Pneumonia Project - Code

In [ ]:
```
#!pip install opencv-python
```

In [ ]:
```
#!pip install tensorflow
```

In [ ]:
```
import os
import numpy as np
import cv2
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
```

In [ ]:
```
# Function to load images and labels in grayscale
def load_images_and_labels(folder):
    images = []
    labels = []

    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        if filename.startswith('NORMAL'):
            labels.append('NORMAL')
        elif filename.startswith('VIRUS') or filename.startswith('BACTERIA'):
            labels.append('PNEUMONIA')
        else:
            # Skip unrelated files
            continue

        if os.path.isfile(img_path):
            # Read image as grayscale
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            if img is not None:
                images.append(img)

    return images, labels
```

In [ ]:
```
# Load images and labels from folder
folder_path = "/Users/andrewgatchalian/Documents/UCI MSBA 24/Spring Quarter/Deep Learning/Project/xray_data
images, labels = load_images_and_labels(folder_path)
```

In [ ]:
```
# Convert labels to numerical values
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
```

In [ ]:
```
# Preprocess images (resize, normalize, etc.)
def preprocess_images(images, size=(64, 64)):
    processed_images = []
    for img in images:
        # Resize images to the specified size
        resized_img = cv2.resize(img, size)

        # Normalize pixel values to be between 0 and 1
        normalized_img = resized_img / 255.0

        # Expand dimensions to retain consistency in shape for deep learning models
        processed_images.append(np.expand_dims(normalized_img, axis=-1))
    return np.array(processed_images)

processed_images = preprocess_images(images)
```

In [ ]:
```
# Split the data into training, cross-validation, and test sets
x_train, x_test, y_train, y_test = train_test_split(processed_images, labels_encoded, test_size=0.2, train_
x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, test_size=0.25, train_size=0.75, random_s
```

In [ ]:
```
# Report the number of examples in each set
num_train_examples = len(x_train)
num_cv_examples = len(x_cv)
num_test_examples = len(x_test)

print(f"Number of examples in training set: {num_train_examples}")
print(f"Number of examples in cross-validation set: {num_cv_examples}")
print(f"Number of examples in test set: {num_test_examples}")
classes = ['NORMAL', 'PNEUMONIA']
```

```
Number of examples in training set: 3513
Number of examples in cross-validation set: 1171
Number of examples in test set: 1172
```

In [ ]:
```
# import
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
from PIL import Image
from scipy import ndimage
#from lr_utils import load_dataset

%matplotlib inline
```

Import

In [ ]:
```
import tensorflow as tf
```

In [ ]:
```
tf.__version__
```

Out[ ]:
```
'2.16.1'
```

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.preprocessing import image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

1) 2 Layer DNN

In [ ]:
```
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # Input shape adjusted to 64x64x1
    Dense(64, activation='relu'),
```

```
    Dense(1, activation='sigmoid')
])

from tensorflow.keras.metrics import Recall

# loss and optimizer
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2
)
```

```
/Users/andrewgatchalian/anaconda3/lib/python3.11/site-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
Epoch 1/8
14/14 - 2s - 127ms/step - accuracy: 0.6305 - loss: 0.8491 - recall_1: 0.7689 - val_accuracy: 0.8454 - val_l
oss: 0.5297 - val_recall_1: 0.8970
Epoch 2/8
14/14 - 0s - 12ms/step - accuracy: 0.7976 - loss: 0.4650 - recall_1: 0.9329 - val_accuracy: 0.8412 - val_lo
ss: 0.3985 - val_recall_1: 0.9799
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.8395 - loss: 0.3741 - recall_1: 0.9727 - val_accuracy: 0.8463 - val_los
s: 0.3516 - val_recall_1: 0.9882
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.8756 - loss: 0.3275 - recall_1: 0.9660 - val_accuracy: 0.8915 - val_los
s: 0.3136 - val_recall_1: 0.9420
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.8907 - loss: 0.2964 - recall_1: 0.9590 - val_accuracy: 0.8864 - val_los
s: 0.2867 - val_recall_1: 0.9799
Epoch 6/8
14/14 - 0s - 7ms/step - accuracy: 0.8950 - loss: 0.2731 - recall_1: 0.9617 - val_accuracy: 0.9069 - val_los
s: 0.2607 - val_recall_1: 0.9763
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.9035 - loss: 0.2570 - recall_1: 0.9602 - val_accuracy: 0.9103 - val_los
s: 0.2451 - val_recall_1: 0.9811
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.9137 - loss: 0.2386 - recall_1: 0.9621 - val_accuracy: 0.9146 - val_los
s: 0.2284 - val_recall_1: 0.9787
```

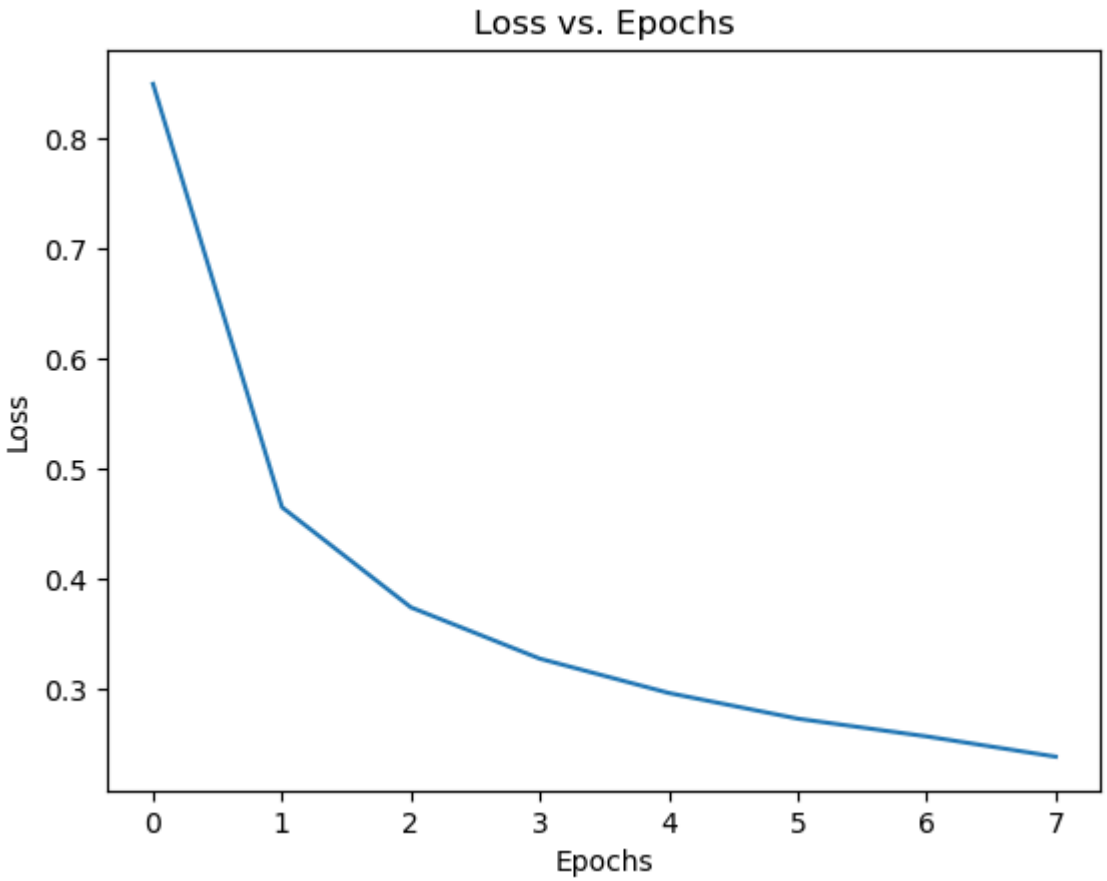
In [ ]:
```
df = pd.DataFrame(history.history)
df.head()
```

