit(
2         x=train_gen,
3         steps_per_epoch=train_gen.n//train_gen.batch_size,
4         validation_data=test_gen,
5         validation_steps=test_gen.n//test_gen.batch_size,
6         epochs=50,
7         callbacks=[es, mc]
8     )
```

Epoch 1/50

37/37 [=====] - 143s 4s/step - loss: 1.4697 - rmse: 0.6656 - mae: 0.5798 - val_loss: 0.9234 - val_rmse: 0.5931 - val_mae: 0.5656

Epoch 2/50

37/37 [=====] - 137s 4s/step - loss: 1.3186 - rmse: 0.6418 - mae: 0.5534 - val_loss: 0.8771 - val_rmse: 0.5774 - val_mae: 0.5558

Epoch 3/50

37/37 [=====] - 139s 4s/step - loss: 1.1789 - rmse: 0.6168 - mae: 0.5296 - val_loss: 0.7878 - val_rmse: 0.5421 - val_mae: 0.5239

Epoch 4/50

37/37 [=====] - 137s 4s/step - loss: 1.1146 - rmse: 0.5980 - mae: 0.5101 - val_loss: 0.7547 - val_rmse: 0.5276 - val_mae: 0.5116

Epoch 5/50

37/37 [=====] - 139s 4s/step - loss: 1.1044 - rmse: 0.6014 - mae: 0.5184 - val_loss: 0.6996 - val_rmse: 0.5015 - val_mae: 0.4836

Epoch 6/50

37/37 [=====] - 139s 4s/step - loss: 0.9832 - rmse: 0.5717 - mae: 0.4884 - val_loss: 0.6807 - val_rmse: 0.4931 - val_mae: 0.4765

Epoch 7/50

37/37 [=====] - 145s 4s/step - loss: 0.8848 - rmse: 0.5418 - mae: 0.4684 - val_loss: 0.6607 - val_rmse: 0.4831 - val_mae: 0.4684

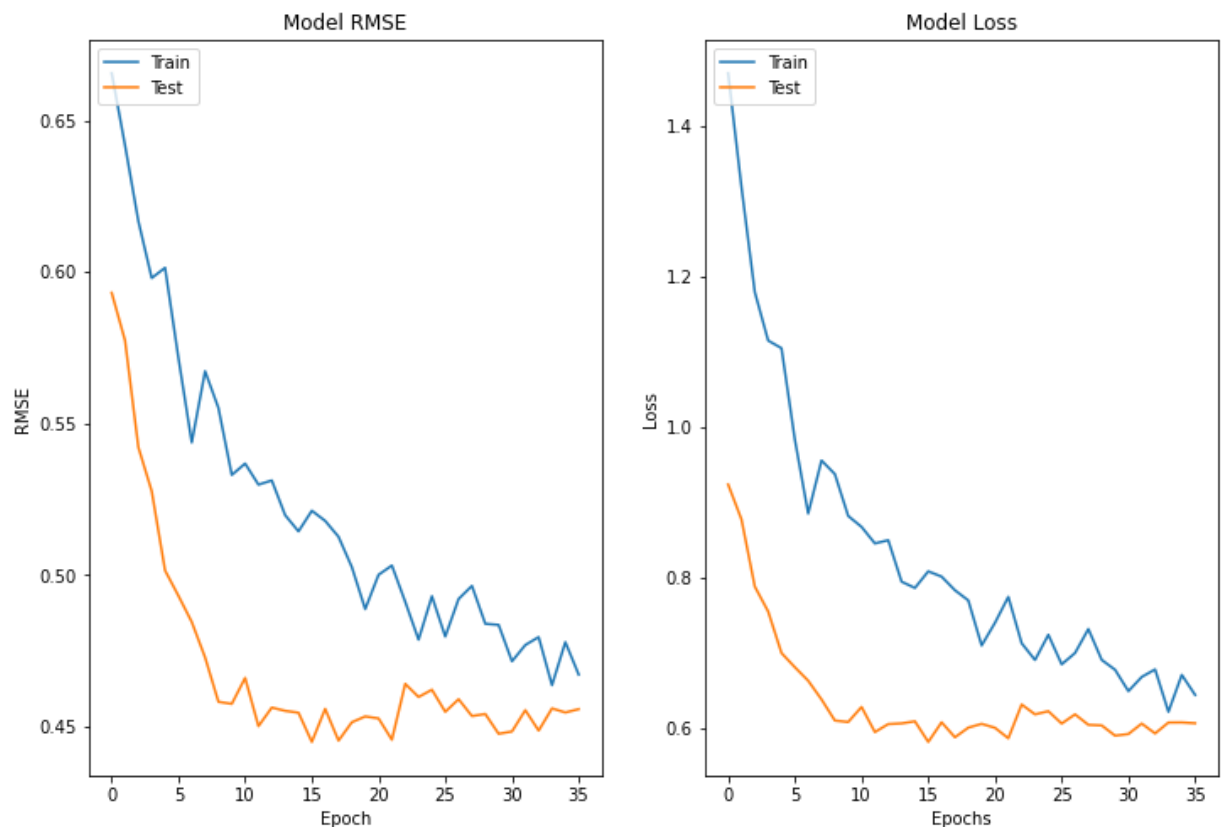
▼ 3.4.2 Evaluate the model

