Submission Info

- Student name: Kamile Yagci
- Student pace: self paced
- Scheduled project review date/time: TBA
- Instructor name: Claude Fried
- Blog post URL: [https://kamileyagci.github.io/ (https://kamileyagci.github.io/)](https://kamileyagci.github.io/)

Chest X-Ray Image Classification with Deep Learning

Overview

In this study, I analyze the Chest X-ray Images of pediatric patients in order to identify whether or not they have pneumonia. I will apply Image Classification wih Deep Learning using the Convolutional Neural Networks (CNN).

Business Problem

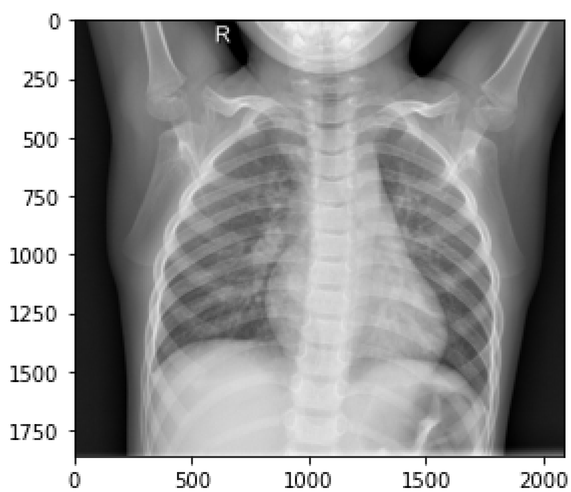
The Baylor Medical Center hired me to improve the accuracy in pneumonia diagnosis on pediatric patients. The study will use the chest X-ray Images of pediatric patients and do the image classification identifying whether the X-ray shows pneuomia or not. The outcome of this study will not only be used at Baylor Centers, but also in partner medical clinics in Africa, where the medical staff is limited. The automated identification system will provide early diagnosis of pediatric patients, so the treatment can start as soon as possible. Moreover it will decrease the human errors in pneumonia diagnosis.

Data

In [313]:
```python
#Import Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from keras.preprocessing.image import ImageDataGenerator, array_to_img, img
from keras import layers
from keras import models
from keras import optimizers
from keras import regularizers

import os, shutil
from datetime import datetime
np.random.seed(123)
```

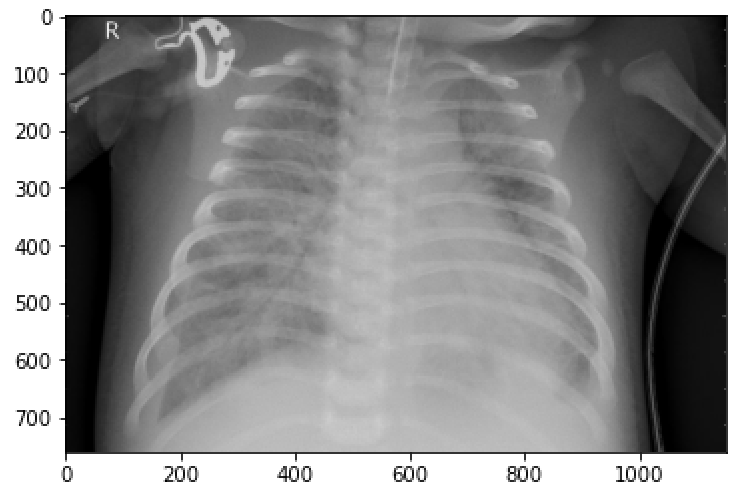
In [13]:
```python
# Sample NORMAL chest X-ray
sample_image_NORMAL = load_img('data/chest_xray/train/NORMAL/IM-0115-0001.j
display(plt.imshow(sample_image_NORMAL))
```

```
<matplotlib.image.AxesImage at 0x7fa98f9d0490>
```



In [9]:
```
# Sample PNEUMONIA Bacterial chest X-ray
sample_image_PNEUMONIA1 = load_img('data/chest_xray/train/PNEUMONIA/person1
display(plt.imshow(sample_image_PNEUMONIA1))
```

```
<matplotlib.image.AxesImage at 0x7fa98ef38880>
```



In [10]:
```
# Sample PNEUMONIA Bacterial chest X-ray
sample_image_PNEUMONIA2 = load_img('data/chest_xray/train/PNEUMONIA/person1
display(plt.imshow(sample_image_PNEUMONIA2))
```

```
<matplotlib.image.AxesImage at 0x7fa98ef8e8e0>
```

In [33]:
```
# Image Dimension
img_to_array(sample_image_NORMAL).shape
```

Out[33]:
```
(1858, 2090, 3)
```

In [263]:

```
# Count image files in directories
import fnmatch
num_train_N = len(fnmatch.filter(os.listdir('data/chest_xray/train/NORMAL')
num_train_P = len(fnmatch.filter(os.listdir('data/chest_xray/train/PNEUMONI
num_val_N = len(fnmatch.filter(os.listdir('data/chest_xray/val/NORMAL'), '*
num_val_P = len(fnmatch.filter(os.listdir('data/chest_xray/val/PNEUMONIA'),
num_test_N = len(fnmatch.filter(os.listdir('data/chest_xray/test/NORMAL'),
num_test_P = len(fnmatch.filter(os.listdir('data/chest_xray/test/PNEUMONIA'

num_train = num_train_N + num_train_P
num_val = num_val_N + num_val_P
num_test = num_test_N + num_test_P
num_all = num_train + num_val + num_test
```

In [264]:

```
print('Image Counts')

print('-------------------')
print("Total number of image files:", num_all)

print('-------------------')
print('# of Train images:', num_train, f'({round((num_train/num_all*100))}%
print('     Train NORMAL:', num_train_N)
print('     Train PNEUMONIA:', num_train_P)

print('-------------------')
print('# of Val images:', num_val, f'({round((num_val/num_all*100))}%)')
print('     Val NORMAL:', num_val_N)
print('     Val PNEUMONIA:', num_val_P)

print('-------------------')
print('# of Test images:', num_test, f'({round((num_test/num_all*100))}%)')
print('     Test NORMAL:', num_test_N)
print('     Test PNEUMONIA:', num_test_P)
```

```
Image Counts
-------------------
Total number of image files: 5856
-------------------
# of Train images: 4098 (70%)
     Train NORMAL: 1107
     Train PNEUMONIA: 2991
-------------------
# of Val images: 879 (15%)
     Val NORMAL: 238
     Val PNEUMONIA: 641
-------------------
# of Test images: 879 (15%)
     Test NORMAL: 238
     Test PNEUMONIA: 641
```

Subset of Data

%15 of the train and val samples

Baseline Model

Run with Convolutional Neural Networks (CNN) without augmentation

In [147]:
```
# Define directories
train_dir = 'data/chest_xray/train'
val_dir = 'data/chest_xray/val'
test_dir = 'data/chest_xray/test'
```

In [208]:
```
# Load the images

train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')

val_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')
```

```
Found 4098 images belonging to 2 classes.
Found 879 images belonging to 2 classes.
```

```
**Baseline Model Design**

* 6 CNN layers
* 2Dense Layers
* Activation function = 'relu' for all layers except output layer
* Activation function = 'sigmoid' for output layer
* Number of units is 32 or 64
```

In [715]:

```python
#Design the model
# This is Baseline Model

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64,
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [210]:

```python
#Compile model

model.compile(loss='binary_crossentropy',
              optimizer= 'sgd',
              metrics=['acc'])
```

In [211]:
```python
# Train model

start=datetime.now()

history = model.fit(train_generator,
                    steps_per_epoch=128, #max 128 (4098/32)
                    epochs=50,
                    validation_data=val_generator,
                    validation_steps=27  #max 27 (879/32)
                   )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/50
128/128 [==============================] - 77s 595ms/step - loss: 0.5936
- acc: 0.7231 - val_loss: 0.5786 - val_acc: 0.7269
Epoch 2/50
 13/128 [==>...........................] - ETA: 55s - loss: 0.5773 - acc:
0.7260

---------------------------------------------------------------------------
--
KeyboardInterrupt                         Traceback (most recent call las
t)
<ipython-input-211-e5ee86a99d6b> in <module>
      3 start=datetime.now()
      4
----> 5 history = model.fit(train_generator,
      6                     steps_per_epoch=128, #max 128 (4098/32)
      7                     epochs=50,

~/opt/anaconda3/lib/python3.8/site-packages/keras/utils/traceback_utils.p
y in error_handler(*args, **kwargs)
     62     filtered_tb = None
     63     try:
---> 64       return fn(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)

~/opt/anaconda3/lib/python3.8/site-packages/keras/engine/training.py in f
it(self, x, y, batch_size, epochs, verbose, callbacks, validation_split,
 validation_data, shuffle, class_weight, sample_weight, initial_epoch, st
eps_per_epoch, validation_steps, validation_batch_size, validation_freq,
 max_queue_size, workers, use_multiprocessing)
   1382                 _r=1):
   1383               callbacks.on_train_batch_begin(step)
-> 1384               tmp_logs = self.train_function(iterator)
   1385               if data_handler.should_sync:
   1386                 context.async_wait()

~/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/util/traceb
ack_utils.py in error_handler(*args, **kwargs)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
```

```
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

~/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/eager/def_f
unction.py in __call__(self, *args, **kwds)
    913
    914       with OptionalXlaContext(self._jit_compile):
--> 915         result = self._call(*args, **kwds)
    916
    917       new_tracing_count = self.experimental_get_tracing_count()

~/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/eager/def_f
unction.py in _call(self, *args, **kwds)
    945       # In this case we have created variables on the first call,
so we run the
    946       # defunned version which is guaranteed to never create vari
ables.
--> 947       return self._stateless_fn(*args, **kwds)  # pylint: disable
=not-callable
    948     elif self._stateful_fn is not None:
    949       # Release the lock early so that multiple threads can perfo
rm the call

~/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/eager/funct
ion.py in __call__(self, *args, **kwargs)
   2954       (graph_function,
   2955        filtered_flat_args) = self._maybe_define_function(args, kw
args)
-> 2956     return graph_function._call_flat(
   2957         filtered_flat_args, captured_inputs=graph_function.captur
ed_inputs)  # pylint: disable=protected-access
   2958

~/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/eager/funct
ion.py in _call_flat(self, args, captured_inputs, cancellation_manager)
   1851         and executing_eagerly):
   1852       # No tape is watching; skip to running the function.
-> 1853       return self._build_call_outputs(self._inference_function.ca
ll(
   1854           ctx, args, cancellation_manager=cancellation_manager))
   1855     forward_backward = self._select_forward_and_backward_function
s(

~/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/eager/funct
ion.py in call(self, ctx, args, cancellation_manager)
    497       with _InterpolateFunctionError(self):
    498         if cancellation_manager is None:
--> 499           outputs = execute.execute(
    500               str(self.signature.name),
    501               num_outputs=self._num_outputs,

~/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/eager/execu
te.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     52   try:
     53     ctx.ensure_initialized()
---> 54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,
op_name,
```

```
     55                                        inputs, attrs, num_output
s)
     56   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

Comments

- The process time will take an hour on my old Macbook Pro. This is too long for baseline model. Therefore I stopped the process.

Train with a smaller data: batch_size=32, steps_per_epoch=30, training sample size = 960

I will run on a smaller training sample. I will control training size with 'steps per epoch'.

In [311]:

```
# Load the images

train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')

val_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')
```

```
Found 4098 images belonging to 2 classes.
Found 879 images belonging to 2 classes.
Found 879 images belonging to 2 classes.
```

In [296]:
```python
#Design model and compile

model_2 = models.Sequential()
model_2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 6
model_2.add(layers.MaxPooling2D((2, 2)))

model_2.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_2.add(layers.MaxPooling2D((2, 2)))

model_2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_2.add(layers.MaxPooling2D((2, 2)))

model_2.add(layers.Flatten())
model_2.add(layers.Dense(64, activation='relu'))
model_2.add(layers.Dense(1, activation='sigmoid'))

model_2.compile(loss='binary_crossentropy',
                optimizer= 'sgd',
                metrics=['acc'])
```

In [297]:
```python
# Train model

start=datetime.now()

history_2 = model_2.fit(train_generator,
                        steps_per_epoch=30, # 30x32 training samaples will run
                        epochs=50,
                        validation_data=val_generator,
                        validation_steps=10  # 10x32 validation samaples will r
                       )

end=datetime.now()

print('Process time:', end-start)
```

```
acc: 0.7083 - val_loss: 0.5989 - val_acc: 0.7063
Epoch 4/50
30/30 [==============================] - 20s 678ms/step - loss: 0.5952 -
acc: 0.7215 - val_loss: 0.5859 - val_acc: 0.7156
Epoch 5/50
30/30 [==============================] - 20s 653ms/step - loss: 0.5755 -
acc: 0.7271 - val_loss: 0.5662 - val_acc: 0.7281
Epoch 6/50
30/30 [==============================] - 20s 669ms/step - loss: 0.5617 -
acc: 0.7240 - val_loss: 0.5443 - val_acc: 0.7469
Epoch 7/50
30/30 [==============================] - 19s 621ms/step - loss: 0.5590 -
acc: 0.7208 - val_loss: 0.5280 - val_acc: 0.7406

Epoch 8/50
30/30 [==============================] - 19s 632ms/step - loss: 0.5156 -
acc: 0.7505 - val_loss: 0.5309 - val_acc: 0.7281
Epoch 9/50
30/30 [==============================] - 21s 690ms/step - loss: 0.5271 -
acc: 0.7219 - val_loss: 0.5295 - val_acc: 0.7781
```

In [445]:
```
model_2.save('saved_model_history/model_2.h5')
# How to open: reconstructed_model = keras.models.load_model("model_2.h5")
```

In [446]:
```
np.save('saved_model_history/history_2.npy', history_2.history)
# How to open: history=np.load('history_2.npy',allow_pickle='TRUE').item()
```

In [447]:
```
def evaluate(model):

    train_results = model.evaluate(train_generator)
    test_results = model.evaluate(test_generator)

    print('----------------------------')
    print('Model Evaluation')
    print('----------------------------')
    print(f'Train Accuracy = {round(train_results[1], 4)}')
    print(f'Train Loss = {round(train_results[0], 4)}')
    print('----------------------------')
    print(f'Test Accuracy = {round(test_results[1], 4)}')
    print(f'Test Loss = {round(test_results[0], 4)}')
```

In [448]:
```
evaluate(model_2)
```

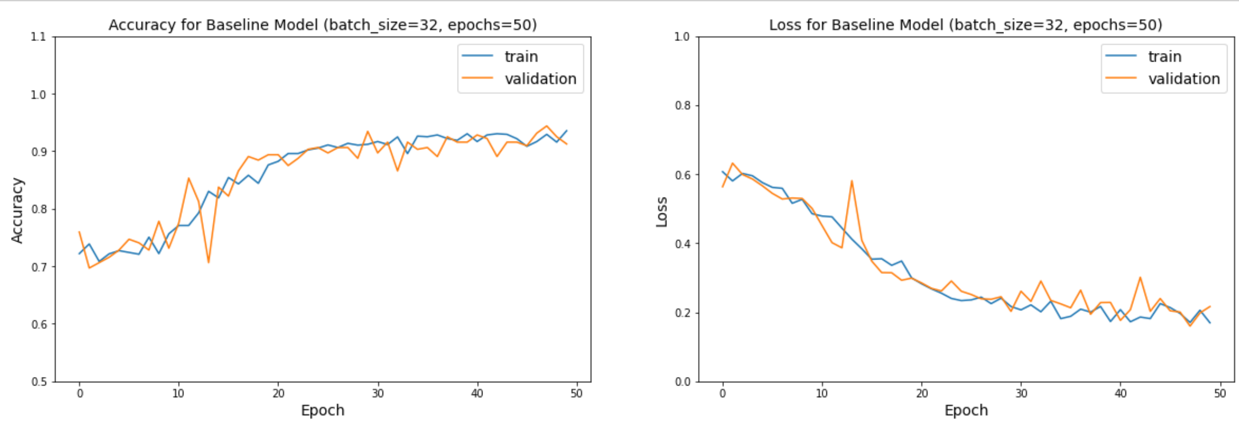
```
129/129 [==============================] - 64s 493ms/step - loss: 0.1628
- acc: 0.9341
28/28 [==============================] - 13s 472ms/step - loss: 0.1740 -
acc: 0.9306
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.9341
Train Loss = 0.1628
----------------------------
Test Accuracy = 0.9306
Test Loss = 0.174
```

In [692]:

```python
# Plot Accuracy and Loss Curves

def plot_acc_loss(history, model_number, title, x_min_acc=0.5, y_max_loss=1

    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc)+1)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

    ax1.plot(acc, label='train')
    ax1.plot(val_acc, label='validation')
    ax1.set_title(f"Accuracy for {title}", fontsize=14)
    ax1.set_xlabel('Epoch', fontsize=14)
    #ax1.set_xticks(epochs)
    ax1.set_ylabel('Accuracy', fontsize=14)
    ax1.set_ylim([x_min_acc, 1.1])
    ax1.legend(fontsize=14)

    ax2.plot(loss, label='train')
    ax2.plot(val_loss, label='validation')
    ax2.set_title(f"Loss for {title}", fontsize=14)
    ax2.set_xlabel('Epoch', fontsize=14)
    #ax2.set_xticks(epochs)
    ax2.set_ylabel('Loss', fontsize=14)
    ax2.set_ylim([0, y_max_loss])
    ax2.legend(fontsize=14)

    plt.savefig(f'images/model_{model_number}_acc_loss_plot.png')
```

In [502]:

```python
plot_acc_loss(history_2, 2, 'Baseline Model (batch_size=32, epochs=50)')
```



Comments

- The results look good, 93% accuracy in train and test data.
- The train and validation curves have very similar trend trend. There are few jumps in validation curve probably due to the small validation sample size.
- No significant overfitting.
- It took 17.5 minutes to complete the training.