Out[ ]:

	accuracy	loss	recall_1	val_accuracy	val_loss	val_recall_1
0	0.630515	0.849054	0.768930	0.845431	0.529701	0.897041
1	0.797609	0.464972	0.932865	0.841161	0.398527	0.979882
2	0.839453	0.374061	0.972678	0.846285	0.351608	0.988166
3	0.875605	0.327516	0.966042	0.891546	0.313555	0.942012
4	0.890692	0.296373	0.959016	0.886422	0.286661	0.979882

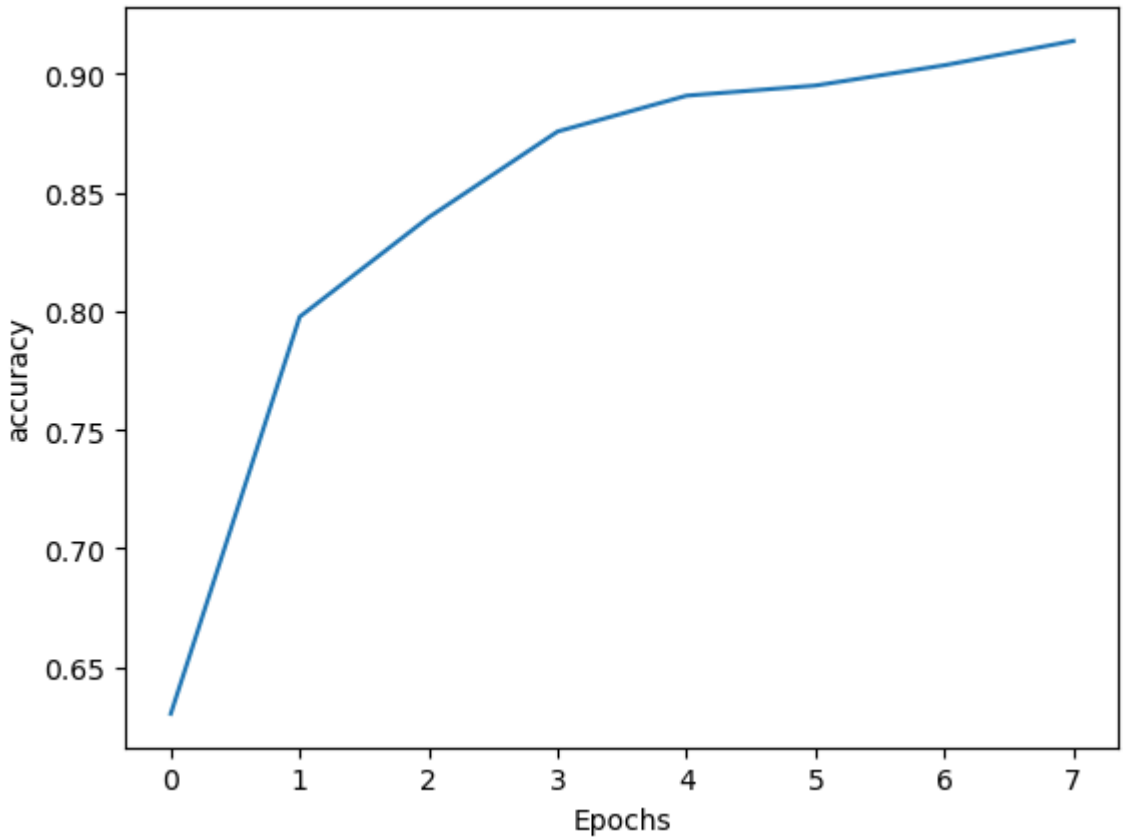
In [ ]:
```
loss_plot = df.plot(y="loss", title = "Loss vs. Epochs", legend=False)
loss_plot.set(xlabel="Epochs", ylabel="Loss")
```

Out[ ]: [Text(0.5, 0, 'Epochs'), Text(0, 0.5, 'Loss')]



In [ ]:
```
accuracy_plot = df.plot(y="accuracy", legend=False)
accuracy_plot.set(xlabel="Epochs", ylabel="accuracy")
```

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



In [ ]:
```
model.evaluate(x_test, y_test)
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9228 - loss: 0.2280 - recall_1: 0.9762
```

Out[ ]:
```
[0.2286173552274704, 0.9172354936599731, 0.9734411239624023]
```

In [ ]:
```
model.evaluate(x_train, y_train)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - accuracy: 0.9122 - loss: 0.2228 - recall_1: 0.9709
```

Out[ ]:
```
[0.23159778118133545, 0.9117563366889954, 0.9738485813140869]
```

In [ ]:
```
model.evaluate(x_cv, y_cv)
```

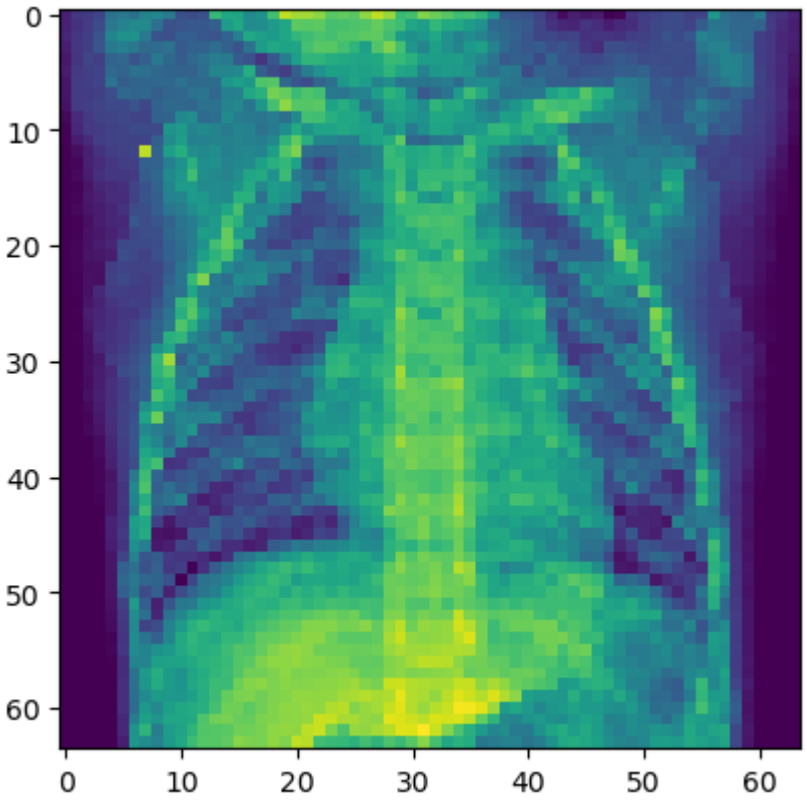
```
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 949us/step - accuracy: 0.9188 - loss: 0.2371 - recall_1: 0.9767
```