```

In [14]: 1 test_loss, test_rmse, test_mae = history.model.evaluate(test_gen)
2 print(f'Test Loss: {test_loss}')
3 print(f'Test RMSE: {test_rmse}')
4 print(f'Test MAE: {test_mae}')
5
6 # create plots for accuracy and loss
7 fig, ax = plt.subplots(1, 2, figsize = (12,8))
8 ax[0].plot(history.history['rmse'])
9 ax[0].plot(history.history['val_rmse'])
10 ax[0].set_title('Model RMSE')
11 ax[0].set_xlabel('Epoch')
12 ax[0].set_ylabel('RMSE')
13 ax[0].legend(['Train', 'Test'], loc='upper left')
14
15 ax[1].plot(history.history['loss'])
16 ax[1].plot(history.history['val_loss'])
17 ax[1].set_title('Model Loss')
18 ax[1].set_xlabel('Epochs')
19 ax[1].set_ylabel('Loss')
20 ax[1].legend(['Train', 'Test'], loc='upper left')
21
22 plt.show()

```

10/10 [=====] - 9s 869ms/step - loss: 0.6038 - rmse: 0.4541 - mae: 0.3998
 Test Loss: 0.6037670969963074
 Test RMSE: 0.45410773158073425
 Test MAE: 0.3997587561607361



```
In [15]: 1 # need integer labels for more evaluation
          2 df_test_tmp = test.copy()
          3 df_test_tmp.loc[df_test_tmp['label'] == 'CE', 'label'] = 0 # CE is now
          4 df_test_tmp.loc[df_test_tmp['label'] == 'LAA', 'label'] = 1 # LAA is now
```



```

In [16]: 1 # Create predictions for the model
          2 y_hat_tmp = history.model.predict(test_gen)
          3
          4 # classify y_hat as either 0 or 1 based on if val is < or >= to 0.5
          5 thresh = 0.5
          6 y_hat = (y_hat_tmp > thresh).astype(np.int)      # cast 0 or 1 to y_hat values
          7
          8 y_t = df_test_tmp.label.astype(int)
          9
         10 cm_vals = confusion_matrix(y_t, y_hat)      # get confusion matrix values
         11
         12 # plot confusion matrix values
         13 sns.heatmap(
         14     cm_vals,
         15     annot=True,
         16     cmap='Blues',
         17     fmt='0.5g'
         18 )
         19
         20 plt.xlabel('Predicted Values')
         21 plt.ylabel('True Values')
         22 plt.title('Baseline Model Confusion Matrix')
         23 plt.show()

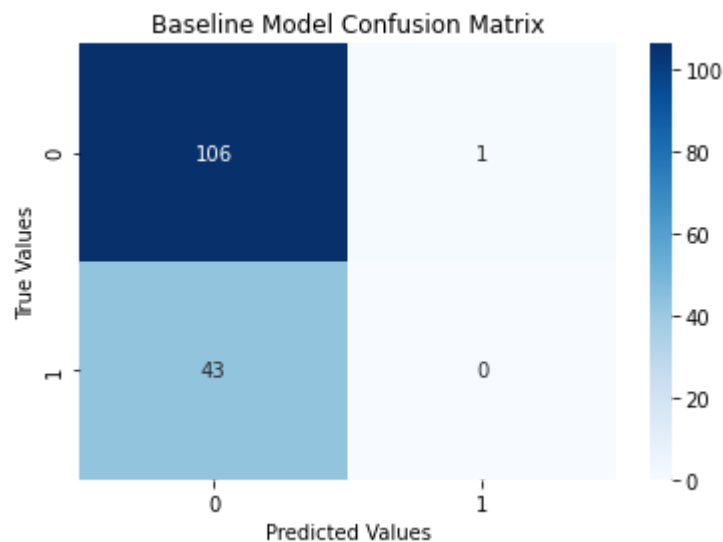
```

10/10 [=====] - 10s 877ms/step

<ipython-input-16-c4d092d549eb>:6: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations> (<https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>)