Train with a smaller data: batch_size=20, steps_per_epoch=48, training sample size = 960

I would like to train the same baseline model with different hypermeters: batch_size=20, steps_per_epoch=48.

It will again run on same tarining and validation sample size.

In [283]:
```
# Load the images
# batch_size = 20

train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator_b20 = train_datagen.flow_from_directory(
        train_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')

val_generator_b20 = val_datagen.flow_from_directory(
        val_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')
```

```
Found 4098 images belonging to 2 classes.
Found 879 images belonging to 2 classes.
```

In [254]:
```
#Design model and compile

model_3 = models.Sequential()
model_3.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 6
model_3.add(layers.MaxPooling2D((2, 2)))

model_3.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_3.add(layers.MaxPooling2D((2, 2)))

model_3.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_3.add(layers.MaxPooling2D((2, 2)))

model_3.add(layers.Flatten())
model_3.add(layers.Dense(64, activation='relu'))
model_3.add(layers.Dense(1, activation='sigmoid'))

model_3.compile(loss='binary_crossentropy',
              optimizer= 'sgd',
              metrics=['acc'])
```

In [255]:

```
# Train model

start=datetime.now()

history_3 = model_3.fit(train_generator_b20,
                        steps_per_epoch=48, # 48x20 training samaples will run
                        epochs=50,
                        validation_data=val_generator_b20,
                        validation_steps=16  # 16x20 validation samaples will r
                       )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 45/50
48/48 [==============================] - 19s 390ms/step - loss: 0.1342 -
acc: 0.9531 - val_loss: 0.1796 - val_acc: 0.9375
Epoch 46/50
48/48 [==============================] - 18s 376ms/step - loss: 0.1713 -
acc: 0.9333 - val_loss: 0.1814 - val_acc: 0.9438
Epoch 47/50
48/48 [==============================] - 19s 399ms/step - loss: 0.1378 -
acc: 0.9478 - val_loss: 0.2269 - val_acc: 0.9250
Epoch 48/50
48/48 [==============================] - 19s 393ms/step - loss: 0.1367 -
acc: 0.9438 - val_loss: 0.1742 - val_acc: 0.9406
Epoch 49/50
48/48 [==============================] - 18s 380ms/step - loss: 0.1784 -
acc: 0.9375 - val_loss: 0.1656 - val_acc: 0.9250
Epoch 50/50
48/48 [==============================] - 18s 385ms/step - loss: 0.1458 -
acc: 0.9426 - val_loss: 0.1985 - val_acc: 0.9187
Process time: 0:16:31.658962
```

In [449]:

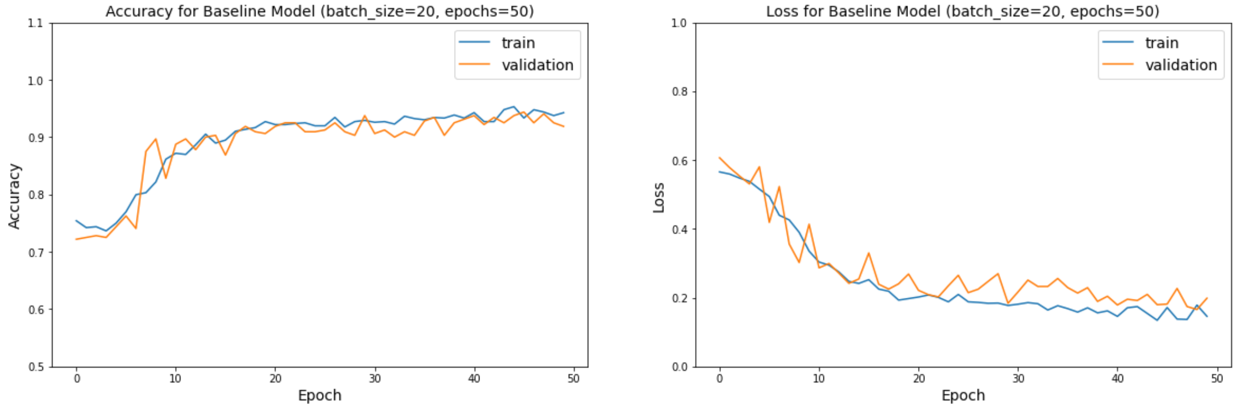
```
model_3.save('saved_model_history/model_3.h5')
np.save('saved_model_history/history_3.npy', history_3.history)
```

In [444]:

```
evaluate(model_3)
```

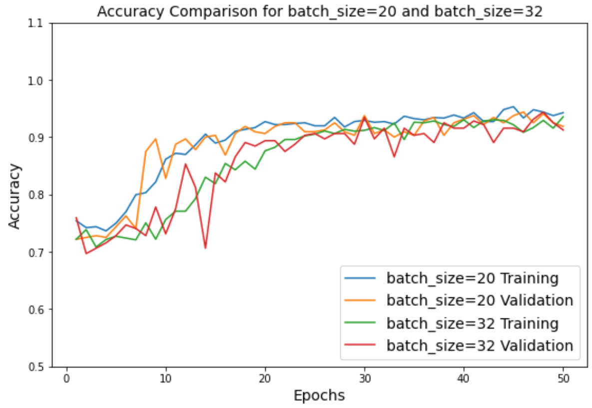
```
129/129 [==============================] - 64s 497ms/step - loss: 0.1469
- acc: 0.9446
28/28 [==============================] - 13s 480ms/step - loss: 0.1648 -
acc: 0.9352
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.9446
Train Loss = 0.1469
----------------------------
Test Accuracy = 0.9352
Test Loss = 0.1648
```

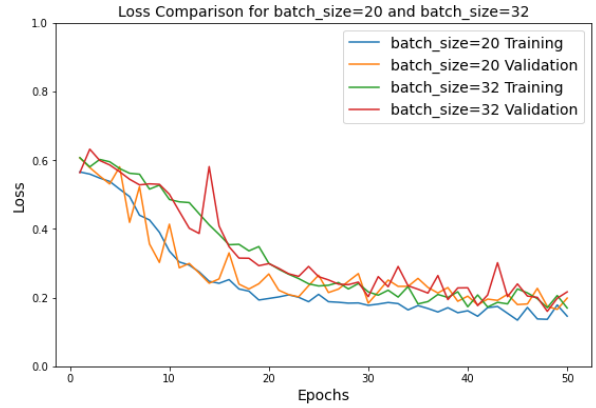
In [503]:
```
plot_acc_loss(history_3, 3, 'Baseline Model (batch_size=20, epochs=50)')
```



In [655]:
```
# Compare models

def compare_models(history1, history2, modelname1, modelname2, epochNumber,

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

    epochs = range(1, epochNumber + 1)

    ax1.plot(epochs, history1.history['acc'][:epochNumber], label=f'{modeln
    ax1.plot(epochs, history1.history['val_acc'][:epochNumber], label=f'{mo
    ax1.plot(epochs, history2.history['acc'][:epochNumber], label=f'{modeln
    ax1.plot(epochs, history2.history['val_acc'][:epochNumber], label=f'{mo
    ax1.set_title(f'Accuracy Comparison for {modelname1} and {modelname2}',
    ax1.set_xlabel('Epochs', fontsize=14)
    ax1.set_ylabel('Accuracy', fontsize=14)
    ax1.set_ylim([x_min_acc, 1.1])
    ax1.legend(loc='lower right', fontsize=14);

    ax2.plot(epochs, history1.history['loss'][:epochNumber], label=f'{model
    ax2.plot(epochs, history1.history['val_loss'][:epochNumber], label=f'{m
    ax2.plot(epochs, history2.history['loss'][:epochNumber], label=f'{model
    ax2.plot(epochs, history2.history['val_loss'][:epochNumber], label=f'{m
    ax2.set_title(f'Loss Comparison for {modelname1} and {modelname2}', fon
    ax2.set_xlabel('Epochs', fontsize=14)
    ax2.set_ylabel('Loss', fontsize=14)
    ax2.set_ylim([0, y_max_loss])
    ax2.legend(loc='upper right', fontsize=14);

    plt.savefig(f'images/compare_{modelname1}_{modelname2}.png')
```

In [657]:
```
compare_models(history_3, history_2, 'batch_size=20', "batch_size=32", 50)
```





Comments

- The result is better with this smaller batch size and larger steps_per_epoch, while training sample size stays same.
- The Loss is smaller and accuracy is larger tarining, validation and testing.
- The graphs shows a little overfitting, especially loss graph (batch_size=20).
- The accuracy starts to flatten after epoch 30. The loss starts to flatten after epcoh 40.
- The training took 16.5 minutes.

Train with smaller number of epochs (epoch = 30)

I have limited computing power. Therefore, I will use smaller number of epochs.

I expect that It will have some negative effect on the results.

In [374]:
```
#Design model and compile

model_4 = models.Sequential()
model_4.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 6
model_4.add(layers.MaxPooling2D((2, 2)))

model_4.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_4.add(layers.MaxPooling2D((2, 2)))

model_4.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_4.add(layers.MaxPooling2D((2, 2)))

model_4.add(layers.Flatten())
model_4.add(layers.Dense(64, activation='relu'))
model_4.add(layers.Dense(1, activation='sigmoid'))

model_4.compile(loss='binary_crossentropy',
              optimizer= 'sgd',
              metrics=['acc'])
```

In [375]:

```
# Train model

start=datetime.now()

history_4 = model_4.fit(train_generator_b20,
                        steps_per_epoch=48, # 48x20 training samaples will run
                        epochs=30,
                        validation_data=val_generator_b20,
                        validation_steps=16  # 16x20 validation samaples will r
                       )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/30
48/48 [==============================] - 23s 477ms/step - loss: 0.5961 -
acc: 0.7244 - val_loss: 0.5997 - val_acc: 0.7094
Epoch 2/30
48/48 [==============================] - 21s 436ms/step - loss: 0.5724 -
acc: 0.7365 - val_loss: 0.5717 - val_acc: 0.7281
Epoch 3/30
48/48 [==============================] - 22s 452ms/step - loss: 0.5639 -
acc: 0.7281 - val_loss: 0.5887 - val_acc: 0.7063
Epoch 4/30
48/48 [==============================] - 22s 447ms/step - loss: 0.5545 -
acc: 0.7229 - val_loss: 0.5428 - val_acc: 0.7250
Epoch 5/30
48/48 [==============================] - 22s 467ms/step - loss: 0.5352 -
acc: 0.7296 - val_loss: 0.5859 - val_acc: 0.7000
Epoch 6/30
48/48 [==============================] - 19s 400ms/step - loss: 0.4793 -
acc: 0.7683 - val_loss: 0.4681 - val_acc: 0.7344
Epoch 7/30
48/48 [==============================] - 22s 460ms/step - loss: 0.4518 -
acc: 0.7875 - val_loss: 0.4604 - val_acc: 0.7281
Epoch 8/30
48/48 [==============================] - 22s 467ms/step - loss: 0.4307 -
acc: 0.8083 - val_loss: 0.3790 - val_acc: 0.8594
Epoch 9/30
48/48 [==============================] - 20s 412ms/step - loss: 0.3723 -
acc: 0.8372 - val_loss: 0.3379 - val_acc: 0.8687
Epoch 10/30
48/48 [==============================] - 20s 406ms/step - loss: 0.3198 -
acc: 0.8719 - val_loss: 0.3631 - val_acc: 0.8531
Epoch 11/30
48/48 [==============================] - 19s 402ms/step - loss: 0.3551 -
acc: 0.8469 - val_loss: 0.3139 - val_acc: 0.8969
Epoch 12/30
48/48 [==============================] - 19s 391ms/step - loss: 0.3179 -
acc: 0.8664 - val_loss: 0.2874 - val_acc: 0.9031
Epoch 13/30
48/48 [==============================] - 19s 395ms/step - loss: 0.2981 -
acc: 0.8781 - val_loss: 0.2974 - val_acc: 0.8875
Epoch 14/30
48/48 [==============================] - 18s 381ms/step - loss: 0.2622 -
acc: 0.8969 - val_loss: 0.2466 - val_acc: 0.8906
```

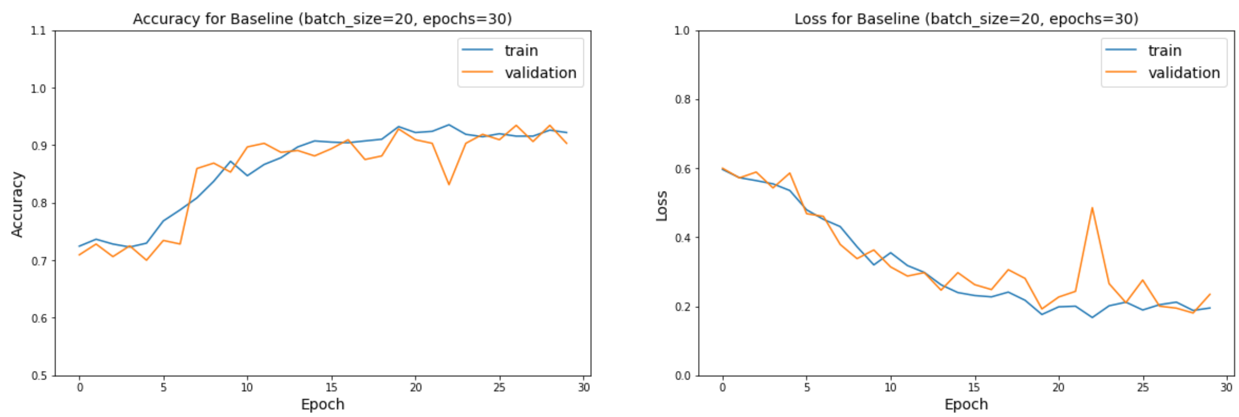
```
Epoch 15/30
48/48 [==============================] - 19s 387ms/step - loss: 0.2396 - acc: 0.9073 - val_loss: 0.2974 - val_acc: 0.8813
Epoch 16/30
48/48 [==============================] - 19s 388ms/step - loss: 0.2311 - acc: 0.9052 - val_loss: 0.2627 - val_acc: 0.8938
Epoch 17/30
48/48 [==============================] - 19s 399ms/step - loss: 0.2272 - acc: 0.9042 - val_loss: 0.2483 - val_acc: 0.9094
Epoch 18/30
48/48 [==============================] - 18s 383ms/step - loss: 0.2410 - acc: 0.9073 - val_loss: 0.3060 - val_acc: 0.8750
Epoch 19/30
48/48 [==============================] - 19s 387ms/step - loss: 0.2173 - acc: 0.9102 - val_loss: 0.2806 - val_acc: 0.8813
Epoch 20/30
48/48 [==============================] - 18s 386ms/step - loss: 0.1762 - acc: 0.9322 - val_loss: 0.1922 - val_acc: 0.9281
Epoch 21/30
48/48 [==============================] - 19s 389ms/step - loss: 0.1984 - acc: 0.9219 - val_loss: 0.2268 - val_acc: 0.9094
Epoch 22/30
48/48 [==============================] - 19s 396ms/step - loss: 0.2002 - acc: 0.9240 - val_loss: 0.2432 - val_acc: 0.9031
Epoch 23/30
48/48 [==============================] - 19s 392ms/step - loss: 0.1674 - acc: 0.9354 - val_loss: 0.4856 - val_acc: 0.8313
Epoch 24/30
48/48 [==============================] - 19s 400ms/step - loss: 0.2014 - acc: 0.9187 - val_loss: 0.2654 - val_acc: 0.9031
Epoch 25/30
48/48 [==============================] - 19s 386ms/step - loss: 0.2118 - acc: 0.9146 - val_loss: 0.2106 - val_acc: 0.9187
Epoch 26/30
48/48 [==============================] - 19s 400ms/step - loss: 0.1892 - acc: 0.9198 - val_loss: 0.2760 - val_acc: 0.9094
Epoch 27/30
48/48 [==============================] - 19s 393ms/step - loss: 0.2045 - acc: 0.9156 - val_loss: 0.2002 - val_acc: 0.9344
Epoch 28/30
48/48 [==============================] - 19s 402ms/step - loss: 0.2120 - acc: 0.9156 - val_loss: 0.1946 - val_acc: 0.9062
Epoch 29/30
48/48 [==============================] - 19s 393ms/step - loss: 0.1879 - acc: 0.9260 - val_loss: 0.1805 - val_acc: 0.9344
Epoch 30/30
48/48 [==============================] - 19s 401ms/step - loss: 0.1950 - acc: 0.9219 - val_loss: 0.2346 - val_acc: 0.9031
Process time: 0:09:49.990940
```