Out[ ]:
```
[0.22843246161937714, 0.9146028757095337, 0.9786982536315918]
```

Model Predict

In [ ]:
```
# Choose a random test image

random_inx = np.random.choice(x_test.shape[0])
X_sample = x_test[random_inx]
print(random_inx)
plt.imshow(X_sample)
plt.show()
print(f"Label: {[y_test[random_inx]]}")
```

```
1156
```




```
Label: [0]
```

In [ ]:
```
X_sample = X_sample.reshape(1, 64, 64) # X_sample: (num_samples, X_sample.shape)
X_sample.shape
```

Out[ ]:
```
(1, 64, 64)
```

In [ ]:
```
model.predict(X_sample)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```

Out[ ]:
```
array([[0.2868394]], dtype=float32)
```

In [ ]:
```
predictions = model.predict(X_sample)
print(np.argmax(predictions))
print(f"Model prediction:{[np.argmax(predictions)]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
0
Model prediction:[0]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
0
Model prediction:[0]
```

Model Fit With Validation

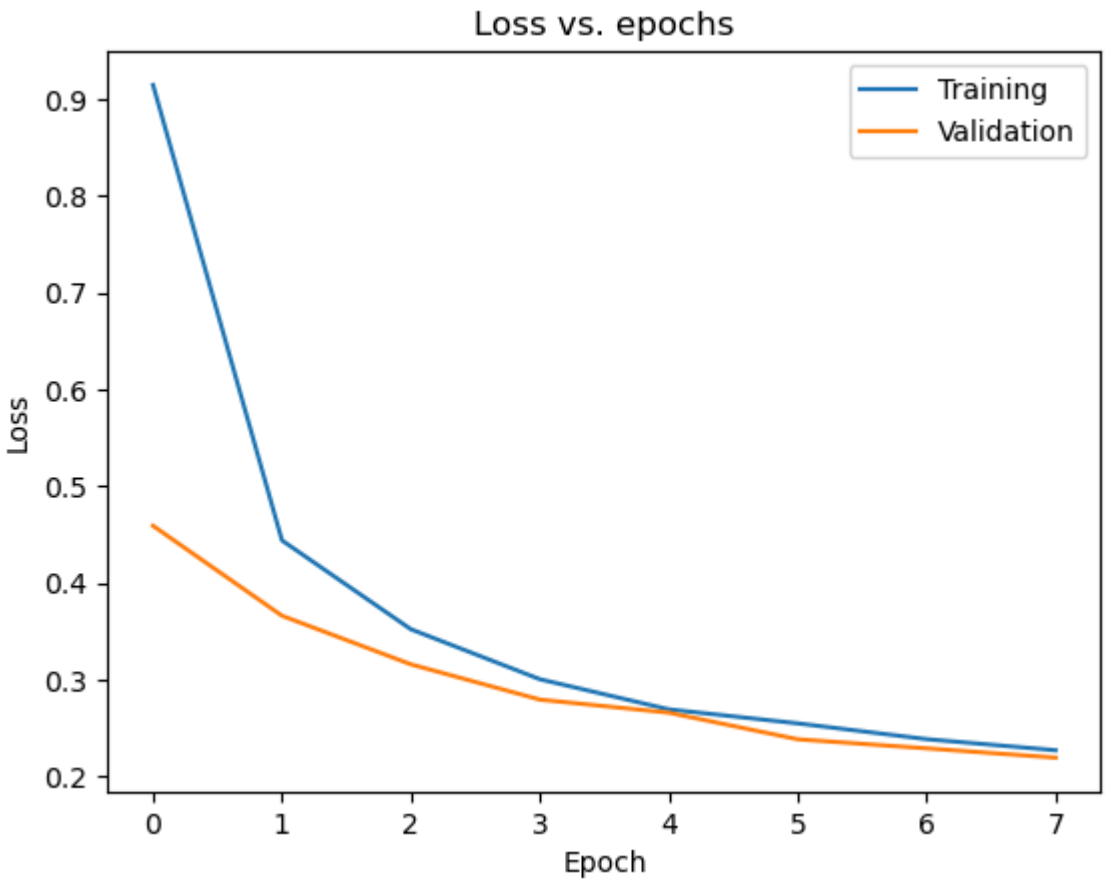
In [ ]:
```
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # Input shape adjusted to 64x64x1
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

from tensorflow.keras.metrics import Recall

# loss and optimizer
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
```

```
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
```

```
Epoch 1/8
/Users/andrewgatchalian/anaconda3/lib/python3.11/site-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
14/14 - 1s - 61ms/step - accuracy: 0.6533 - loss: 0.9146 - recall_2: 0.7721 - val_accuracy: 0.7301 - val_loss: 0.4589 - val_recall_2: 0.9988
Epoch 2/8
Epoch 2/8
14/14 - 0s - 7ms/step - accuracy: 0.7885 - loss: 0.4437 - recall_2: 0.9602 - val_accuracy: 0.8762 - val_loss: 0.3659 - val_recall_2: 0.9444
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.8548 - loss: 0.3519 - recall_2: 0.9301 - val_accuracy: 0.8719 - val_loss: 0.3156 - val_recall_2: 0.9822
Epoch 4/8
14/14 - 0s - 8ms/step - accuracy: 0.8799 - loss: 0.3002 - recall_2: 0.9633 - val_accuracy: 0.9009 - val_loss: 0.2791 - val_recall_2: 0.9633
Epoch 5/8
14/14 - 0s - 32ms/step - accuracy: 0.8989 - loss: 0.2690 - recall_2: 0.9598 - val_accuracy: 0.8950 - val_loss: 0.2656 - val_recall_2: 0.9882
Epoch 6/8
14/14 - 0s - 16ms/step - accuracy: 0.9044 - loss: 0.2546 - recall_2: 0.9590 - val_accuracy: 0.9146 - val_loss: 0.2381 - val_recall_2: 0.9799
Epoch 7/8
14/14 - 0s - 9ms/step - accuracy: 0.9106 - loss: 0.2382 - recall_2: 0.9668 - val_accuracy: 0.9274 - val_loss: 0.2287 - val_recall_2: 0.9444
Epoch 8/8
14/14 - 0s - 9ms/step - accuracy: 0.9174 - loss: 0.2268 - recall_2: 0.9614 - val_accuracy: 0.9172 - val_loss: 0.2191 - val_recall_2: 0.9893
```

In [ ]:
```
# Plot the training and validation loss

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

```
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [ ]:
```
# Testing mode

model.evaluate(x_cv, y_cv)
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9183 - loss: 0.2283 - recall_2: 0.9843
```

Out[ ]:
```
[0.219087615609169, 0.9171648025512695, 0.9893491268157959]
```

2) Change the NN Model

In [ ]:

```
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # Input shape adjusted to 64x64x1
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

from tensorflow.keras.metrics import Recall

# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
```

```
Epoch 1/8
14/14 - 1s - 69ms/step - accuracy: 0.6049 - loss: 1.7733 - recall_3: 0.7158 - val_accuracy: 0.7216 - val_lo
ss: 0.5369 - val_recall_3: 1.0000
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.7763 - loss: 0.4663 - recall_3: 0.9001 - val_accuracy: 0.8736 - val_los
s: 0.3445 - val_recall_3: 0.9491
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.8662 - loss: 0.3217 - recall_3: 0.9489 - val_accuracy: 0.8762 - val_los
s: 0.2952 - val_recall_3: 0.9822
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.8876 - loss: 0.2850 - recall_3: 0.9493 - val_accuracy: 0.9044 - val_los
s: 0.2522 - val_recall_3: 0.9740
Epoch 5/8
14/14 - 0s - 8ms/step - accuracy: 0.9038 - loss: 0.2436 - recall_3: 0.9528 - val_accuracy: 0.9129 - val_los
s: 0.2309 - val_recall_3: 0.9858
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.9132 - loss: 0.2295 - recall_3: 0.9617 - val_accuracy: 0.9308 - val_los
s: 0.2052 - val_recall_3: 0.9704
Epoch 7/8
14/14 - 0s - 9ms/step - accuracy: 0.9214 - loss: 0.2074 - recall_3: 0.9649 - val_accuracy: 0.9377 - val_los
s: 0.1956 - val_recall_3: 0.9538
Epoch 8/8
14/14 - 0s - 9ms/step - accuracy: 0.9308 - loss: 0.1950 - recall_3: 0.9684 - val_accuracy: 0.9453 - val_los
s: 0.1804 - val_recall_3: 0.9822
```

In [ ]:
```python
import numpy as np

unique, counts = np.unique(y_train, return_counts=True)
print(dict(zip(unique, counts)))
```

```
{0: 951, 1: 2562}
```

In [ ]:
```python
model.evaluate(x_train, y_train)
from sklearn.metrics import confusion_matrix

# Predict the training data
y_pred = model.predict(x_train)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_train, y_pred_classes)
print(cm)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 903us/step - accuracy: 0.9306 - loss: 0.1814 - recall_3: 0.9722
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 746us/step
[[ 776  175]
 [  65 2497]]
```

In [ ]:
```python
model.evaluate(x_test, y_test)
from sklearn.metrics import confusion_matrix

# Predict the training data
y_pred = model.predict(x_test)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
print(cm)
```

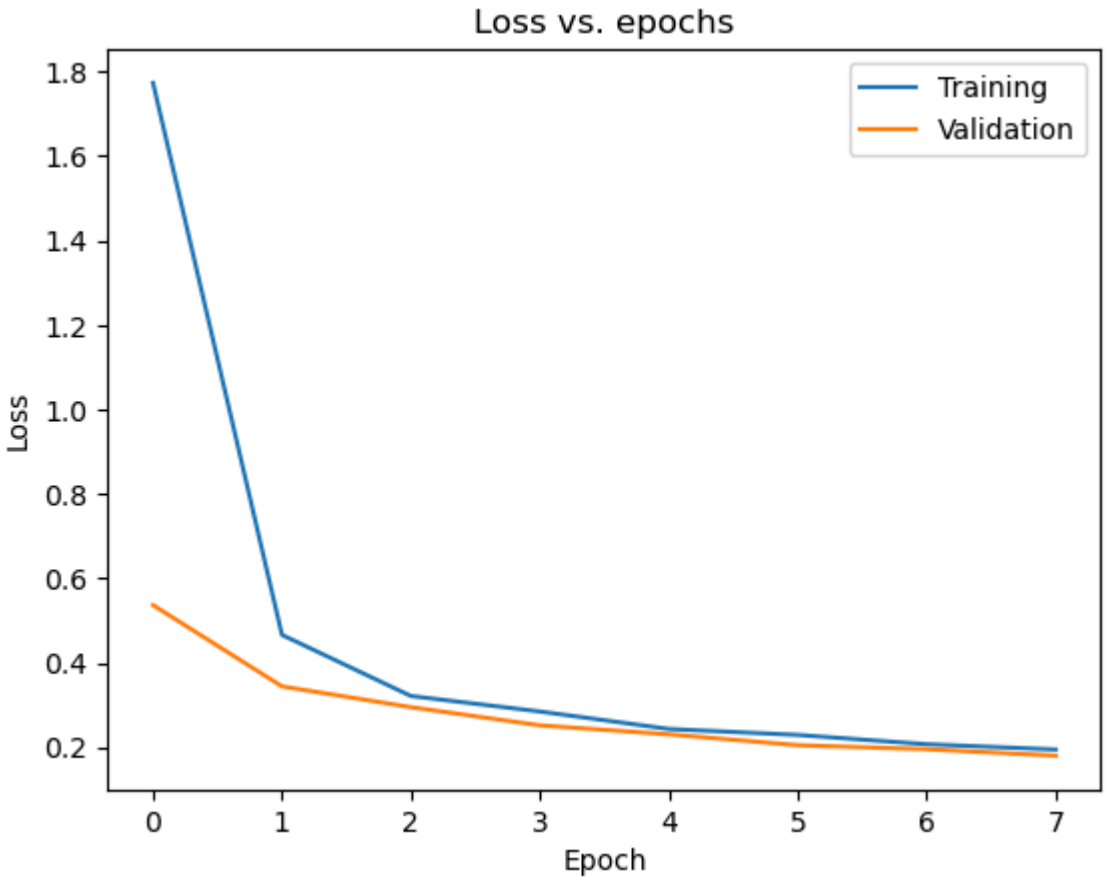
```
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 869us/step - accuracy: 0.9285 - loss: 0.1848 - recall_3: 0.9642
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step
[[256  50]
 [ 29 837]]
```

In [ ]:
```python
model.evaluate(x_cv, y_cv)
# Predict the training data
y_pred = model.predict(x_cv)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_cv, y_pred_classes)
print(cm)
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 751us/step - accuracy: 0.9477 - loss: 0.1913 - recall_3: 0.9777
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 672us/step
[[277  49]
 [ 15 830]]
```

In [ ]:
```python
# Plot the training and validation loss