```
y_hat = (y_hat_tmp > thresh).astype(np.int)      # cast 0 or 1 to y_hat values
```



Unfortunately, this model is guessing CE category almost every time it makes a prediction.

Notice the lower loss here. Lowest loss score thus far.

▼ 3.5 Xception Model retry

Maybe higher resolution will allow for better predictions. I am going to try to double the resolution used in the previous model from 299x299 to 598x598.

```
In [17]: ▼ 1 train_gen = permutest.flow_from_dataframe(dataframe=train, x_col='file_path',  
2 y_col='label', target_size=(598,598),  
3 class_mode='binary', batch_size=16,  
4 seed=42)  
5  
▼ 6 test_gen = permutest.flow_from_dataframe(dataframe=test, x_col='file_path',  
7 y_col='label', target_size=(598,598),  
8 class_mode='binary', batch_size=16,  
9 seed=42)
```

Found 598 validated image filenames belonging to 2 classes.

Found 150 validated image filenames belonging to 2 classes.

▼ 3.5.1 Create and fit model

```

In [18]: 1 xception_model = Xception(
          2     weights='imagenet',
          3     input_shape=(598, 598, 3),
          4     include_top=False,
          5     pooling='max',
          6     classes=2
          7 )
          8
          9 # using dropout of 0.5 in hopes of decreasing overfitting.
         10
         11 model = models.Sequential()
         12 model.add(xception_model)
         13 model.add(Dense(units=299, activation='relu'))
         14 model.add(Dropout(0.5))
         15 model.add(Dense(units=598, activation = 'relu'))
         16 model.add(Dropout(0.5))
         17
         18 # sigmoid activation with binary crossentropy loss
         19 model.add(Dense(1, activation = 'sigmoid'))
         20
         21 model.compile(
         22     loss = 'binary_crossentropy',
         23     metrics = [tf.keras.metrics.RootMeanSquaredError(name="rmse"), 'mae'],
         24     optimizer = tf.keras.optimizers.Adam(1e-6)
         25 )
         26
         27 model.summary()

```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
=====		
xception (Functional)	(None, 2048)	20861480
dense_3 (Dense)	(None, 299)	612651
dropout_2 (Dropout)	(None, 299)	0
dense_4 (Dense)	(None, 598)	179400
dropout_3 (Dropout)	(None, 598)	0
dense_5 (Dense)	(None, 1)	599
=====		
Total params: 21,654,130		
Trainable params: 21,599,602		
Non-trainable params: 54,528		
=====		