In [476]:
```
model_4.save('saved_model_history/model_4.h5')
np.save('saved_model_history/history_4.npy', history_4.history)
```

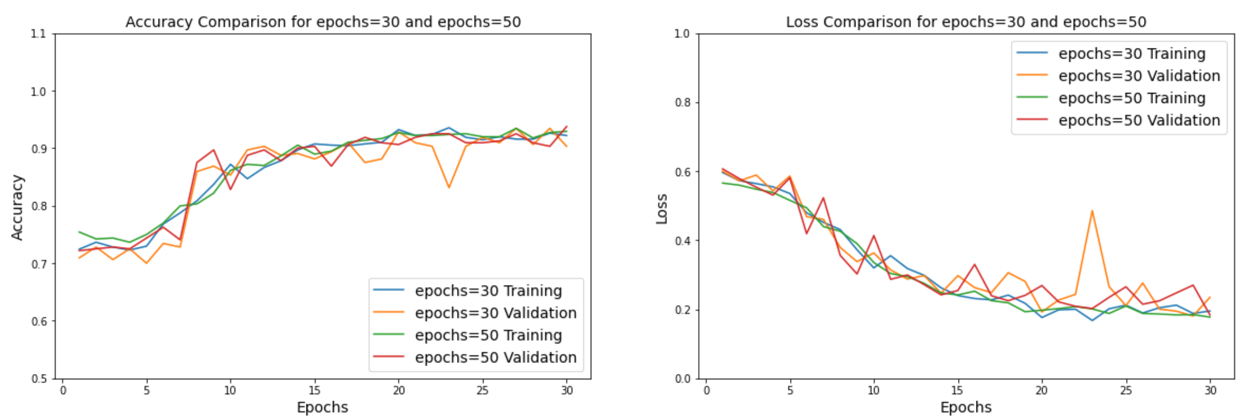
In [477]:
```
evaluate(model_4)
```

```
129/129 [==============================] - 61s 469ms/step - loss: 0.1696
- acc: 0.9370
28/28 [==============================] - 14s 487ms/step - loss: 0.1855 -
acc: 0.9295
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.937
Train Loss = 0.1696
----------------------------
Test Accuracy = 0.9295
Test Loss = 0.1855
```

In [504]:
```
plot_acc_loss(history_4, 4, 'Baseline (batch_size=20, epochs=30)')
```



In [658]:
```
compare_models(history_4, history_3, 'epochs=30', "epochs=50", 30)
```



Comments

- As expected, the smaller epoch number slightly decreased the accuracy and increased the loss both in training and testing.
- For baseline model, the effect is small. However, the effect might be larger in the other models. They may need more epochs to train.
- I will continue to run with 30 epochs considering the computer power limitation.
- The overfitting is not observed in epochs up to 30.
- Runtime is 9.5 minutes.

EarlyStopping

Why not use Early Stopping?

- The validation data is small, so validation loss fluctuates.
- Mostly it stops after epoch=30. The run time is still long.

In [496]:
```
#from keras.callbacks import EarlyStopping
#early_stopping = EarlyStopping(monitor='val_loss', patience=5)
```

Baseline Model + Regularization

I will apply L2 Regularization.

In [353]:
```
#Design model and compile

model_5 = models.Sequential()
model_5.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_regularizer
                          input_shape=(64, 64, 3)))
model_5.add(layers.MaxPooling2D((2, 2)))

model_5.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_regularizer
model_5.add(layers.MaxPooling2D((2, 2)))

model_5.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_regularizer
model_5.add(layers.MaxPooling2D((2, 2)))

model_5.add(layers.Flatten())
model_5.add(layers.Dense(64, activation='relu', kernel_regularizer=regulari
model_5.add(layers.Dense(1, activation='sigmoid'))

model_5.compile(loss='binary_crossentropy',
              optimizer= 'sgd',
              metrics=['acc'])
```

In [354]:
```
# Train model

start=datetime.now()

history_5 = model_5.fit(train_generator_b20,
                        steps_per_epoch=48, # 48x20 training samaples will run
                        epochs=30,
                        validation_data=val_generator_b20,
                        validation_steps=16,  # 16x20 validation samaples will
                       )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/30
48/48 [==============================] - 21s 427ms/step - loss: 1.6276 - acc: 0.7307 - val_loss: 1.6329 - val_acc: 0.7000
Epoch 2/30
48/48 [==============================] - 23s 488ms/step - loss: 1.5894 - acc: 0.7406 - val_loss: 1.5618 - val_acc: 0.7625
Epoch 3/30
48/48 [==============================] - 21s 430ms/step - loss: 1.5738 - acc: 0.7417 - val_loss: 1.5963 - val_acc: 0.7156
Epoch 4/30
48/48 [==============================] - 21s 436ms/step - loss: 1.5811 - acc: 0.7167 - val_loss: 1.5668 - val_acc: 0.7188
Epoch 5/30
48/48 [==============================] - 21s 440ms/step - loss: 1.5598 - acc: 0.7213 - val_loss: 1.5658 - val_acc: 0.7000
Epoch 6/30
48/48 [==============================] - 20s 419ms/step - loss: 1.5256 - acc: 0.7323 - val_loss: 1.5407 - val_acc: 0.7063
Epoch 7/30
48/48 [==============================] - 19s 397ms/step - loss: 1.5096 - acc: 0.7213 - val_loss: 1.4792 - val_acc: 0.7156
Epoch 8/30
48/48 [==============================] - 22s 457ms/step - loss: 1.4553 - acc: 0.7594 - val_loss: 1.4350 - val_acc: 0.7469
Epoch 9/30
48/48 [==============================] - 21s 446ms/step - loss: 1.4306 - acc: 0.7865 - val_loss: 1.5769 - val_acc: 0.6031
Epoch 10/30
48/48 [==============================] - 24s 492ms/step - loss: 1.3669 - acc: 0.8021 - val_loss: 1.3122 - val_acc: 0.8656
Epoch 11/30
48/48 [==============================] - 20s 422ms/step - loss: 1.3697 - acc: 0.7865 - val_loss: 1.3018 - val_acc: 0.8531
Epoch 12/30
48/48 [==============================] - 19s 403ms/step - loss: 1.3172 - acc: 0.8340 - val_loss: 1.2819 - val_acc: 0.8094
Epoch 13/30
48/48 [==============================] - 19s 399ms/step - loss: 1.2667 - acc: 0.8500 - val_loss: 1.2791 - val_acc: 0.8188
Epoch 14/30
48/48 [==============================] - 19s 393ms/step - loss: 1.2676 - acc: 0.8458 - val_loss: 1.2494 - val_acc: 0.8375
```

```
Epoch 15/30
48/48 [==============================] - 18s 385ms/step - loss: 1.1918 - acc: 0.8823 - val_loss: 1.2294 - val_acc: 0.8719
Epoch 16/30
48/48 [==============================] - 18s 385ms/step - loss: 1.1696 - acc: 0.8823 - val_loss: 1.1598 - val_acc: 0.8969
Epoch 17/30
48/48 [==============================] - 19s 393ms/step - loss: 1.1822 - acc: 0.8792 - val_loss: 1.3020 - val_acc: 0.7969
Epoch 18/30
48/48 [==============================] - 20s 416ms/step - loss: 1.1696 - acc: 0.8833 - val_loss: 1.1214 - val_acc: 0.9062
Epoch 19/30
48/48 [==============================] - 19s 404ms/step - loss: 1.1288 - acc: 0.8958 - val_loss: 1.1016 - val_acc: 0.8906
Epoch 20/30
48/48 [==============================] - 20s 418ms/step - loss: 1.0932 - acc: 0.9021 - val_loss: 1.0747 - val_acc: 0.9281
Epoch 21/30
48/48 [==============================] - 19s 387ms/step - loss: 1.0938 - acc: 0.8956 - val_loss: 1.2012 - val_acc: 0.8562
Epoch 22/30
48/48 [==============================] - 19s 399ms/step - loss: 1.0637 - acc: 0.9228 - val_loss: 1.0724 - val_acc: 0.9281
Epoch 23/30
48/48 [==============================] - 20s 416ms/step - loss: 1.0868 - acc: 0.8990 - val_loss: 1.0970 - val_acc: 0.9000
Epoch 24/30
48/48 [==============================] - 18s 384ms/step - loss: 1.0353 - acc: 0.9187 - val_loss: 1.1110 - val_acc: 0.8844
Epoch 25/30
48/48 [==============================] - 22s 466ms/step - loss: 1.0520 - acc: 0.9094 - val_loss: 1.1171 - val_acc: 0.8813
Epoch 26/30
48/48 [==============================] - 19s 390ms/step - loss: 1.0831 - acc: 0.8948 - val_loss: 1.0334 - val_acc: 0.9250
Epoch 27/30
48/48 [==============================] - 21s 431ms/step - loss: 1.0220 - acc: 0.9186 - val_loss: 1.0763 - val_acc: 0.8938
Epoch 28/30
48/48 [==============================] - 21s 437ms/step - loss: 1.0202 - acc: 0.9146 - val_loss: 1.0882 - val_acc: 0.8813
Epoch 29/30
48/48 [==============================] - 19s 398ms/step - loss: 1.0280 - acc: 0.9073 - val_loss: 1.1053 - val_acc: 0.8687
Epoch 30/30
48/48 [==============================] - 19s 387ms/step - loss: 0.9774 - acc: 0.9248 - val_loss: 1.0254 - val_acc: 0.9062
Process time: 0:10:04.192717
```

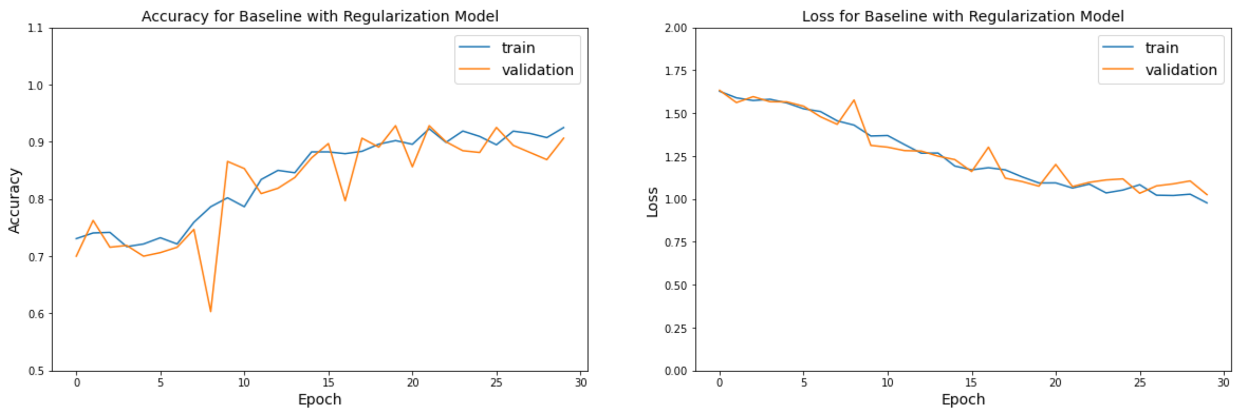
In [493]:

```
model_5.save('saved_model_history/model_5.h5')
np.save('saved_model_history/history_5.npy', history_5.history)
```

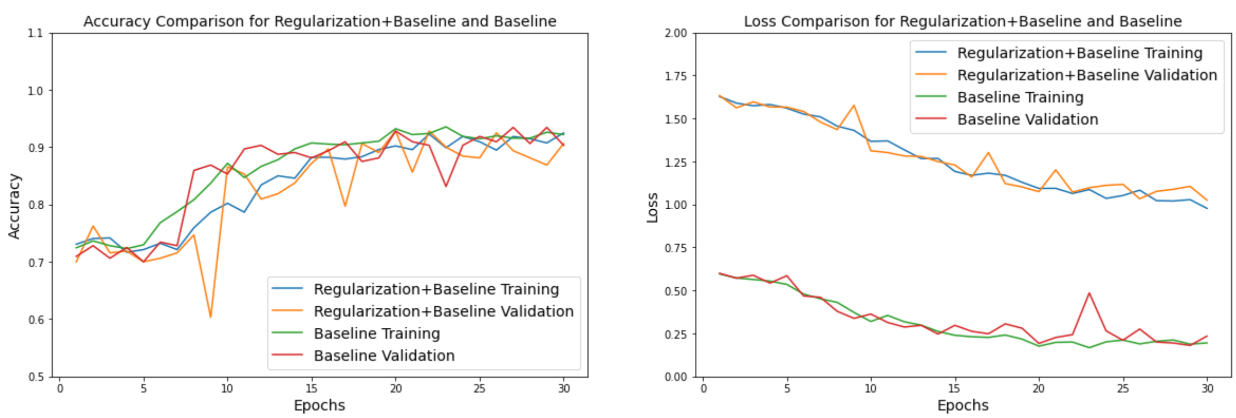
In [494]:
```
evaluate(model_5)
```

```
129/129 [==============================] - 60s 461ms/step - loss: 0.9938
- acc: 0.9212
28/28 [==============================] - 12s 432ms/step - loss: 1.0141 -
acc: 0.8976
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.9212
Train Loss = 0.9938
----------------------------
Test Accuracy = 0.8976
Test Loss = 1.0141
```

In [529]:
```
plot_acc_loss(history_5, 5, 'Baseline with Regularization Model', 2)
```



In [659]:
```
compare_models(history_5, history_4, "Regularization+Baseline", 'Baseline',
```



Comments

- Regularization did improve the slight overfitting.
- However, the model performance with regularization is worse than the baseline model.
- The accuracy in testing decreased significantly.
- The loss values incresaed as a result of the regularization penalty on loss function.
- L2 regularization didn't help much.
- Process time is 10 mins.

L1 Regularization

I tried to train with the L1 Regularization. I run about five epochs, but then stopped the training. The beginning performance was bad compared L2 regularization. The loss value was very high at beginning epochs, way higher than the beginning loss values with L2.

Baseline Model + Dropout Layers

I will add Dropout Layer to mode.