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

```
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



3) L2 Regularization (L2 LAMBDA = 0.001)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)), # per
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))
])

from tensorflow.keras.metrics import Recall
```

```python
# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
```

```
Epoch 1/8
```

```
/Users/andrewgatchalian/anaconda3/lib/python3.11/site-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

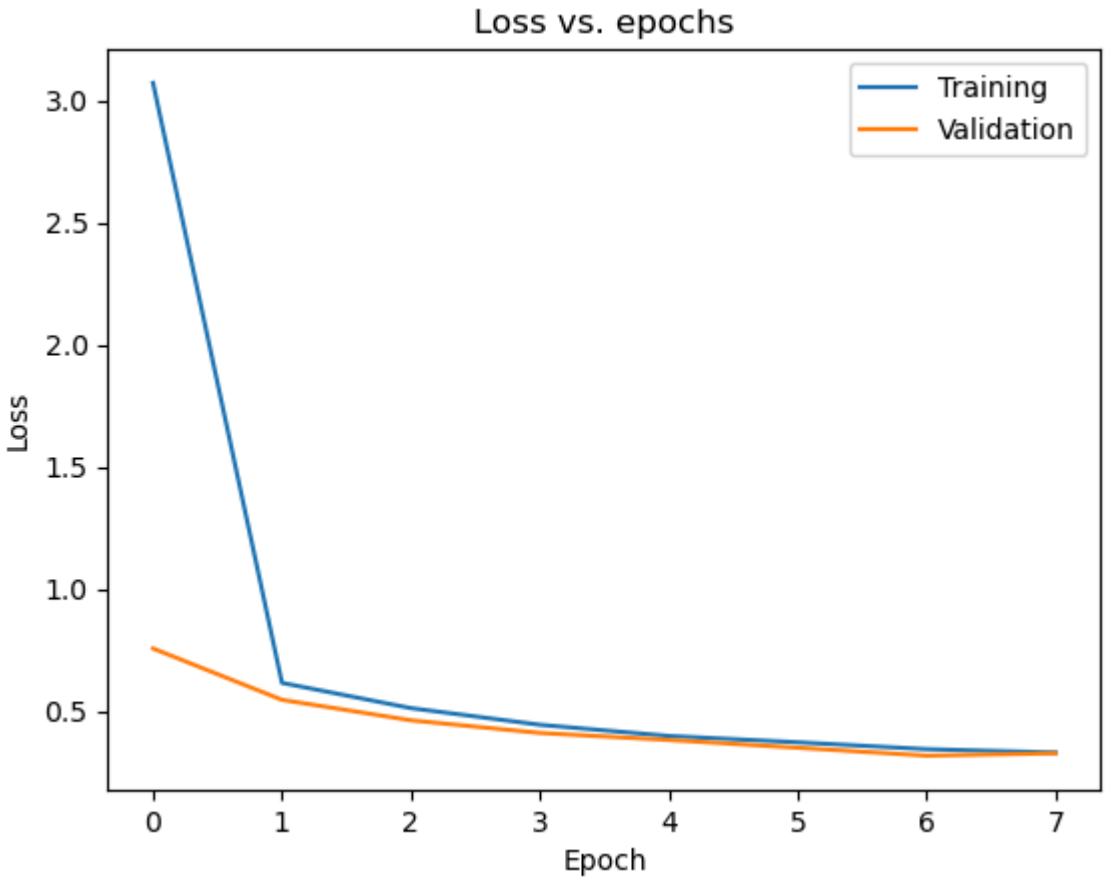
```
14/14 - 1s - 66ms/step - accuracy: 0.5881 - loss: 3.0727 - recall_4: 0.7037 - val_accuracy: 0.7216 - val_lo
ss: 0.7565 - val_recall_4: 1.0000
Epoch 2/8
Epoch 2/8
14/14 - 0s - 25ms/step - accuracy: 0.8135 - loss: 0.6156 - recall_4: 0.9852 - val_accuracy: 0.8813 - val_lo
ss: 0.5462 - val_recall_4: 0.9598
Epoch 3/8
14/14 - 0s - 15ms/step - accuracy: 0.8782 - loss: 0.5122 - recall_4: 0.9567 - val_accuracy: 0.9044 - val_lo
ss: 0.4623 - val_recall_4: 0.9621
Epoch 4/8
14/14 - 0s - 13ms/step - accuracy: 0.9026 - loss: 0.4440 - recall_4: 0.9582 - val_accuracy: 0.9180 - val_lo
ss: 0.4105 - val_recall_4: 0.9751
Epoch 5/8
14/14 - 0s - 9ms/step - accuracy: 0.9174 - loss: 0.3982 - recall_4: 0.9660 - val_accuracy: 0.9231 - val_los
s: 0.3818 - val_recall_4: 0.9302
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.9243 - loss: 0.3723 - recall_4: 0.9617 - val_accuracy: 0.9266 - val_los
s: 0.3500 - val_recall_4: 0.9893
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.9249 - loss: 0.3441 - recall_4: 0.9692 - val_accuracy: 0.9479 - val_los
s: 0.3175 - val_recall_4: 0.9811
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.9291 - loss: 0.3303 - recall_4: 0.9617 - val_accuracy: 0.9214 - val_los
s: 0.3273 - val_recall_4: 0.9917
```

In [ ]:
```
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 812us/step - accuracy: 0.9165 - loss: 0.3251 - recall_4: 0.9847
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 735us/step - accuracy: 0.9217 - loss: 0.3298 - recall_4: 0.9830
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 725us/step - accuracy: 0.9247 - loss: 0.3387 - recall_4: 0.9900
```

Out[ ]:
```
[0.32734787464141846, 0.9214347004890442, 0.9917159676551819]
```

In [ ]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



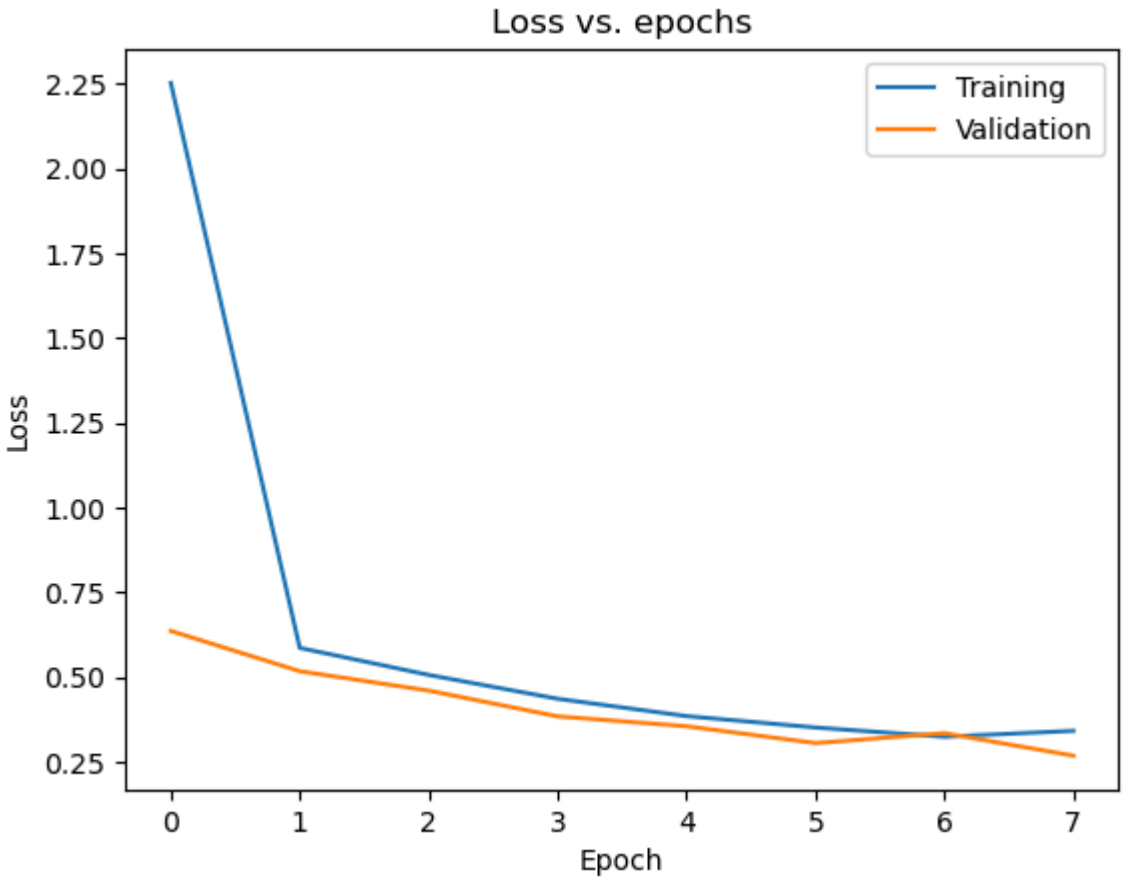
4) L2 LAMBDA = 0.002

In [ ]:

```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.002)), # per
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.002))
])


from tensorflow.keras.metrics import Recall

# loss and optimizer
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()

model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 65ms/step - accuracy: 0.6550 - loss: 2.2531 - recall_5: 0.7678 - val_accuracy: 0.7874 - val_lo
ss: 0.6361 - val_recall_5: 0.9929
Epoch 2/8
14/14 - 0s - 7ms/step - accuracy: 0.8582 - loss: 0.5860 - recall_5: 0.9372 - val_accuracy: 0.9018 - val_los
s: 0.5172 - val_recall_5: 0.9657
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.8924 - loss: 0.5062 - recall_5: 0.9391 - val_accuracy: 0.9120 - val_los
s: 0.4599 - val_recall_5: 0.9882
Epoch 4/8
14/14 - 0s - 9ms/step - accuracy: 0.9163 - loss: 0.4358 - recall_5: 0.9621 - val_accuracy: 0.9402 - val_los
s: 0.3839 - val_recall_5: 0.9704
Epoch 5/8
14/14 - 0s - 9ms/step - accuracy: 0.9206 - loss: 0.3847 - recall_5: 0.9641 - val_accuracy: 0.9342 - val_los
s: 0.3547 - val_recall_5: 0.9361
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.9240 - loss: 0.3510 - recall_5: 0.9575 - val_accuracy: 0.9488 - val_los
s: 0.3051 - val_recall_5: 0.9811
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.9243 - loss: 0.3238 - recall_5: 0.9633 - val_accuracy: 0.9095 - val_los
s: 0.3343 - val_recall_5: 0.9941
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.9075 - loss: 0.3412 - recall_5: 0.9539 - val_accuracy: 0.9513 - val_los
s: 0.2679 - val_recall_5: 0.9858
```




```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 926us/step - accuracy: 0.9352 - loss: 0.2708 - recall_5: 0.9782
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - accuracy: 0.9351 - loss: 0.2806 - recall_5: 0.9693
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - accuracy: 0.9468 - loss: 0.2808 - recall_5: 0.9791
```

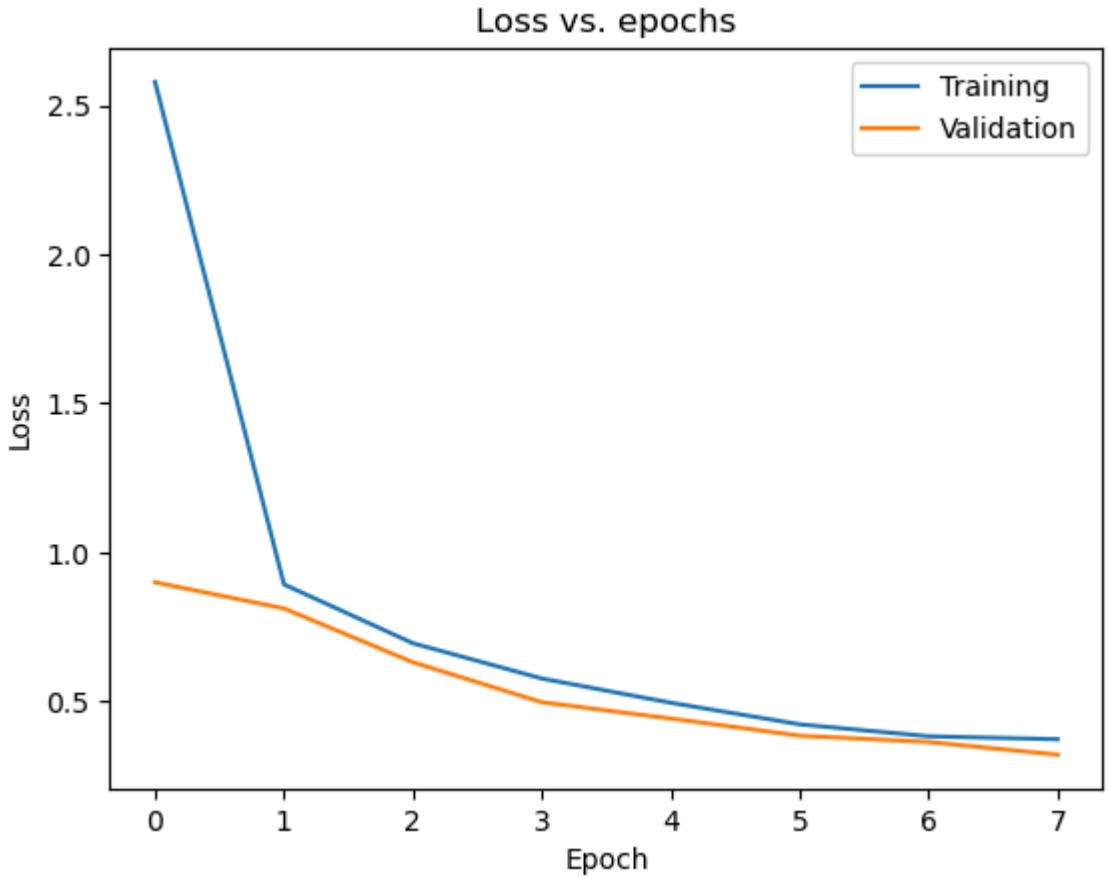
Out[ ]: [0.26791366934776306, 0.9513236284255981, 0.9857988357543945]

4) L2 LAMBDA = 0.003

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.003)), # per
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.003))
])
```

```
from tensorflow.keras.metrics import Recall

# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()

model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 65ms/step - accuracy: 0.6206 - loss: 2.5795 - recall_6: 0.7385 - val_accuracy: 0.7916 - val_lo
ss: 0.8991 - val_recall_6: 0.7538
Epoch 2/8
14/14 - 0s - 11ms/step - accuracy: 0.7788 - loss: 0.8920 - recall_6: 0.8774 - val_accuracy: 0.8104 - val_lo
ss: 0.8113 - val_recall_6: 0.7669
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.8662 - loss: 0.6943 - recall_6: 0.9309 - val_accuracy: 0.8796 - val_los
s: 0.6305 - val_recall_6: 0.8781
Epoch 4/8
14/14 - 0s - 8ms/step - accuracy: 0.8822 - loss: 0.5759 - recall_6: 0.9446 - val_accuracy: 0.9197 - val_los
s: 0.4962 - val_recall_6: 0.9740
Epoch 5/8
14/14 - 0s - 9ms/step - accuracy: 0.9032 - loss: 0.4945 - recall_6: 0.9567 - val_accuracy: 0.9274 - val_los
s: 0.4415 - val_recall_6: 0.9420
Epoch 6/8
14/14 - 0s - 9ms/step - accuracy: 0.9203 - loss: 0.4220 - recall_6: 0.9637 - val_accuracy: 0.9377 - val_los
s: 0.3839 - val_recall_6: 0.9680
Epoch 7/8
14/14 - 0s - 12ms/step - accuracy: 0.9231 - loss: 0.3819 - recall_6: 0.9641 - val_accuracy: 0.9223 - val_lo
ss: 0.3627 - val_recall_6: 0.9893
Epoch 8/8
14/14 - 0s - 14ms/step - accuracy: 0.9157 - loss: 0.3720 - recall_6: 0.9559 - val_accuracy: 0.9479 - val_lo
ss: 0.3201 - val_recall_6: 0.9811
```




```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9325 - loss: 0.3204 - recall_6: 0.9723
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9291 - loss: 0.3287 - recall_6: 0.9679
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 894us/step - accuracy: 0.9497 - loss: 0.3304 - recall_6: 0.9771
```