```
In [20]: 1 history = model.fit(
2         x=train_gen,
3         steps_per_epoch=train_gen.n//train_gen.batch_size,
4         validation_data=test_gen,
5         validation_steps=test_gen.n//test_gen.batch_size,
6         epochs=50,
7         callbacks=[es, mc]
8     )
```

Epoch 1/50

37/37 [=====] - 612s 16s/step - loss: 1.0309 - rmse: 0.5600 - mae: 0.4393 - val_loss: 0.6205 - val_rmse: 0.4597 - val_mae: 0.3930

Epoch 2/50

37/37 [=====] - 528s 14s/step - loss: 0.9878 - rmse: 0.5541 - mae: 0.4348 - val_loss: 0.6196 - val_rmse: 0.4579 - val_mae: 0.3790

Epoch 3/50

37/37 [=====] - 521s 14s/step - loss: 1.1258 - rmse: 0.5720 - mae: 0.4500 - val_loss: 0.6255 - val_rmse: 0.4598 - val_mae: 0.3652

Epoch 4/50

37/37 [=====] - 518s 14s/step - loss: 0.9677 - rmse: 0.5411 - mae: 0.4190 - val_loss: 0.6160 - val_rmse: 0.4565 - val_mae: 0.3502

Epoch 5/50

37/37 [=====] - 521s 14s/step - loss: 0.8985 - rmse: 0.5249 - mae: 0.4117 - val_loss: 0.6667 - val_rmse: 0.4699 - val_mae: 0.3638

Epoch 6/50

37/37 [=====] - 529s 14s/step - loss: 0.9132 - rmse: 0.5329 - mae: 0.4125 - val_loss: 0.6556 - val_rmse: 0.4645 - val_mae: 0.3535

Epoch 7/50

37/37 [=====] - 522s 14s/step - loss: 0.8740 - rmse: 0.5230 - mae: 0.4087 - val_loss: 0.6631 - val_rmse: 0.4720 - val_mae: 0.3606

Epoch 8/50

37/37 [=====] - 521s 14s/step - loss: 0.9306 - rmse: 0.5287 - mae: 0.4132 - val_loss: 0.6737 - val_rmse: 0.4733 - val_mae: 0.3592

Epoch 9/50

37/37 [=====] - 524s 14s/step - loss: 0.8649 - rmse: 0.5180 - mae: 0.4017 - val_loss: 0.6432 - val_rmse: 0.4661 - val_mae: 0.3593

Epoch 10/50

37/37 [=====] - 523s 14s/step - loss: 0.8810 - rmse: 0.5196 - mae: 0.4032 - val_loss: 0.6697 - val_rmse: 0.4741 - val_mae: 0.3647

Epoch 11/50

37/37 [=====] - 535s 14s/step - loss: 0.8995 - rmse: 0.5239 - mae: 0.4073 - val_loss: 0.6592 - val_rmse: 0.4689 - val_mae: 0.3661

Epoch 12/50

37/37 [=====] - 522s 14s/step - loss: 0.8118 - rmse: 0.5041 - mae: 0.3981 - val_loss: 0.6659 - val_rmse: 0.4744 - val_mae: 0.3679

Epoch 13/50

37/37 [=====] - 520s 14s/step - loss: 0.7760 - rmse: 0.5027 - mae: 0.4059 - val_loss: 0.6364 - val_rmse: 0.4621 - val_mae: 0.3613

Epoch 14/50

37/37 [=====] - 521s 14s/step - loss: 0.8235 - rmse: 0.5136 - mae: 0.4189 - val_loss: 0.6492 - val_rmse: 0.4687 - val_mae: 0.3703

Epoch 15/50

37/37 [=====] - 525s 14s/step - loss: 0.7659 - rmse: 0.4941 - mae: 0.3980 - val_loss: 0.6477 - val_rmse: 0.4678 - val_mae: 0.3669

Epoch 16/50

37/37 [=====] - 573s 16s/step - loss: 0.7541 - rmse:

```
0.4923 - mae: 0.3903 - val_loss: 0.6197 - val_rmse: 0.4554 - val_mae: 0.3563
Epoch 17/50
37/37 [=====] - 597s 16s/step - loss: 0.7516 - rmse:
0.4969 - mae: 0.4033 - val_loss: 0.6469 - val_rmse: 0.4688 - val_mae: 0.3715
Epoch 18/50
37/37 [=====] - 601s 16s/step - loss: 0.7461 - rmse:
0.4885 - mae: 0.3864 - val_loss: 0.6215 - val_rmse: 0.4596 - val_mae: 0.3701
Epoch 19/50
37/37 [=====] - 534s 14s/step - loss: 0.7608 - rmse:
0.4948 - mae: 0.4077 - val_loss: 0.6585 - val_rmse: 0.4727 - val_mae: 0.3777
Epoch 20/50
37/37 [=====] - 535s 14s/step - loss: 0.6838 - rmse:
0.4788 - mae: 0.3870 - val_loss: 0.6453 - val_rmse: 0.4680 - val_mae: 0.3814
Epoch 21/50
37/37 [=====] - 529s 14s/step - loss: 0.7577 - rmse:
0.4965 - mae: 0.4064 - val_loss: 0.6325 - val_rmse: 0.4656 - val_mae: 0.3823
Epoch 22/50
37/37 [=====] - 523s 14s/step - loss: 0.7209 - rmse:
0.4894 - mae: 0.4052 - val_loss: 0.6344 - val_rmse: 0.4644 - val_mae: 0.3819
Epoch 23/50
37/37 [=====] - 526s 14s/step - loss: 0.6997 - rmse:
0.4837 - mae: 0.4034 - val_loss: 0.6164 - val_rmse: 0.4571 - val_mae: 0.3778
Epoch 24/50
37/37 [=====] - 528s 14s/step - loss: 0.7519 - rmse:
0.5009 - mae: 0.4160 - val_loss: 0.6260 - val_rmse: 0.4619 - val_mae: 0.3822
Epoch 24: early stopping
```

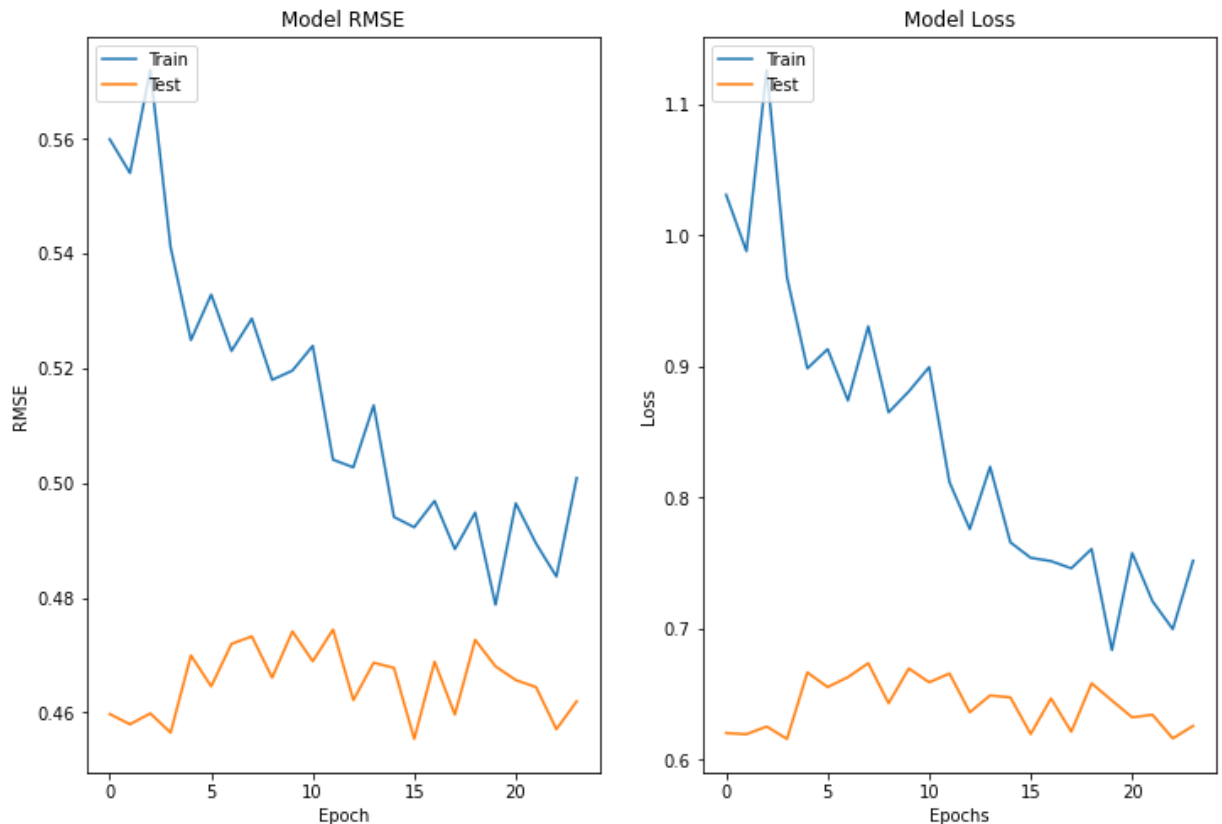
▼ 3.5.2 Evaluate the model