In [377]:
```
model_6 = models.Sequential()
model_6.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 6
model_6.add(layers.MaxPooling2D((2, 2)))
model_6.add(layers.Dropout(0.3))

model_6.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_6.add(layers.MaxPooling2D((2, 2)))
model_6.add(layers.Dropout(0.3))

model_6.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_6.add(layers.MaxPooling2D((2, 2)))
model_6.add(layers.Dropout(0.3))

model_6.add(layers.Flatten())
model_6.add(layers.Dense(64, activation='relu'))
model_6.add(layers.Dropout(0.3))
model_6.add(layers.Dense(1, activation='sigmoid'))

model_6.compile(loss='binary_crossentropy',
              optimizer= 'sgd',
              metrics=['acc'])
```

In [378]:

```
# Train model

start=datetime.now()

history_6 = model_6.fit(train_generator_b20,
                        steps_per_epoch=48, # 48x20 training samaples will run
                        epochs=30,
                        validation_data=val_generator_b20,
                        validation_steps=16,  # 16x20 validation samaples will
                       )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/30
48/48 [==============================] - 20s 404ms/step - loss: 0.6114 -
acc: 0.7240 - val_loss: 0.6156 - val_acc: 0.7469
Epoch 2/30
48/48 [==============================] - 21s 444ms/step - loss: 0.6018 -
acc: 0.7219 - val_loss: 0.6341 - val_acc: 0.7094
Epoch 3/30
48/48 [==============================] - 22s 469ms/step - loss: 0.5893 -
acc: 0.7385 - val_loss: 0.5950 - val_acc: 0.7812
Epoch 4/30
48/48 [==============================] - 21s 443ms/step - loss: 0.5882 -
acc: 0.7260 - val_loss: 0.6417 - val_acc: 0.7219
Epoch 5/30
48/48 [==============================] - 20s 419ms/step - loss: 0.5802 -
acc: 0.7265 - val_loss: 0.6063 - val_acc: 0.7188
Epoch 6/30
48/48 [==============================] - 20s 417ms/step - loss: 0.5704 -
acc: 0.7333 - val_loss: 0.5927 - val_acc: 0.7344
Epoch 7/30
48/48 [==============================] - 23s 473ms/step - loss: 0.5711 -
acc: 0.7344 - val_loss: 0.6276 - val_acc: 0.6938
Epoch 8/30
48/48 [==============================] - 21s 427ms/step - loss: 0.5770 -
acc: 0.7198 - val_loss: 0.6063 - val_acc: 0.7375
Epoch 9/30
48/48 [==============================] - 20s 412ms/step - loss: 0.5491 -
acc: 0.7344 - val_loss: 0.6072 - val_acc: 0.7188
Epoch 10/30
48/48 [==============================] - 24s 499ms/step - loss: 0.5562 -
acc: 0.7244 - val_loss: 0.6157 - val_acc: 0.7531
Epoch 11/30
48/48 [==============================] - 26s 539ms/step - loss: 0.5342 -
acc: 0.7432 - val_loss: 0.5509 - val_acc: 0.7563
Epoch 12/30
48/48 [==============================] - 26s 541ms/step - loss: 0.5179 -
acc: 0.7479 - val_loss: 0.5451 - val_acc: 0.7844
Epoch 13/30
48/48 [==============================] - 23s 471ms/step - loss: 0.5142 -
acc: 0.7365 - val_loss: 0.5595 - val_acc: 0.7937
Epoch 14/30
48/48 [==============================] - 23s 475ms/step - loss: 0.4775 -
acc: 0.7729 - val_loss: 0.5592 - val_acc: 0.8375
```

```
Epoch 15/30
48/48 [==============================] - 22s 461ms/step - loss: 0.4903 -
acc: 0.7656 - val_loss: 0.5217 - val_acc: 0.8375
Epoch 16/30
48/48 [==============================] - 21s 434ms/step - loss: 0.4848 -
acc: 0.7667 - val_loss: 0.4988 - val_acc: 0.8344
Epoch 17/30
48/48 [==============================] - 19s 401ms/step - loss: 0.4600 -
acc: 0.7792 - val_loss: 0.5054 - val_acc: 0.8844
Epoch 18/30
48/48 [==============================] - 19s 389ms/step - loss: 0.4333 -
acc: 0.7948 - val_loss: 0.4646 - val_acc: 0.8750
Epoch 19/30
48/48 [==============================] - 19s 395ms/step - loss: 0.4205 -
acc: 0.7954 - val_loss: 0.4268 - val_acc: 0.8750
Epoch 20/30
48/48 [==============================] - 18s 383ms/step - loss: 0.3862 -
acc: 0.8340 - val_loss: 0.4410 - val_acc: 0.8687
Epoch 21/30
48/48 [==============================] - 19s 401ms/step - loss: 0.3794 -
acc: 0.8271 - val_loss: 0.4428 - val_acc: 0.8594
Epoch 22/30
48/48 [==============================] - 19s 388ms/step - loss: 0.3862 -
acc: 0.8302 - val_loss: 0.4363 - val_acc: 0.8844
Epoch 23/30
48/48 [==============================] - 18s 372ms/step - loss: 0.3632 -
acc: 0.8281 - val_loss: 0.4266 - val_acc: 0.8687
Epoch 24/30
48/48 [==============================] - 18s 380ms/step - loss: 0.3527 -
acc: 0.8542 - val_loss: 0.3319 - val_acc: 0.9250
Epoch 25/30
48/48 [==============================] - 19s 387ms/step - loss: 0.3319 -
acc: 0.8719 - val_loss: 0.4502 - val_acc: 0.8562
Epoch 26/30
48/48 [==============================] - 18s 378ms/step - loss: 0.3281 -
acc: 0.8542 - val_loss: 0.3852 - val_acc: 0.8813
Epoch 27/30
48/48 [==============================] - 19s 388ms/step - loss: 0.3284 -
acc: 0.8635 - val_loss: 0.4195 - val_acc: 0.8687
Epoch 28/30
48/48 [==============================] - 19s 389ms/step - loss: 0.3529 -
acc: 0.8313 - val_loss: 0.4199 - val_acc: 0.8594
Epoch 29/30
48/48 [==============================] - 18s 385ms/step - loss: 0.3000 -
acc: 0.8698 - val_loss: 0.3962 - val_acc: 0.8625
Epoch 30/30
48/48 [==============================] - 19s 395ms/step - loss: 0.3183 -
acc: 0.8667 - val_loss: 0.4164 - val_acc: 0.8406
Process time: 0:10:12.917649
```

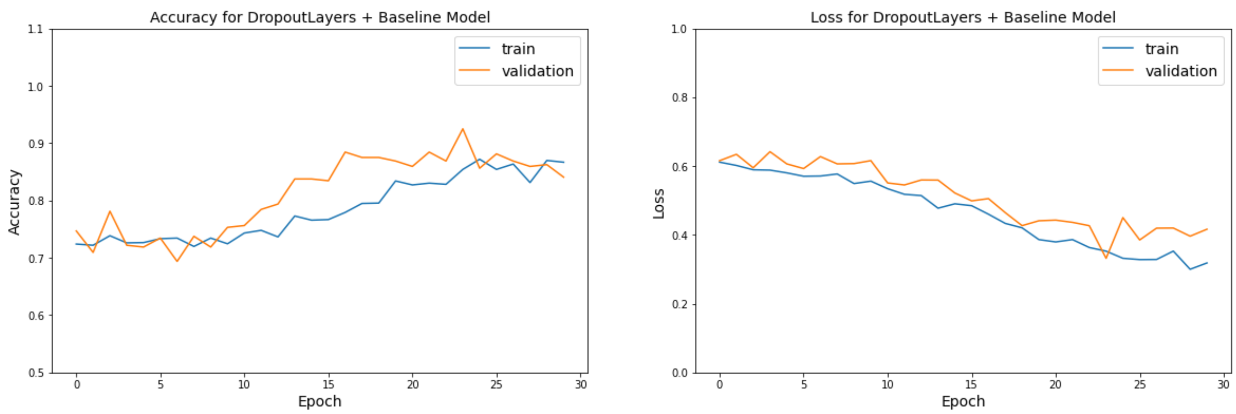
In [519]:

```
model_6.save('saved_model_history/model_6.h5')
np.save('saved_model_history/history_6.npy', history_6.history)
```

```
In [520]: evaluate(model_6)
```

```
129/129 [==============================] - 57s 441ms/step - loss: 0.3846
- acc: 0.8694
28/28 [==============================] - 13s 451ms/step - loss: 0.4017 -
acc: 0.8464
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.8694
Train Loss = 0.3846
----------------------------
Test Accuracy = 0.8464
Test Loss = 0.4017
```

```
In [524]: plot_acc_loss(history_6, 6, 'DropoutLayers + Baseline Model')
```




```
In [660]: compare_models(history_6, history_4, "DropoutLayers+Baseline", 'Baseline',
```

Comments

- The result with Dropout is a worse than the baseline model. The accuracy in training and testing decreased significantly.
- Interestingly, the validation performance looks better than training.
- I will not use Dropout Layers in my model.
- Process time is 10 mins.

Baseline Model + Augmentation

The baseline model performed better without Regularization and Dropout Layers.

I will not train data with Augmentation.

In [528]:
```
# Load the training images with Augmentation
# batch_size = 20

train_datagen_aug = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator_b20_aug = train_datagen_aug.flow_from_directory(
        train_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')

val_generator_b20 = val_datagen.flow_from_directory(
        val_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')
```

```
Found 4098 images belonging to 2 classes.
Found 879 images belonging to 2 classes.
```

In [404]:
```
# Design Model

model_7 = models.Sequential()
model_7.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 6
model_7.add(layers.MaxPooling2D((2, 2)))

model_7.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_7.add(layers.MaxPooling2D((2, 2)))

model_7.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_7.add(layers.MaxPooling2D((2, 2)))

model_7.add(layers.Flatten())
model_7.add(layers.Dense(64, activation='relu'))
model_7.add(layers.Dense(1, activation='sigmoid'))

model_7.compile(loss='binary_crossentropy',
              optimizer= 'sgd',
              metrics=['acc'])
```

In [405]:

```
# Train model

start=datetime.now()

history_7 = model_7.fit(train_generator_b20_aug,
                        steps_per_epoch=48, # 48x20 training samaples will run
                        epochs=30,
                        validation_data=val_generator_b20,
                        validation_steps=16,  # 16x20 validation samaples will
                       )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/30
48/48 [==============================] - 21s 423ms/step - loss: 0.6136 -
acc: 0.7146 - val_loss: 0.5737 - val_acc: 0.7469
Epoch 2/30
48/48 [==============================] - 20s 416ms/step - loss: 0.5954 -
acc: 0.7219 - val_loss: 0.5662 - val_acc: 0.7469
Epoch 3/30
48/48 [==============================] - 20s 421ms/step - loss: 0.5757 -
acc: 0.7354 - val_loss: 0.6501 - val_acc: 0.6562
Epoch 4/30
48/48 [==============================] - 21s 432ms/step - loss: 0.5914 -
acc: 0.7104 - val_loss: 0.5697 - val_acc: 0.7250
Epoch 5/30
48/48 [==============================] - 21s 435ms/step - loss: 0.5908 -
acc: 0.7073 - val_loss: 0.5690 - val_acc: 0.7188
Epoch 6/30
48/48 [==============================] - 23s 473ms/step - loss: 0.5481 -
acc: 0.7406 - val_loss: 0.5487 - val_acc: 0.7281
Epoch 7/30
48/48 [==============================] - 22s 468ms/step - loss: 0.5495 -
acc: 0.7281 - val_loss: 0.5359 - val_acc: 0.7406
Epoch 8/30
48/48 [==============================] - 20s 423ms/step - loss: 0.5189 -
acc: 0.7458 - val_loss: 0.5488 - val_acc: 0.8313
Epoch 9/30
48/48 [==============================] - 20s 409ms/step - loss: 0.5155 -
acc: 0.7458 - val_loss: 0.4265 - val_acc: 0.7812
Epoch 10/30
48/48 [==============================] - 20s 421ms/step - loss: 0.5268 -
acc: 0.7380 - val_loss: 0.4313 - val_acc: 0.8344
Epoch 11/30
48/48 [==============================] - 20s 415ms/step - loss: 0.4665 -
acc: 0.7875 - val_loss: 0.4155 - val_acc: 0.8719
Epoch 12/30
48/48 [==============================] - 20s 423ms/step - loss: 0.4475 -
acc: 0.8052 - val_loss: 0.3669 - val_acc: 0.8469
Epoch 13/30
48/48 [==============================] - 20s 419ms/step - loss: 0.4185 -
acc: 0.8208 - val_loss: 0.3780 - val_acc: 0.8562
Epoch 14/30
48/48 [==============================] - 20s 408ms/step - loss: 0.3817 -
```

```
acc: 0.8396 - val_loss: 0.6545 - val_acc: 0.6313
Epoch 15/30
48/48 [==============================] - 20s 415ms/step - loss: 0.3993 -
acc: 0.8271 - val_loss: 0.3492 - val_acc: 0.8594
Epoch 16/30
48/48 [==============================] - 20s 413ms/step - loss: 0.3634 -
acc: 0.8340 - val_loss: 0.3562 - val_acc: 0.8656
Epoch 17/30
48/48 [==============================] - 20s 411ms/step - loss: 0.3481 -
acc: 0.8455 - val_loss: 0.2863 - val_acc: 0.8969
Epoch 18/30
48/48 [==============================] - 22s 456ms/step - loss: 0.3431 -
acc: 0.8455 - val_loss: 0.2650 - val_acc: 0.9094
Epoch 19/30
48/48 [==============================] - 23s 479ms/step - loss: 0.3242 -
acc: 0.8583 - val_loss: 0.4260 - val_acc: 0.7937
Epoch 20/30
48/48 [==============================] - 21s 428ms/step - loss: 0.3141 -
acc: 0.8677 - val_loss: 0.4472 - val_acc: 0.8031
Epoch 21/30
48/48 [==============================] - 20s 422ms/step - loss: 0.3027 -
acc: 0.8844 - val_loss: 0.2446 - val_acc: 0.9094
Epoch 22/30
48/48 [==============================] - 20s 414ms/step - loss: 0.3048 -
acc: 0.8716 - val_loss: 0.2465 - val_acc: 0.9062
Epoch 23/30
48/48 [==============================] - 22s 463ms/step - loss: 0.2915 -
acc: 0.8719 - val_loss: 0.2308 - val_acc: 0.9125
Epoch 24/30
48/48 [==============================] - 23s 468ms/step - loss: 0.3131 -
acc: 0.8698 - val_loss: 0.3528 - val_acc: 0.8562
Epoch 25/30
48/48 [==============================] - 20s 414ms/step - loss: 0.2994 -
acc: 0.8716 - val_loss: 0.2974 - val_acc: 0.8687
Epoch 26/30
48/48 [==============================] - 22s 466ms/step - loss: 0.2727 -
acc: 0.8823 - val_loss: 0.2705 - val_acc: 0.8844
Epoch 27/30
48/48 [==============================] - 20s 421ms/step - loss: 0.2818 -
acc: 0.8771 - val_loss: 0.2127 - val_acc: 0.9187
Epoch 28/30
48/48 [==============================] - 21s 430ms/step - loss: 0.2770 -
acc: 0.8771 - val_loss: 0.3095 - val_acc: 0.8781
Epoch 29/30
48/48 [==============================] - 20s 416ms/step - loss: 0.2837 -
acc: 0.8716 - val_loss: 0.2602 - val_acc: 0.9031
Epoch 30/30
48/48 [==============================] - 20s 422ms/step - loss: 0.2714 -
acc: 0.8914 - val_loss: 0.2367 - val_acc: 0.9000
Process time: 0:10:20.903161
```

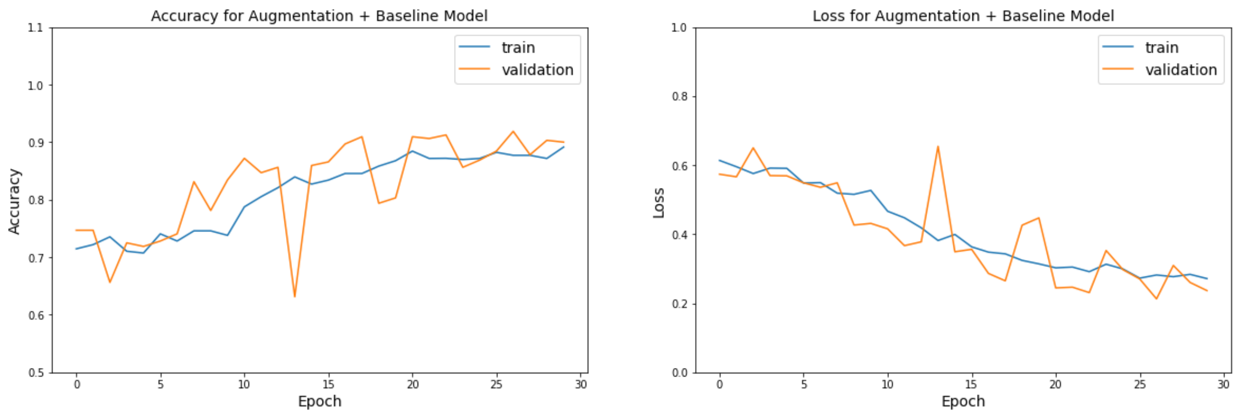
In [ ]:

```
model_7.save('saved_model_history/model_7.h5')
np.save('saved_model_history/history_7.npy', history_7.history)
```

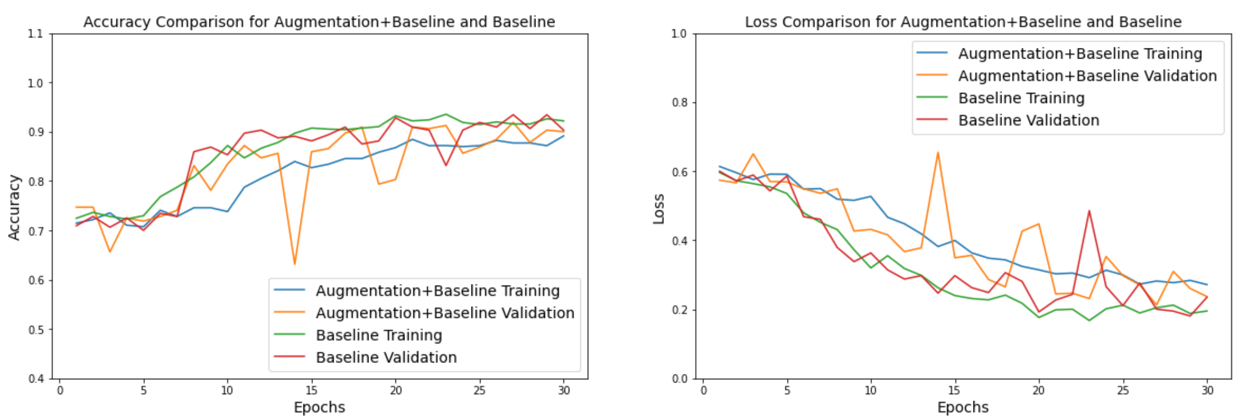
In [525]:
```
evaluate(model_7)
```

```
129/129 [==============================] - 57s 446ms/step - loss: 0.2375
- acc: 0.8973
28/28 [==============================] - 12s 442ms/step - loss: 0.2404 -
acc: 0.8987
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.8973
Train Loss = 0.2375
----------------------------
Test Accuracy = 0.8987
Test Loss = 0.2404
```

In [526]:
```
plot_acc_loss(history_7, 7, 'Augmentation + Baseline Model')
```



In [661]:
```
compare_models(history_7, history_4, "Augmentation+Baseline", 'Baseline', 3
```



Comments

- The accuracy and loss is worse than the baseline model
- Too much fluctuation in accuracy and Loss graphs for Validation.
- Process time is 10.5 mins.