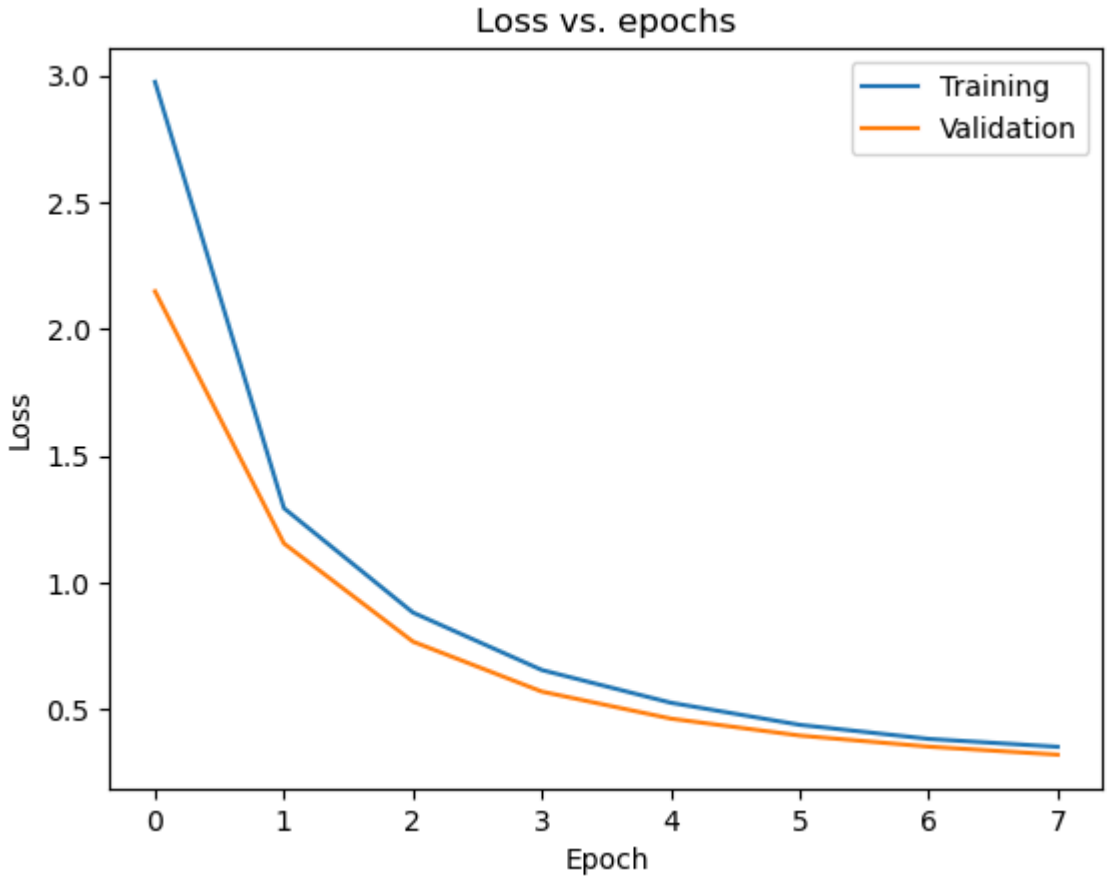
Out[ ]: [0.32013750076293945, 0.9479077458381653, 0.9810650944709778]

5) L2 LAMBDA = 0.004

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.004)), # per
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.004))
])
```

```
from tensorflow.keras.metrics import Recall

# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()

model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 64ms/step - accuracy: 0.6086 - loss: 2.9748 - recall_7: 0.6827 - val_accuracy: 0.7216 - val_lo
ss: 2.1494 - val_recall_7: 1.0000
Epoch 2/8
14/14 - 0s - 7ms/step - accuracy: 0.7486 - loss: 1.2944 - recall_7: 0.8474 - val_accuracy: 0.7523 - val_los
s: 1.1554 - val_recall_7: 0.9976
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.8412 - loss: 0.8831 - recall_7: 0.9188 - val_accuracy: 0.8625 - val_los
s: 0.7686 - val_recall_7: 0.9905
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.8935 - loss: 0.6565 - recall_7: 0.9508 - val_accuracy: 0.9120 - val_los
s: 0.5715 - val_recall_7: 0.9751
Epoch 5/8
14/14 - 0s - 9ms/step - accuracy: 0.9109 - loss: 0.5276 - recall_7: 0.9559 - val_accuracy: 0.9291 - val_los
s: 0.4650 - val_recall_7: 0.9775
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.9220 - loss: 0.4404 - recall_7: 0.9657 - val_accuracy: 0.9377 - val_los
s: 0.3982 - val_recall_7: 0.9787
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.9268 - loss: 0.3850 - recall_7: 0.9688 - val_accuracy: 0.9445 - val_los
s: 0.3545 - val_recall_7: 0.9621
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.9263 - loss: 0.3539 - recall_7: 0.9680 - val_accuracy: 0.9445 - val_los
s: 0.3228 - val_recall_7: 0.9598
```