```

In [21]: 1 test_loss, test_rmse, test_mae = history.model.evaluate(test_gen)
2 print(f'Test Loss: {test_loss}')
3 print(f'Test RMSE: {test_rmse}')
4 print(f'Test MAE: {test_mae}')
5
6 # create plots for accuracy and loss
7 fig, ax = plt.subplots(1, 2, figsize = (12,8))
8 ax[0].plot(history.history['rmse'])
9 ax[0].plot(history.history['val_rmse'])
10 ax[0].set_title('Model RMSE')
11 ax[0].set_xlabel('Epoch')
12 ax[0].set_ylabel('RMSE')
13 ax[0].legend(['Train', 'Test'], loc='upper left')
14
15 ax[1].plot(history.history['loss'])
16 ax[1].plot(history.history['val_loss'])
17 ax[1].set_title('Model Loss')
18 ax[1].set_xlabel('Epochs')
19 ax[1].set_ylabel('Loss')
20 ax[1].legend(['Train', 'Test'], loc='upper left')
21
22 plt.show()

```

10/10 [=====] - 24s 2s/step - loss: 0.6063 - rmse: 0.4531 - mae: 0.3778
 Test Loss: 0.6063445806503296
 Test RMSE: 0.4530891180038452
 Test MAE: 0.3777797818183899



The loss and RMSE of the test function is flat throughout the duration but looks like maybe they would eventually converge. The loss is just a bit more than the previous Xception model.


```

In [25]: 1 # need integer labels for more evaluation
2 df_test_tmp = test.copy()
3 df_test_tmp.loc[df_test_tmp['label'] == 'CE', 'label'] = 0 # CE is now
4 df_test_tmp.loc[df_test_tmp['label'] == 'LAA', 'label'] = 1 # LAA is now
5
6 # Create predictions for the model
7 y_hat_tmp = history.model.predict(test_gen)
8
9 # classify y_hat as either 0 or 1 based on if val is < or >= to 0.5
10 thresh = 0.5
11 y_hat = (y_hat_tmp > thresh).astype(np.int) # cast 0 or 1 to y_hat values
12
13 y_t = df_test_tmp.label.astype(int)
14
15 cm_vals = confusion_matrix(y_t, y_hat) # get confusion matrix values
16
17 # plot confusion matrix values
18 sns.heatmap(
19     cm_vals,
20     annot=True,
21     cmap='Blues',
22     fmt='0.5g'
23 )
24
25 plt.xlabel('Predicted Values')
26 plt.ylabel('True Values')
27 plt.title('Baseline Model Confusion Matrix')
28 plt.show()

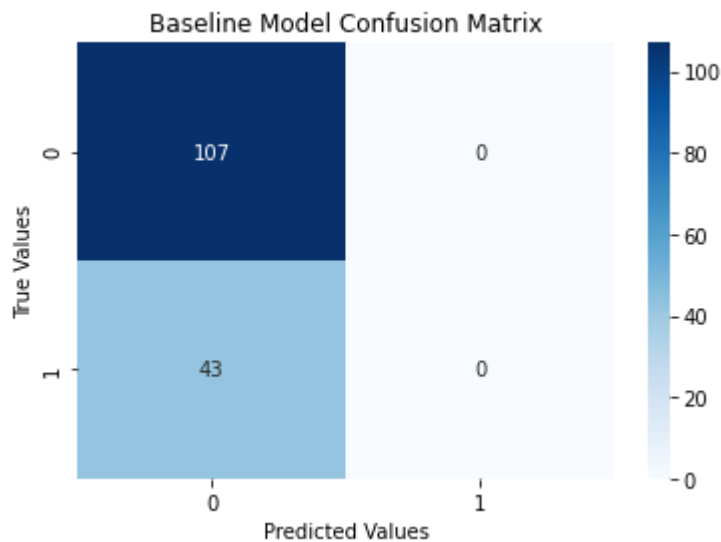
```

10/10 [=====] - 23s 2s/step

<ipython-input-25-14eb97735682>:11: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.

Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations> (<https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>)

```
y_hat = (y_hat_tmp > thresh).astype(np.int) # cast 0 or 1 to y_hat values
```

Again, the model is predicting that each image represents the CE category.

4 Modeling Conclusions

Ultimately the results here are very disappointing. This dataset is exceptionally hard and it's impossible to not lose most of the image's resolution when preprocessing the image because it is so easy to run into OOM issues.

Most of the models seem to essentially be guessing CE category for every validation image. I highly doubt there is any sort feature detection involved in any of these images (i.e. models do not detect any pattern in particular differentiating the two categories).

One of the Xception models performed slightly better in terms of loss than the baseline model. It might be *slightly* more confident in guessing the images when they are CE than the baseline, but it should be assumed that both models are guessing CE for each image with roughly the same confidence. It should be assumed there is not a difference between the performance of these two models especially considering there are no indications of a significant difference in any metric - even .06 or so difference in loss does not indicate significance. This difference could be attributed to randomness in how confident the models are for each individual image in the test set.