Augmentation with various parameters

I plan to try different augmentation parameter for training model.

However, the run time is quite long to play around.

I decided to run the training with only 10 epochs.

According to the previous comparison graphs, the beginning performance (10 epochs) gives an idea about the performance of the model.

I will run the model and compare it with the beginning 10 epochs of the baseline model (model 4).

The following parameters are tested one by one and compared to baseline model:

- horizontal_flip=True => lower performance
- brightness_range=
  - [0.5, 2] => sligtly lower performace
  - [0.3, 3] => lower performance
  - [0.5, 1.5] => lower performance
- width_shift_range=0.2, height_shift_range=0.2 => lower performance
- rotation_range=
  - 360 => lower performance
  - 40 => lower performance
  - 90 => lower performance
- zoom_range=
  - 0.1 => lower performance
  - 0.2 => lower performance
- shear_range=
  - 0.2 => sligtly lower performance
  - 0.3 => similar
  - 0.4 => lower performance

Unfortunately, the augmentation is not helping. Most of the parameters caused a lower performance. Only shear_range=0.3 keep the performance of the baseline model.

Augmentation with shear_range=0.3

I am wondering if the augmentation with shear_range=0.3 will improve the model performance when running with larger epoch number.

In [623]:
```
# Model with Augmetation 2

train_datagen_aug2 = ImageDataGenerator(rescale=1./255,
                                   #rotation_range=90,
                                   #width_shift_range=0.2,
                                   #height_shift_range=0.2,
                                   shear_range=0.3,
                                   #zoom_range=0.2,
                                   #horizontal_flip=True,
                                    #brightness_range=[0.5, 1.5]
                                       )

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator_b20_aug2 = train_datagen_aug2.flow_from_directory(
        train_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')

val_generator_b20 = val_datagen.flow_from_directory(
        val_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')
```

```
Found 4098 images belonging to 2 classes.
Found 879 images belonging to 2 classes.
```

In [624]:
```
# Design Model

model_8 = models.Sequential()
model_8.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 6
model_8.add(layers.MaxPooling2D((2, 2)))

model_8.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_8.add(layers.MaxPooling2D((2, 2)))

model_8.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_8.add(layers.MaxPooling2D((2, 2)))

model_8.add(layers.Flatten())
model_8.add(layers.Dense(64, activation='relu'))
model_8.add(layers.Dense(1, activation='sigmoid'))

model_8.compile(loss='binary_crossentropy',
              optimizer= 'sgd',
              metrics=['acc'])
```

In [625]:
```python
start=datetime.now()

history_8 = model_8.fit(train_generator_b20_aug2,
                       steps_per_epoch=48, # 48x20 training samaples will run
                       epochs=30,
                       validation_data=val_generator_b20,
                       validation_steps=16,  # 16x20 validation samaples will
                      )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/30
48/48 [==============================] - 21s 438ms/step - loss: 0.6329 -
acc: 0.6823 - val_loss: 0.5988 - val_acc: 0.7219
Epoch 2/30
48/48 [==============================] - 20s 419ms/step - loss: 0.5757 -
acc: 0.7323 - val_loss: 0.5789 - val_acc: 0.7188
Epoch 3/30
48/48 [==============================] - 20s 426ms/step - loss: 0.5785 -
acc: 0.7167 - val_loss: 0.5646 - val_acc: 0.7219
Epoch 4/30
48/48 [==============================] - 20s 425ms/step - loss: 0.5536 -
acc: 0.7208 - val_loss: 0.5502 - val_acc: 0.7031
Epoch 5/30
48/48 [==============================] - 20s 425ms/step - loss: 0.5341 -
acc: 0.7302 - val_loss: 0.5194 - val_acc: 0.8156
Epoch 6/30
48/48 [==============================] - 20s 419ms/step - loss: 0.4992 -
acc: 0.7490 - val_loss: 0.4461 - val_acc: 0.8719
Epoch 7/30
48/48 [==============================] - 20s 426ms/step - loss: 0.4423 -
acc: 0.8048 - val_loss: 0.6577 - val_acc: 0.7094
Epoch 8/30
48/48 [==============================] - 23s 476ms/step - loss: 0.4463 -
acc: 0.8104 - val_loss: 0.3908 - val_acc: 0.8344
Epoch 9/30
48/48 [==============================] - 21s 428ms/step - loss: 0.4123 -
acc: 0.8125 - val_loss: 0.4280 - val_acc: 0.7688
Epoch 10/30
48/48 [==============================] - 21s 431ms/step - loss: 0.3554 -
acc: 0.8375 - val_loss: 0.3737 - val_acc: 0.8625
Epoch 11/30
48/48 [==============================] - 22s 452ms/step - loss: 0.3263 -
acc: 0.8698 - val_loss: 0.4158 - val_acc: 0.8156
Epoch 12/30
48/48 [==============================] - 21s 443ms/step - loss: 0.3079 -
acc: 0.8646 - val_loss: 0.3268 - val_acc: 0.8531
Epoch 13/30
48/48 [==============================] - 22s 453ms/step - loss: 0.2853 -
acc: 0.8800 - val_loss: 0.2526 - val_acc: 0.9062
Epoch 14/30
48/48 [==============================] - 21s 432ms/step - loss: 0.2951 -
acc: 0.8594 - val_loss: 0.2348 - val_acc: 0.9125
Epoch 15/30
```

```
48/48 [==============================] - 20s 423ms/step - loss: 0.2533 -
acc: 0.8977 - val_loss: 0.2974 - val_acc: 0.8656
Epoch 16/30
48/48 [==============================] - 20s 411ms/step - loss: 0.2463 -
acc: 0.8969 - val_loss: 0.2712 - val_acc: 0.8813
Epoch 17/30
48/48 [==============================] - 20s 414ms/step - loss: 0.2471 -
acc: 0.8904 - val_loss: 0.4533 - val_acc: 0.8344
Epoch 18/30
48/48 [==============================] - 20s 418ms/step - loss: 0.2152 -
acc: 0.9156 - val_loss: 0.2331 - val_acc: 0.9094
Epoch 19/30
48/48 [==============================] - 21s 437ms/step - loss: 0.2322 -
acc: 0.9083 - val_loss: 0.2143 - val_acc: 0.9156
Epoch 20/30
48/48 [==============================] - 19s 405ms/step - loss: 0.2141 -
acc: 0.9052 - val_loss: 0.2401 - val_acc: 0.9031
Epoch 21/30
48/48 [==============================] - 21s 426ms/step - loss: 0.2199 -
acc: 0.9094 - val_loss: 0.2700 - val_acc: 0.8906
Epoch 22/30
48/48 [==============================] - 20s 421ms/step - loss: 0.2300 -
acc: 0.9062 - val_loss: 0.2052 - val_acc: 0.9281
Epoch 23/30
48/48 [==============================] - 20s 417ms/step - loss: 0.2153 -
acc: 0.9050 - val_loss: 0.2114 - val_acc: 0.9219
Epoch 24/30
48/48 [==============================] - 20s 411ms/step - loss: 0.1889 -
acc: 0.9229 - val_loss: 0.2135 - val_acc: 0.9062
Epoch 25/30
48/48 [==============================] - 20s 412ms/step - loss: 0.2147 -
acc: 0.9208 - val_loss: 0.2672 - val_acc: 0.9062
Epoch 26/30
48/48 [==============================] - 19s 407ms/step - loss: 0.2231 -
acc: 0.9146 - val_loss: 0.2961 - val_acc: 0.8781
Epoch 27/30
48/48 [==============================] - 20s 427ms/step - loss: 0.2119 -
acc: 0.9217 - val_loss: 0.2013 - val_acc: 0.9312
Epoch 28/30
48/48 [==============================] - 24s 505ms/step - loss: 0.1842 -
acc: 0.9323 - val_loss: 0.2061 - val_acc: 0.9031
Epoch 29/30
48/48 [==============================] - 20s 417ms/step - loss: 0.2166 -
acc: 0.9146 - val_loss: 0.2171 - val_acc: 0.9250
Epoch 30/30
48/48 [==============================] - 20s 418ms/step - loss: 0.1908 -
acc: 0.9240 - val_loss: 0.1871 - val_acc: 0.9312
Process time: 0:10:38.555020
```

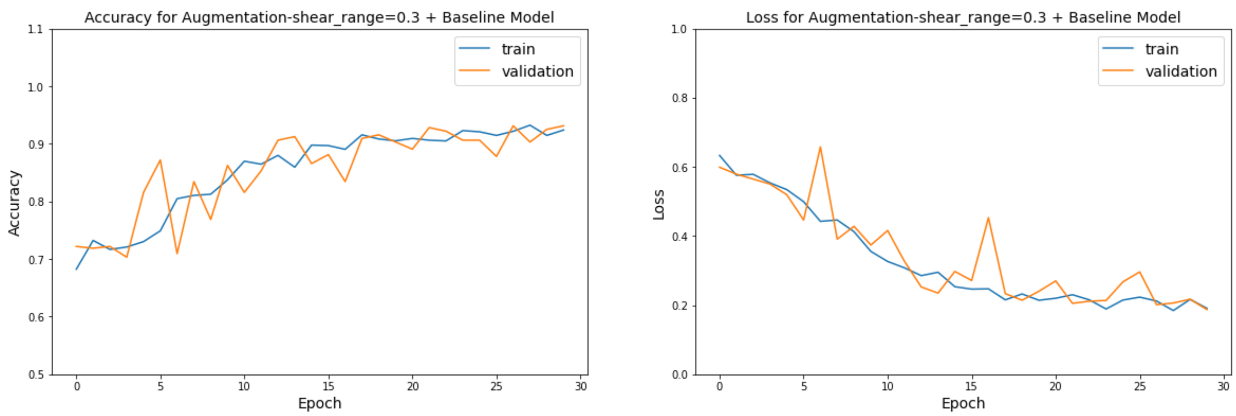
In [626]:

```
model_8.save('saved_model_history/model_8.h5')
np.save('saved_model_history/history_8.npy', history_8.history)
```

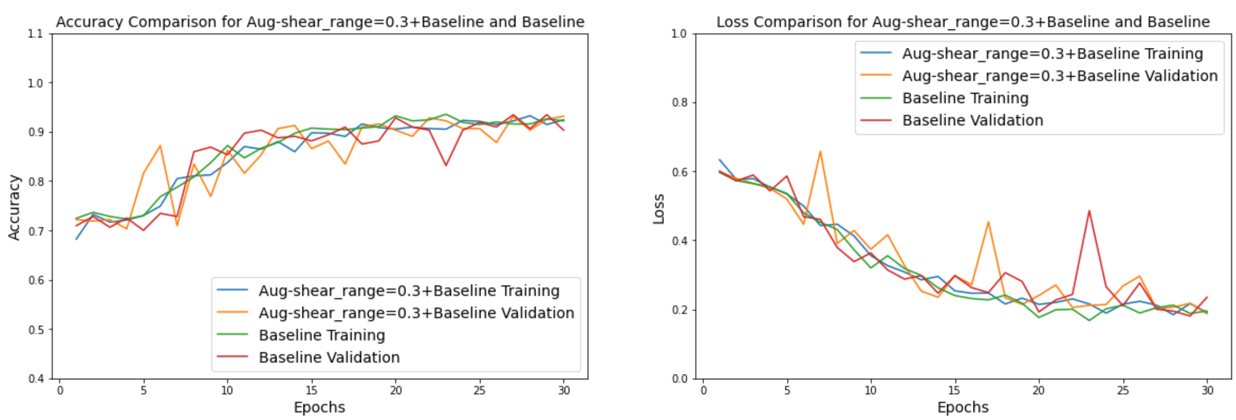
In [627]:
```
evaluate(model_8)
```

```
129/129 [==============================] - 56s 437ms/step - loss: 0.1698
- acc: 0.9356
28/28 [==============================] - 12s 424ms/step - loss: 0.1824 -
acc: 0.9295
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.9356
Train Loss = 0.1698
----------------------------
Test Accuracy = 0.9295
Test Loss = 0.1824
```

In [662]:
```
plot_acc_loss(history_8, 8, 'Augmentation-shear_range=0.3 + Baseline Model'
```



In [663]:
```
compare_models(history_8, history_4, "Aug-shear_range=0.3+Baseline", 'Basel
```



Comments

- Augmentation with shear_range=0.3 didn't have improve the model performance. It is almost same as the basline results.
- I do not plan to use augmentation in my model.
- Runtime: 10.5 minutes

Model with Optimizer = 'Adam'

I will use a different optimizer = 'Adam' when compiling the model.

In [719]:
```
# Load the data

train_datagen = ImageDataGenerator(rescale=1./255)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator_b20 = train_datagen.flow_from_directory(
        train_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')

val_generator_b20 = val_datagen.flow_from_directory(
        val_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')
```

```
Found 4098 images belonging to 2 classes.
Found 879 images belonging to 2 classes.
```

In [722]:
```
# Design Model
# Optiizer = 'adam'

model_9 = models.Sequential()
model_9.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 6
model_9.add(layers.MaxPooling2D((2, 2)))

model_9.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_9.add(layers.MaxPooling2D((2, 2)))

model_9.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_9.add(layers.MaxPooling2D((2, 2)))

model_9.add(layers.Flatten())
model_9.add(layers.Dense(64, activation='relu'))
model_9.add(layers.Dense(1, activation='sigmoid'))

model_9.compile(loss='binary_crossentropy',
              optimizer= 'adam',
              metrics=['acc'])
```

In [723]:
```
start=datetime.now()

history_9 = model_9.fit(train_generator_b20,
                     steps_per_epoch=48, # 48x20 training samaples will run
                     epochs=30,
                     validation_data=val_generator_b20,
                     validation_steps=16,  # 16x20 validation samaples will
                    )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/30
48/48 [==============================] - 22s 450ms/step - loss: 0.5840 -
acc: 0.7094 - val_loss: 0.4783 - val_acc: 0.7000
Epoch 2/30
48/48 [==============================] - 20s 409ms/step - loss: 0.3699 -
acc: 0.8302 - val_loss: 0.2939 - val_acc: 0.8969
Epoch 3/30
48/48 [==============================] - 19s 391ms/step - loss: 0.2655 -
acc: 0.8844 - val_loss: 0.2108 - val_acc: 0.9187
Epoch 4/30
48/48 [==============================] - 19s 397ms/step - loss: 0.1964 -
acc: 0.9240 - val_loss: 0.4717 - val_acc: 0.8594
Epoch 5/30
48/48 [==============================] - 19s 395ms/step - loss: 0.2258 -
acc: 0.9029 - val_loss: 0.2389 - val_acc: 0.9094
Epoch 6/30
48/48 [==============================] - 19s 406ms/step - loss: 0.2071 -
acc: 0.9198 - val_loss: 0.2212 - val_acc: 0.9156
Epoch 7/30
48/48 [==============================] - 20s 408ms/step - loss: 0.1955 -
acc: 0.9156 - val_loss: 0.1768 - val_acc: 0.9375
Epoch 8/30
48/48 [==============================] - 19s 402ms/step - loss: 0.1982 -
acc: 0.9198 - val_loss: 0.2339 - val_acc: 0.9094
Epoch 9/30
48/48 [==============================] - 19s 399ms/step - loss: 0.2034 -
acc: 0.9207 - val_loss: 0.2194 - val_acc: 0.9062
Epoch 10/30
48/48 [==============================] - 19s 401ms/step - loss: 0.1892 -
acc: 0.9333 - val_loss: 0.2893 - val_acc: 0.9125
Epoch 11/30
48/48 [==============================] - 19s 393ms/step - loss: 0.1549 -
acc: 0.9365 - val_loss: 0.2123 - val_acc: 0.9094
Epoch 12/30
48/48 [==============================] - 20s 411ms/step - loss: 0.1540 -
acc: 0.9374 - val_loss: 0.1746 - val_acc: 0.9469
Epoch 13/30
48/48 [==============================] - 20s 421ms/step - loss: 0.1757 -
acc: 0.9312 - val_loss: 0.1838 - val_acc: 0.9406
Epoch 14/30
48/48 [==============================] - 19s 407ms/step - loss: 0.1560 -
acc: 0.9469 - val_loss: 0.1998 - val_acc: 0.9375
Epoch 15/30
```

```
48/48 [==============================] - 19s 400ms/step - loss: 0.1157 -
acc: 0.9563 - val_loss: 0.2832 - val_acc: 0.9187
Epoch 16/30
48/48 [==============================] - 23s 481ms/step - loss: 0.1474 -
acc: 0.9479 - val_loss: 0.2157 - val_acc: 0.9062
Epoch 17/30
48/48 [==============================] - 26s 545ms/step - loss: 0.1060 -
acc: 0.9625 - val_loss: 0.3015 - val_acc: 0.9125
Epoch 18/30
48/48 [==============================] - 19s 390ms/step - loss: 0.1409 -
acc: 0.9365 - val_loss: 0.2063 - val_acc: 0.9187
Epoch 19/30
48/48 [==============================] - 19s 402ms/step - loss: 0.1361 -
acc: 0.9500 - val_loss: 0.1667 - val_acc: 0.9219
Epoch 20/30
48/48 [==============================] - 21s 435ms/step - loss: 0.1273 -
acc: 0.9562 - val_loss: 0.1420 - val_acc: 0.9531
Epoch 21/30
48/48 [==============================] - 19s 387ms/step - loss: 0.1200 -
acc: 0.9573 - val_loss: 0.2196 - val_acc: 0.9375
Epoch 22/30
48/48 [==============================] - 19s 396ms/step - loss: 0.1338 -
acc: 0.9531 - val_loss: 0.1502 - val_acc: 0.9281
Epoch 23/30
48/48 [==============================] - 19s 401ms/step - loss: 0.1055 -
acc: 0.9677 - val_loss: 0.1315 - val_acc: 0.9469
Epoch 24/30
48/48 [==============================] - 19s 395ms/step - loss: 0.1118 -
acc: 0.9572 - val_loss: 0.2792 - val_acc: 0.9094
Epoch 25/30
48/48 [==============================] - 20s 415ms/step - loss: 0.1368 -
acc: 0.9521 - val_loss: 0.1569 - val_acc: 0.9469
Epoch 26/30
48/48 [==============================] - 19s 397ms/step - loss: 0.1012 -
acc: 0.9677 - val_loss: 0.1850 - val_acc: 0.9281
Epoch 27/30
48/48 [==============================] - 18s 378ms/step - loss: 0.0952 -
acc: 0.9656 - val_loss: 0.1327 - val_acc: 0.9531
Epoch 28/30
48/48 [==============================] - 19s 403ms/step - loss: 0.0878 -
acc: 0.9656 - val_loss: 0.1276 - val_acc: 0.9469
Epoch 29/30
48/48 [==============================] - 19s 393ms/step - loss: 0.0802 -
acc: 0.9667 - val_loss: 0.2036 - val_acc: 0.9156
Epoch 30/30
48/48 [==============================] - 20s 413ms/step - loss: 0.0897 -
acc: 0.9656 - val_loss: 0.1864 - val_acc: 0.9344
Process time: 0:09:54.230131
```

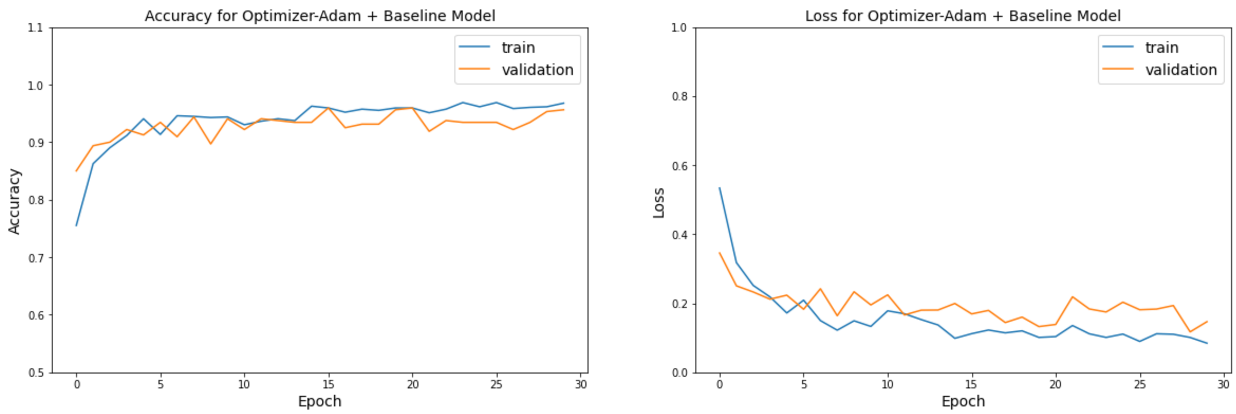
In [724]:

```
model_9.save('saved_model_history/model_9.h5')
np.save('saved_model_history/history_9.npy', history_9.history)
```

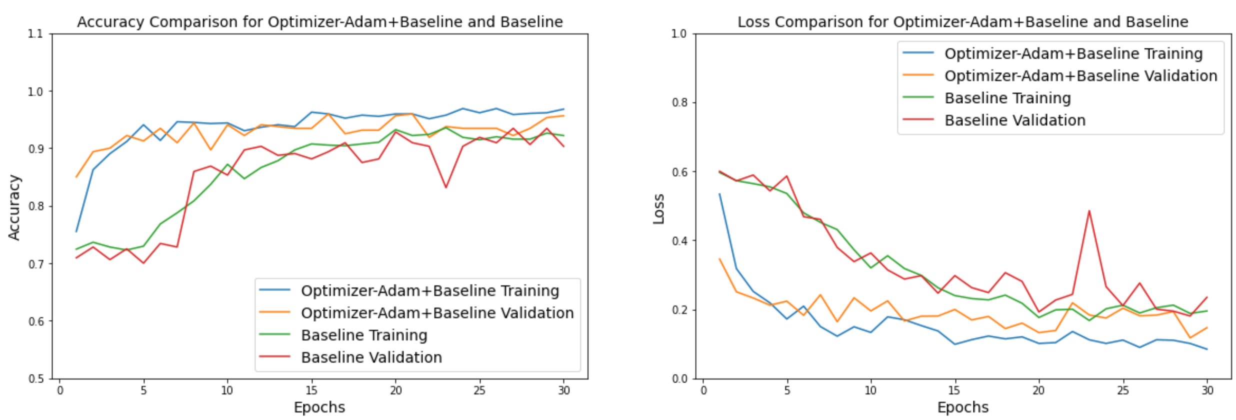
In [725]:
```
evaluate(model_9)
```

```
129/129 [==============================] - 54s 420ms/step - loss: 0.0774
- acc: 0.9736
28/28 [==============================] - 12s 420ms/step - loss: 0.1214 -
acc: 0.9545
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.9736
Train Loss = 0.0774
----------------------------
Test Accuracy = 0.9545
Test Loss = 0.1214
```

In [639]:
```
plot_acc_loss(history_9, 9, 'Optimizer-Adam + Baseline Model')
```



In [664]:
```
compare_models(history_9, history_4, "Optimizer-Adam+Baseline", 'Baseline',
```



Comments

- The optimizer 'Adam' significantly improved the model performance compared to optimizer 'sgd'.
- The accuracy increased and loss decreased in training, validation and testing samples.
- There is slight overfitting.
- I will use optimizer 'adam' for my final model.
- Runtime: 10 mins

Model with Optimizer = 'Adam' with Regularization

In [643]:
```
#Design model and compile
# Optimizer = 'adam'

model_10 = models.Sequential()
model_10.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_regularize
                           input_shape=(64, 64, 3)))
model_10.add(layers.MaxPooling2D((2, 2)))

model_10.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_regularize
model_10.add(layers.MaxPooling2D((2, 2)))

model_10.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_regularize
model_10.add(layers.MaxPooling2D((2, 2)))

model_10.add(layers.Flatten())
model_10.add(layers.Dense(64, activation='relu', kernel_regularizer=regular
model_10.add(layers.Dense(1, activation='sigmoid'))

model_10.compile(loss='binary_crossentropy',
              optimizer= 'adam',
              metrics=['acc'])
```

In [644]:
```python
start=datetime.now()

history_10 = model_10.fit(train_generator_b20,
                          steps_per_epoch=48, # 48x20 training samaples will run
                          epochs=30,
                          validation_data=val_generator_b20,
                          validation_steps=16,  # 16x20 validation samaples will
                         )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/30
48/48 [==============================] - 21s 424ms/step - loss: 1.1207 -
acc: 0.7188 - val_loss: 0.7917 - val_acc: 0.7437
Epoch 2/30
48/48 [==============================] - 19s 398ms/step - loss: 0.6219 -
acc: 0.7927 - val_loss: 0.4979 - val_acc: 0.8750
Epoch 3/30
48/48 [==============================] - 20s 411ms/step - loss: 0.4832 -
acc: 0.8542 - val_loss: 0.4472 - val_acc: 0.8906
Epoch 4/30
48/48 [==============================] - 23s 490ms/step - loss: 0.3833 -
acc: 0.8967 - val_loss: 0.3892 - val_acc: 0.9125
Epoch 5/30
48/48 [==============================] - 19s 405ms/step - loss: 0.3461 -
acc: 0.9135 - val_loss: 0.4030 - val_acc: 0.8719
Epoch 6/30
48/48 [==============================] - 21s 430ms/step - loss: 0.3411 -
acc: 0.9031 - val_loss: 0.3569 - val_acc: 0.9031
Epoch 7/30
48/48 [==============================] - 20s 409ms/step - loss: 0.3101 -
acc: 0.9156 - val_loss: 0.3767 - val_acc: 0.9031
Epoch 8/30
48/48 [==============================] - 23s 472ms/step - loss: 0.3514 -
acc: 0.8987 - val_loss: 0.4110 - val_acc: 0.8562
Epoch 9/30
48/48 [==============================] - 21s 430ms/step - loss: 0.3200 -
acc: 0.9167 - val_loss: 0.3356 - val_acc: 0.9062
Epoch 10/30
48/48 [==============================] - 21s 443ms/step - loss: 0.2943 -
acc: 0.9219 - val_loss: 0.3963 - val_acc: 0.8938
Epoch 11/30
48/48 [==============================] - 19s 405ms/step - loss: 0.2616 -
acc: 0.9344 - val_loss: 0.3398 - val_acc: 0.9219
Epoch 12/30
48/48 [==============================] - 22s 450ms/step - loss: 0.2791 -
acc: 0.9302 - val_loss: 0.2962 - val_acc: 0.9219
Epoch 13/30
48/48 [==============================] - 20s 427ms/step - loss: 0.2533 -
acc: 0.9448 - val_loss: 0.2689 - val_acc: 0.9250
Epoch 14/30
48/48 [==============================] - 19s 400ms/step - loss: 0.2674 -
acc: 0.9395 - val_loss: 0.3158 - val_acc: 0.9125
Epoch 15/30
48/48 [==============================] - 19s 388ms/step - loss: 0.2573 -
```

```
acc: 0.9396 - val_loss: 0.2952 - val_acc: 0.9219
Epoch 16/30
48/48 [==============================] - 19s 399ms/step - loss: 0.2411 -
acc: 0.9354 - val_loss: 0.2404 - val_acc: 0.9469
Epoch 17/30
48/48 [==============================] - 21s 429ms/step - loss: 0.2319 -
acc: 0.9342 - val_loss: 0.2738 - val_acc: 0.9219
Epoch 18/30
48/48 [==============================] - 20s 411ms/step - loss: 0.2345 -
acc: 0.9448 - val_loss: 0.2325 - val_acc: 0.9500
Epoch 19/30
48/48 [==============================] - 20s 408ms/step - loss: 0.2424 -
acc: 0.9333 - val_loss: 0.3149 - val_acc: 0.8781
Epoch 20/30
48/48 [==============================] - 19s 393ms/step - loss: 0.2608 -
acc: 0.9375 - val_loss: 0.4747 - val_acc: 0.8375
Epoch 21/30
48/48 [==============================] - 18s 382ms/step - loss: 0.2812 -
acc: 0.9198 - val_loss: 0.3256 - val_acc: 0.8875
Epoch 22/30
48/48 [==============================] - 19s 389ms/step - loss: 0.2733 -
acc: 0.9208 - val_loss: 0.2686 - val_acc: 0.9438
Epoch 23/30
48/48 [==============================] - 18s 376ms/step - loss: 0.2312 -
acc: 0.9396 - val_loss: 0.2888 - val_acc: 0.9281
Epoch 24/30
48/48 [==============================] - 19s 394ms/step - loss: 0.2106 -
acc: 0.9427 - val_loss: 0.2740 - val_acc: 0.9187
Epoch 25/30
48/48 [==============================] - 19s 394ms/step - loss: 0.2134 -
acc: 0.9458 - val_loss: 0.2661 - val_acc: 0.9250
Epoch 26/30
48/48 [==============================] - 19s 390ms/step - loss: 0.2330 -
acc: 0.9415 - val_loss: 0.2715 - val_acc: 0.9219
Epoch 27/30
48/48 [==============================] - 18s 383ms/step - loss: 0.2409 -
acc: 0.9344 - val_loss: 0.2713 - val_acc: 0.9219
Epoch 28/30
48/48 [==============================] - 19s 391ms/step - loss: 0.2241 -
acc: 0.9384 - val_loss: 0.2861 - val_acc: 0.9219
Epoch 29/30
48/48 [==============================] - 19s 407ms/step - loss: 0.2285 -
acc: 0.9406 - val_loss: 0.3354 - val_acc: 0.9031
Epoch 30/30
48/48 [==============================] - 21s 445ms/step - loss: 0.2033 -
acc: 0.9406 - val_loss: 0.3450 - val_acc: 0.9031
Process time: 0:09:56.749394
```

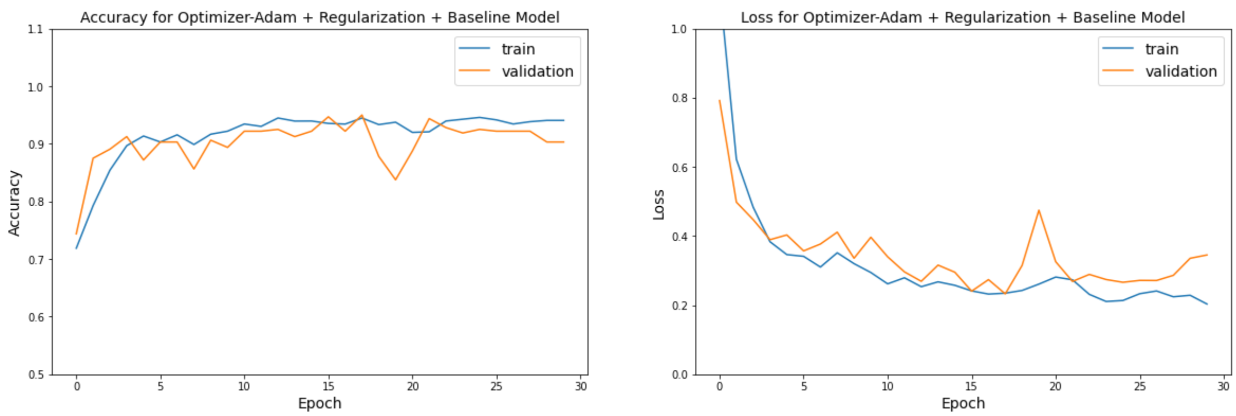
In [645]:

```
model_10.save('saved_model_history/model_10.h5')
np.save('saved_model_history/history_10.npy', history_10.history)
```

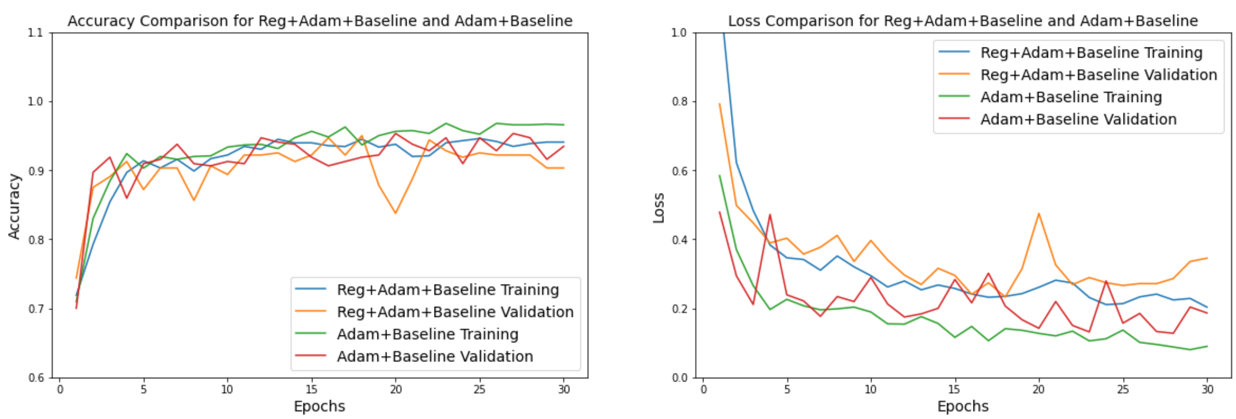
```
In [646]: evaluate(model_10)
```

```
129/129 [==============================] - 58s 453ms/step - loss: 0.2380
- acc: 0.9285
28/28 [==============================] - 12s 433ms/step - loss: 0.2423 -
acc: 0.9272
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.9285
Train Loss = 0.238
----------------------------
Test Accuracy = 0.9272
Test Loss = 0.2423
```

```
In [647]: plot_acc_loss(history_10, 10, 'Regularization + Optimizer-Adam + Baseline M
```




```
In [728]: compare_models(history_10, history_9, "Reg+Adam+Baseline", 'Adam+Baseline',
```



Comments

- Regularization decreased the ovefitting, but also decreased the model performance.
- The accuracy and loss in testing and training are worse than without regularization.
- I will not use Regularization in my final model.

Model with more layers and optimizer='adam'

I will add 2 more CNN layers and 1 more Dense layer to my baseline model.

I will continue to use optimizer='adam'

In [670]:
```
# Design Model
# Optiizer = 'adam'
# Add 2 more CNN and 1 more Dense layers.

model_11 = models.Sequential()
model_11.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64,
model_11.add(layers.MaxPooling2D((2, 2)))

model_11.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_11.add(layers.MaxPooling2D((2, 2)))

model_11.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_11.add(layers.MaxPooling2D((2, 2)))

model_11.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_11.add(layers.MaxPooling2D((2, 2)))

model_11.add(layers.Flatten())
model_11.add(layers.Dense(128, activation='relu'))
model_11.add(layers.Dense(128, activation='relu'))
model_11.add(layers.Dense(1, activation='sigmoid'))

model_11.compile(loss='binary_crossentropy',
              optimizer= 'adam',
              metrics=['acc'])
```

In [671]:
```python
start=datetime.now()

history_11 = model_11.fit(train_generator_b20,
                          steps_per_epoch=48, # 48x20 training samaples will run
                          epochs=30,
                          validation_data=val_generator_b20,
                          validation_steps=16,  # 16x20 validation samaples will
                         )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/30
48/48 [==============================] - 21s 424ms/step - loss: 0.5998 -
acc: 0.7250 - val_loss: 0.5491 - val_acc: 0.7500
Epoch 2/30
48/48 [==============================] - 19s 388ms/step - loss: 0.5405 -
acc: 0.7437 - val_loss: 0.3683 - val_acc: 0.8531
Epoch 3/30
48/48 [==============================] - 19s 394ms/step - loss: 0.3873 -
acc: 0.8125 - val_loss: 0.2597 - val_acc: 0.9094
Epoch 4/30
48/48 [==============================] - 19s 396ms/step - loss: 0.2387 -
acc: 0.9010 - val_loss: 0.2091 - val_acc: 0.9156
Epoch 5/30
48/48 [==============================] - 18s 386ms/step - loss: 0.2383 -
acc: 0.8956 - val_loss: 0.2832 - val_acc: 0.9000
Epoch 6/30
48/48 [==============================] - 19s 385ms/step - loss: 0.2093 -
acc: 0.9146 - val_loss: 0.2658 - val_acc: 0.9062
Epoch 7/30
48/48 [==============================] - 19s 404ms/step - loss: 0.1392 -
acc: 0.9458 - val_loss: 0.2335 - val_acc: 0.9125
Epoch 8/30
48/48 [==============================] - 19s 395ms/step - loss: 0.1687 -
acc: 0.9312 - val_loss: 0.1799 - val_acc: 0.9187
Epoch 9/30
48/48 [==============================] - 19s 395ms/step - loss: 0.1538 -
acc: 0.9438 - val_loss: 0.1595 - val_acc: 0.9344
Epoch 10/30
48/48 [==============================] - 19s 391ms/step - loss: 0.1582 -
acc: 0.9427 - val_loss: 0.2121 - val_acc: 0.9156
Epoch 11/30
48/48 [==============================] - 19s 390ms/step - loss: 0.1531 -
acc: 0.9500 - val_loss: 0.2658 - val_acc: 0.9094
Epoch 12/30
48/48 [==============================] - 19s 403ms/step - loss: 0.1654 -
acc: 0.9426 - val_loss: 0.1871 - val_acc: 0.9219
Epoch 13/30
48/48 [==============================] - 24s 495ms/step - loss: 0.1566 -
acc: 0.9365 - val_loss: 0.1576 - val_acc: 0.9469
Epoch 14/30
48/48 [==============================] - 20s 418ms/step - loss: 0.1392 -
acc: 0.9468 - val_loss: 0.1263 - val_acc: 0.9594
Epoch 15/30
48/48 [==============================] - 18s 380ms/step - loss: 0.1535 -
```

```
acc: 0.9427 - val_loss: 0.2351 - val_acc: 0.9156
Epoch 16/30
48/48 [==============================] - 19s 389ms/step - loss: 0.1492 -
acc: 0.9417 - val_loss: 0.1969 - val_acc: 0.9187
Epoch 17/30
48/48 [==============================] - 19s 390ms/step - loss: 0.1201 -
acc: 0.9594 - val_loss: 0.1055 - val_acc: 0.9594
Epoch 18/30
48/48 [==============================] - 20s 429ms/step - loss: 0.1404 -
acc: 0.9457 - val_loss: 0.1708 - val_acc: 0.9375
Epoch 19/30
48/48 [==============================] - 21s 435ms/step - loss: 0.1222 -
acc: 0.9479 - val_loss: 0.2267 - val_acc: 0.9312
Epoch 20/30
48/48 [==============================] - 19s 387ms/step - loss: 0.0945 -
acc: 0.9666 - val_loss: 0.2242 - val_acc: 0.9125
Epoch 21/30
48/48 [==============================] - 18s 383ms/step - loss: 0.1008 -
acc: 0.9656 - val_loss: 0.2049 - val_acc: 0.9406
Epoch 22/30
48/48 [==============================] - 19s 404ms/step - loss: 0.0959 -
acc: 0.9656 - val_loss: 0.1173 - val_acc: 0.9594
Epoch 23/30
48/48 [==============================] - 21s 438ms/step - loss: 0.0975 -
acc: 0.9604 - val_loss: 0.1095 - val_acc: 0.9563
Epoch 24/30
48/48 [==============================] - 22s 459ms/step - loss: 0.0919 -
acc: 0.9646 - val_loss: 0.1514 - val_acc: 0.9375
Epoch 25/30
48/48 [==============================] - 20s 418ms/step - loss: 0.1173 -
acc: 0.9562 - val_loss: 0.1908 - val_acc: 0.9312
Epoch 26/30
48/48 [==============================] - 21s 438ms/step - loss: 0.1113 -
acc: 0.9604 - val_loss: 0.1508 - val_acc: 0.9531
Epoch 27/30
48/48 [==============================] - 19s 400ms/step - loss: 0.0857 -
acc: 0.9677 - val_loss: 0.1455 - val_acc: 0.9438
Epoch 28/30
48/48 [==============================] - 20s 411ms/step - loss: 0.0926 -
acc: 0.9697 - val_loss: 0.1758 - val_acc: 0.9469
Epoch 29/30
48/48 [==============================] - 19s 399ms/step - loss: 0.0873 -
acc: 0.9698 - val_loss: 0.1872 - val_acc: 0.9406
Epoch 30/30
48/48 [==============================] - 19s 393ms/step - loss: 0.0749 -
acc: 0.9749 - val_loss: 0.2103 - val_acc: 0.9375
Process time: 0:09:47.262779
```

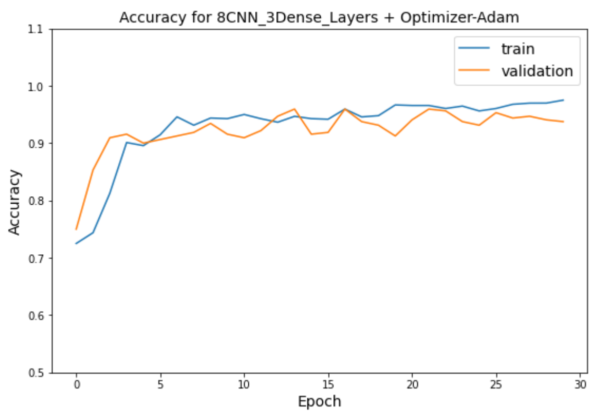
In [672]:

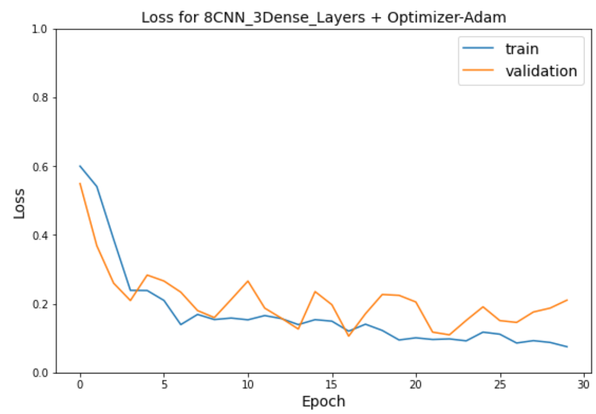
```
model_11.save('saved_model_history/model_11.h5')
np.save('saved_model_history/history_11.npy', history_11.history)
```

In [675]:
```
evaluate(model_11)
```

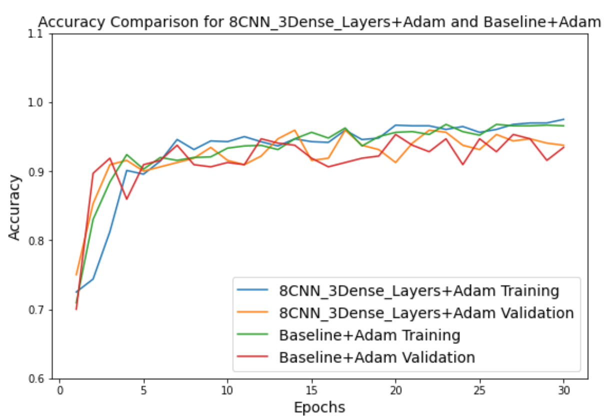
```
129/129 [==============================] - 56s 430ms/step - loss: 0.0806
- acc: 0.9678
28/28 [==============================] - 12s 437ms/step - loss: 0.1536 -
acc: 0.9522
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.9678
Train Loss = 0.0806
----------------------------
Test Accuracy = 0.9522
Test Loss = 0.1536
```

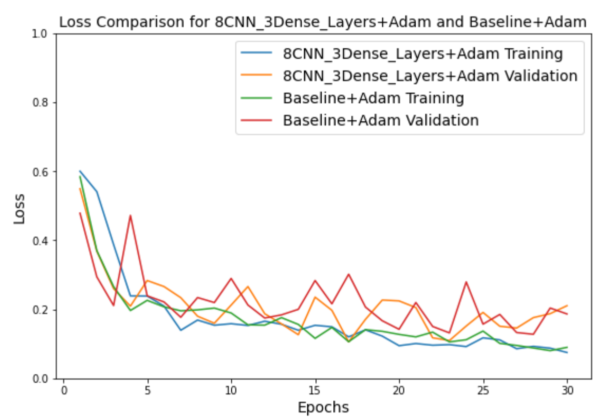
In [673]:
```
plot_acc_loss(history_11, 11, '8CNN_3Dense_Layers + Optimizer-Adam')
```





In [727]:
```
compare_models(history_11, history_9, "8CNN_3Dense_Layers+Adam", 'Baseline+
```





Comments

- The new model with 8 CNN and 3 Dense layers didn't perform better than our Baseline Model.
- The testing accuracy is same, but testing loss is a bit larger.
- There is no point of using this model.
- Runtime = 10 mins

Train on whole dataset, epoch=30

I will use the baseline model with optimizer='adam'.

This is model 9, but now I will run it on whole training and validation.

I will keep epoch=30. Even though increasing the epoch number will improve the model performance, I will use epoch 30 considering my limited computing power. The accuracy and loss curve has very small slope after epoch 10. Therefore, stopping at epoch 30 should be fine.

In [685]:

```
# Load the images
# batch_size = 20 for trainig and validation

train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator_b20 = train_datagen.flow_from_directory(
        train_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')

val_generator_b20 = val_datagen.flow_from_directory(
        val_dir,
        target_size=(64, 64),
        batch_size=20,
        class_mode='binary')

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')
```

```
Found 4098 images belonging to 2 classes.
Found 879 images belonging to 2 classes.
Found 879 images belonging to 2 classes.
```

In [690]:

```
# Design Model
# Optiizer = 'adam'

model_12 = models.Sequential()
model_12.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64,
model_12.add(layers.MaxPooling2D((2, 2)))

model_12.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_12.add(layers.MaxPooling2D((2, 2)))

model_12.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_12.add(layers.MaxPooling2D((2, 2)))

model_12.add(layers.Flatten())
model_12.add(layers.Dense(64, activation='relu'))
model_12.add(layers.Dense(1, activation='sigmoid'))

model_12.compile(loss='binary_crossentropy',
              optimizer= 'adam',
              metrics=['acc'])
```

In [691]:

```
start=datetime.now()

history_12 = model_12.fit(train_generator_b20,
                          steps_per_epoch=204, #max 204 (4098/20)
                          epochs=30,
                          validation_data=val_generator_b20,
                          validation_steps=43  #max 43 (879/20)
                         )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 1/30
204/204 [==============================] - 76s 371ms/step - loss: 0.3335
- acc: 0.8534 - val_loss: 0.2559 - val_acc: 0.8849
Epoch 2/30
204/204 [==============================] - 72s 354ms/step - loss: 0.1876
- acc: 0.9262 - val_loss: 0.2496 - val_acc: 0.9128
Epoch 3/30
204/204 [==============================] - 73s 358ms/step - loss: 0.1675
- acc: 0.9372 - val_loss: 0.2302 - val_acc: 0.9233
Epoch 4/30
204/204 [==============================] - 72s 354ms/step - loss: 0.1469
- acc: 0.9468 - val_loss: 0.1636 - val_acc: 0.9372
Epoch 5/30
204/204 [==============================] - 73s 355ms/step - loss: 0.1307
- acc: 0.9519 - val_loss: 0.1595 - val_acc: 0.9384
Epoch 6/30
204/204 [==============================] - 73s 356ms/step - loss: 0.1220
- acc: 0.9544 - val_loss: 0.1848 - val_acc: 0.9419
Epoch 7/30
204/204 [==============================] - 75s 365ms/step - loss: 0.1072
- acc: 0.9610 - val_loss: 0.1444 - val_acc: 0.9442
Epoch 8/30
204/204 [==============================] - 77s 376ms/step - loss: 0.1008
- acc: 0.9617 - val_loss: 0.1559 - val_acc: 0.9488
Epoch 9/30
204/204 [==============================] - 72s 354ms/step - loss: 0.0988
- acc: 0.9637 - val_loss: 0.1889 - val_acc: 0.9465
Epoch 10/30
204/204 [==============================] - 74s 364ms/step - loss: 0.0835
- acc: 0.9725 - val_loss: 0.1720 - val_acc: 0.9430
Epoch 11/30
204/204 [==============================] - 73s 356ms/step - loss: 0.0817
- acc: 0.9698 - val_loss: 0.1931 - val_acc: 0.9442
Epoch 12/30
204/204 [==============================] - 76s 373ms/step - loss: 0.0704
- acc: 0.9755 - val_loss: 0.2152 - val_acc: 0.9465
Epoch 13/30
204/204 [==============================] - 72s 355ms/step - loss: 0.0678
- acc: 0.9738 - val_loss: 0.1467 - val_acc: 0.9570
Epoch 14/30
204/204 [==============================] - 72s 352ms/step - loss: 0.0507
- acc: 0.9828 - val_loss: 0.2005 - val_acc: 0.9500
Epoch 15/30
204/204 [==============================] - 72s 353ms/step - loss: 0.0470
```

```
- acc: 0.9850 - val_loss: 0.1889 - val_acc: 0.9535
Epoch 16/30
204/204 [==============================] - 73s 357ms/step - loss: 0.0373
- acc: 0.9858 - val_loss: 0.2340 - val_acc: 0.9407
Epoch 17/30
204/204 [==============================] - 72s 353ms/step - loss: 0.0427
- acc: 0.9855 - val_loss: 0.2137 - val_acc: 0.9535
Epoch 18/30
204/204 [==============================] - 73s 358ms/step - loss: 0.0296
- acc: 0.9892 - val_loss: 0.2162 - val_acc: 0.9547
Epoch 19/30
204/204 [==============================] - 73s 356ms/step - loss: 0.0273
- acc: 0.9904 - val_loss: 0.2216 - val_acc: 0.9500
Epoch 20/30
204/204 [==============================] - 81s 400ms/step - loss: 0.0183
- acc: 0.9936 - val_loss: 0.2238 - val_acc: 0.9547
Epoch 21/30
204/204 [==============================] - 81s 398ms/step - loss: 0.0174
- acc: 0.9946 - val_loss: 0.2857 - val_acc: 0.9547
Epoch 22/30
204/204 [==============================] - 72s 354ms/step - loss: 0.0159
- acc: 0.9951 - val_loss: 0.2717 - val_acc: 0.9512
Epoch 23/30
204/204 [==============================] - 72s 355ms/step - loss: 0.0212
- acc: 0.9924 - val_loss: 0.2677 - val_acc: 0.9558
Epoch 24/30
204/204 [==============================] - 73s 359ms/step - loss: 0.0200
- acc: 0.9922 - val_loss: 0.2681 - val_acc: 0.9605
Epoch 25/30
204/204 [==============================] - 74s 363ms/step - loss: 0.0132
- acc: 0.9953 - val_loss: 0.3149 - val_acc: 0.9581
Epoch 26/30
204/204 [==============================] - 75s 366ms/step - loss: 0.0086
- acc: 0.9973 - val_loss: 0.3107 - val_acc: 0.9453
Epoch 27/30
204/204 [==============================] - 76s 373ms/step - loss: 0.0128
- acc: 0.9961 - val_loss: 0.3496 - val_acc: 0.9395
Epoch 28/30
204/204 [==============================] - 78s 385ms/step - loss: 0.0226
- acc: 0.9919 - val_loss: 0.2900 - val_acc: 0.9558
Epoch 29/30
204/204 [==============================] - 74s 361ms/step - loss: 0.0051
- acc: 0.9990 - val_loss: 0.2665 - val_acc: 0.9523
Epoch 30/30
204/204 [==============================] - 81s 398ms/step - loss: 0.0025
- acc: 0.9998 - val_loss: 0.3361 - val_acc: 0.9547
Process time: 0:37:20.504208
```

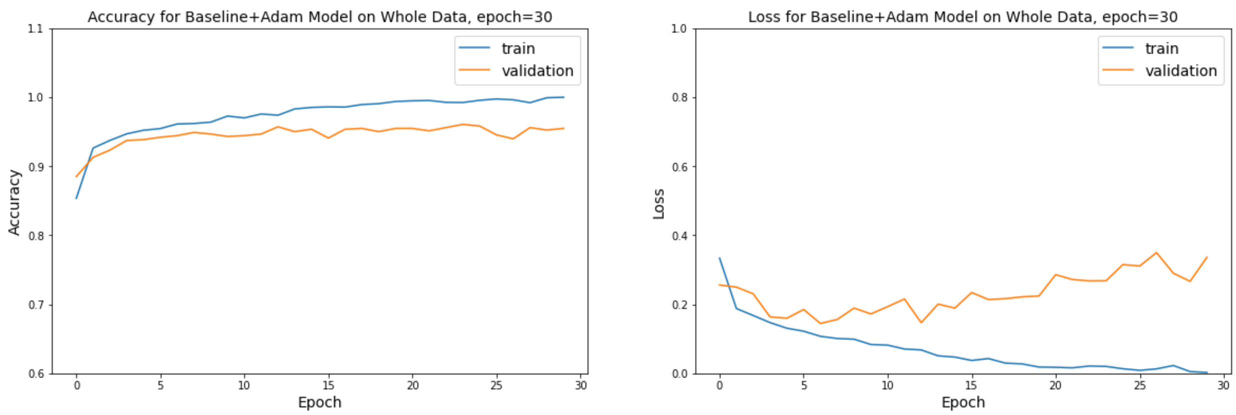
In [693]:

```
model_12.save('saved_model_history/model_12.h5')
np.save('saved_model_history/history_12.npy', history_12.history)
```

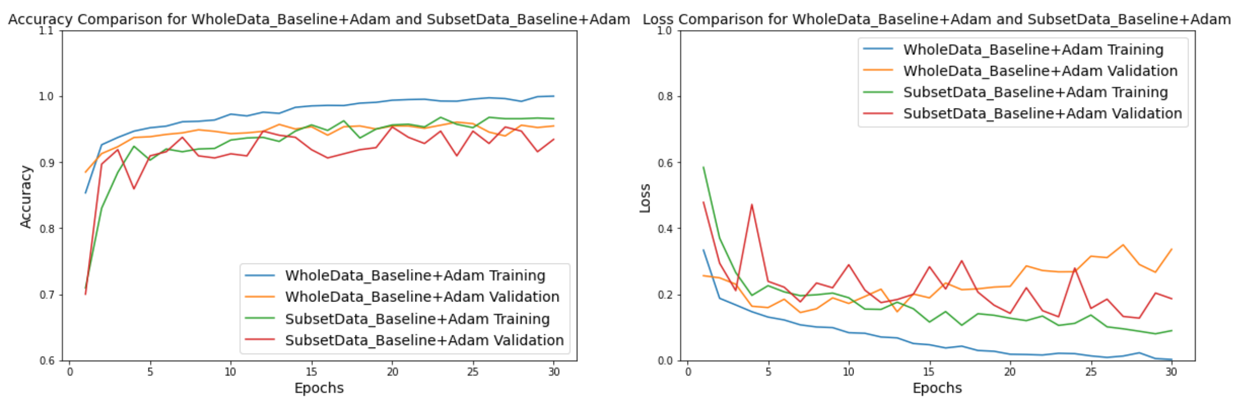
In [694]:
```
evaluate(model_12)
```

```
129/129 [==============================] - 55s 425ms/step - loss: 6.0769e
-04 - acc: 1.0000
28/28 [==============================] - 12s 423ms/step - loss: 0.3068 -
acc: 0.9590
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 1.0
Train Loss = 0.0006
----------------------------
Test Accuracy = 0.959
Test Loss = 0.3068
```

In [708]:
```
plot_acc_loss(history_12, 12, 'Baseline+Adam Model on Whole Data, epoch=30'
```



In [726]:
```
compare_models(history_12, history_9, "WholeData_Baseline+Adam", 'SubsetDat
```



Comments

- Running on whole data did improve the model performance very little.
- The testing accuracy (0.9590) is very slightly higher than the accuracy from the subset data (0.9545).
- However, testing loss significantly increasing on whole data (0.1214 -> 0.3068).
- Large dataset increased overfitting. The training accuracy is 1.00. The overfitting is especially observed in loss curve.

- With the larger sample size, the fluctuations in the accuracy and loss curves decrease, especially on the validation data.
- Runtime: 37 mins

With L2 Regularization

I have run this model with L2 regularization on whole data.

The results:

- Train Accuracy = 0.9492 & Train Loss = 0.1811
- Test Accuracy = 0.9352 & Test Loss = 0.2043

Train on subset of dataset, epoch=50

Beacuse I wonder, I will run the baseline+adam model on subset of dataset (~20% of whole data) one more time with epoch=50.

According to my analysis on baseline model, larger epoch size should improve model.

In [701]:
```
# Design Model
# Optiizer = 'adam'

model_13 = models.Sequential()
model_13.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64,
model_13.add(layers.MaxPooling2D((2, 2)))

model_13.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_13.add(layers.MaxPooling2D((2, 2)))

model_13.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_13.add(layers.MaxPooling2D((2, 2)))

model_13.add(layers.Flatten())
model_13.add(layers.Dense(64, activation='relu'))
model_13.add(layers.Dense(1, activation='sigmoid'))

model_13.compile(loss='binary_crossentropy',
              optimizer= 'adam',
              metrics=['acc'])
```

In [702]:

```python
start=datetime.now()

history_13 = model_13.fit(train_generator_b20,
                          steps_per_epoch=48, # 48x20 training samaples will run
                          epochs=50,
                          validation_data=val_generator_b20,
                          validation_steps=16,  # 16x20 validation samaples will
                         )

end=datetime.now()

print('Process time:', end-start)
```

```
Epoch 45/50
48/48 [==============================] - 19s 396ms/step - loss: 0.0715 -
acc: 0.9760 - val_loss: 0.1542 - val_acc: 0.9438
Epoch 46/50
48/48 [==============================] - 19s 394ms/step - loss: 0.0672 -
acc: 0.9771 - val_loss: 0.1797 - val_acc: 0.9344
Epoch 47/50
48/48 [==============================] - 19s 405ms/step - loss: 0.0492 -
acc: 0.9833 - val_loss: 0.2244 - val_acc: 0.9281
Epoch 48/50
48/48 [==============================] - 20s 407ms/step - loss: 0.0693 -
acc: 0.9729 - val_loss: 0.1699 - val_acc: 0.9531
Epoch 49/50
48/48 [==============================] - 19s 399ms/step - loss: 0.0698 -
acc: 0.9750 - val_loss: 0.1427 - val_acc: 0.9469
Epoch 50/50
48/48 [==============================] - 20s 407ms/step - loss: 0.0766 -
acc: 0.9729 - val_loss: 0.1831 - val_acc: 0.9375
Process time: 0:16:06.312513
```

In [703]:

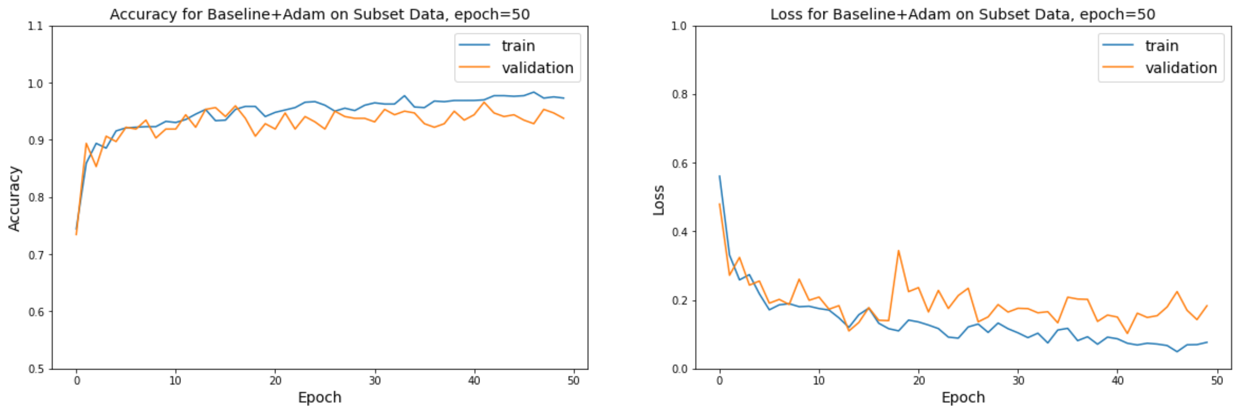
```python
model_13.save('saved_model_history/model_13.h5')
np.save('saved_model_history/history_13.npy', history_13.history)
```

In [704]:

```python
evaluate(model_13)
```

```
129/129 [==============================] - 55s 425ms/step - loss: 0.0818
- acc: 0.9661
28/28 [==============================] - 12s 425ms/step - loss: 0.1697 -
acc: 0.9431
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.9661
Train Loss = 0.0818
----------------------------
Test Accuracy = 0.9431
Test Loss = 0.1697
```

In [712]:
```
plot_acc_loss(history_13, 13, 'Baseline+Adam on Subset Data, epoch=50')
```



Comments

- Running the model on subset of data with epoch=50 did not improve the model.
- The testing accuracy is slightly worse then model 9 (subset data with epoch=30) and model 12 (whole dataset, epoch=30).
- The testing loss larger than model 9, but smaller than model 12.
- Ovefitting is more than model 9, but less than model 12.
- Running on larger epoch increased the overfitting, so negatively affected the model performance.
- Runtime: 16 mins

Final Model

The model with best performance is model_9. It is my final model.

- The final model has 6 CNN layers and 2Dense Layers.
- Optimizer = 'adam'
- Epochs = 30
- Trained on subset of data (~20% of whole data)

The structure of the model, and compile and fit steps are shown below.

In [716]:

```
# Baseline Model
# Optiizer = 'adam'

model_9 = models.Sequential()
model_9.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 6
model_9.add(layers.MaxPooling2D((2, 2)))

model_9.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_9.add(layers.MaxPooling2D((2, 2)))

model_9.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_9.add(layers.MaxPooling2D((2, 2)))

model_9.add(layers.Flatten())
model_9.add(layers.Dense(64, activation='relu'))
model_9.add(layers.Dense(1, activation='sigmoid'))

model_9.compile(loss='binary_crossentropy',
              optimizer= 'adam',
              metrics=['acc'])
start=datetime.now()

history_9 = model_9.fit(train_generator_b20,
                        steps_per_epoch=48, # 48x20 training samaples will run
                        epochs=30,
                        validation_data=val_generator_b20,
                        validation_steps=16,  # 16x20 validation samaples will
                       )

end=datetime.now()

print('Process time:', end-start)
```

I already have trained the subset data with this model. So I do not run model training again.

The performance of the model is shown below.

In [729]:

```
evaluate(model_9)
```

```
129/129 [==============================] - 56s 436ms/step - loss: 0.0774
- acc: 0.9736
28/28 [==============================] - 13s 479ms/step - loss: 0.1214 -
acc: 0.9545
----------------------------
Model Evaluation
----------------------------
Train Accuracy = 0.9736
Train Loss = 0.0774
----------------------------
Test Accuracy = 0.9545
Test Loss = 0.1214
```

Future Work

- Even though, the model performance is good. There is always a room for improvement.
- In this study, my main limitation was computing power.
  - I did most of the model training in a subset of data. (~20% of training and validation)
  - Used (64x64) images instead of (128x128) or (256x256).
  - Used batch_size=20 instead of 32, beacuse of the small sample size.
  - Ran with a small epoch number(30).
- I would like run all steps of this analysis with whole dataset on a powerful computer, or grid system. In my analysis, I did run on the whole data just once. And it didn't give the best performance, mainly due to overfitting. There might also be other issues. I didn't have power to investigate and tune the model with whole dataset.
- I would like to study augmentation more. Theoratically, it should improve the model performance. However, it didn't on my subset data. Why? Would it yield better performance in whole data? I would like to investigate this more.