```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9370 - loss: 0.3239 - recall_7: 0.9578
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - accuracy: 0.9247 - loss: 0.3342 - recall_7: 0.9403
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 857us/step - accuracy: 0.9433 - loss: 0.3342 - recall_7: 0.9608
```

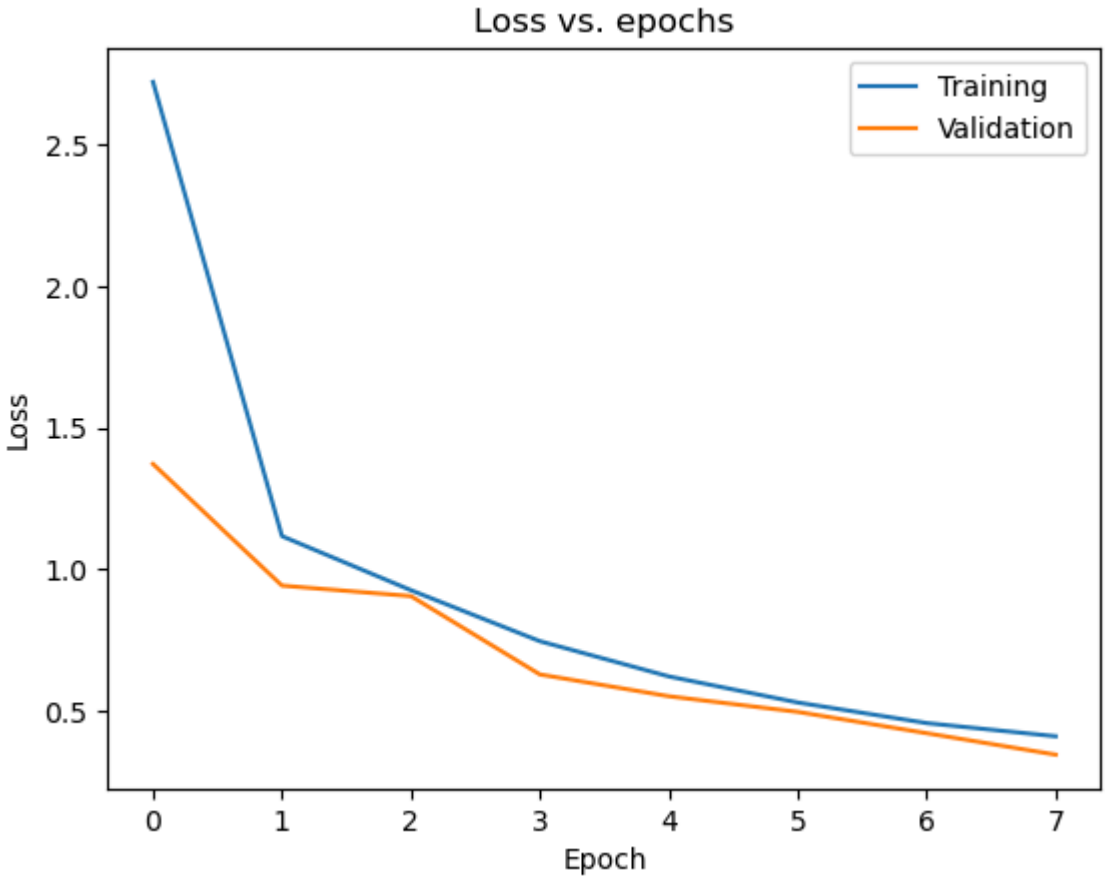
Out[ ]: [0.3228365182876587, 0.9444918632507324, 0.9597632884979248]

6) L2 LAMBDA = 0.005

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.005)), # per
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.005))
])
```

```
# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()

model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 90ms/step - accuracy: 0.5810 - loss: 2.7226 - recall_8: 0.6690 - val_accuracy: 0.7216 - val_lo
ss: 1.3716 - val_recall_8: 1.0000
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.7990 - loss: 1.1163 - recall_8: 0.8860 - val_accuracy: 0.8915 - val_los
s: 0.9414 - val_recall_8: 0.9302
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.8645 - loss: 0.9251 - recall_8: 0.9321 - val_accuracy: 0.8386 - val_los
s: 0.9042 - val_recall_8: 0.7917
Epoch 4/8
14/14 - 0s - 8ms/step - accuracy: 0.8913 - loss: 0.7448 - recall_8: 0.9379 - val_accuracy: 0.9249 - val_los
s: 0.6273 - val_recall_8: 0.9882
Epoch 5/8
14/14 - 0s - 9ms/step - accuracy: 0.8984 - loss: 0.6203 - recall_8: 0.9399 - val_accuracy: 0.9035 - val_los
s: 0.5501 - val_recall_8: 0.9941
Epoch 6/8
14/14 - 0s - 9ms/step - accuracy: 0.9058 - loss: 0.5279 - recall_8: 0.9617 - val_accuracy: 0.9061 - val_los
s: 0.4948 - val_recall_8: 0.8828
Epoch 7/8
14/14 - 0s - 9ms/step - accuracy: 0.9126 - loss: 0.4552 - recall_8: 0.9551 - val_accuracy: 0.9283 - val_los
s: 0.4194 - val_recall_8: 0.9195
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.9103 - loss: 0.4085 - recall_8: 0.9536 - val_accuracy: 0.9471 - val_los
s: 0.3435 - val_recall_8: 0.9763
```




```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 906us/step - accuracy: 0.9367 - loss: 0.3443 - recall_8: 0.9702
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 756us/step - accuracy: 0.9351 - loss: 0.3527 - recall_8: 0.9617
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - accuracy: 0.9434 - loss: 0.3552 - recall_8: 0.9707
```

Out[ ]: [0.3435116410255432, 0.9470537900924683, 0.976331353187561]

7) Dropout Regulazation (0.5)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu'),
                    Dropout(0.5), #dropout rate
                    Dense(1, activation = 'sigmoid')
```

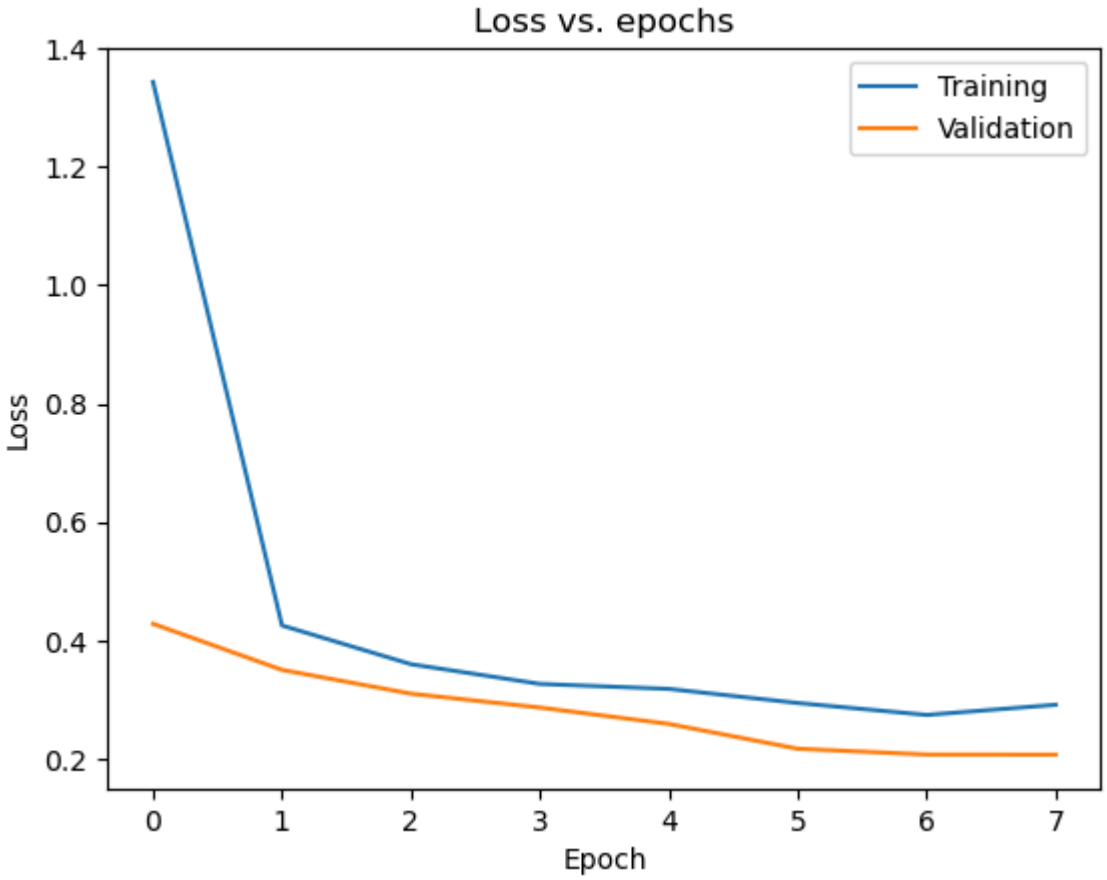
```
])

# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
```

```
Epoch 1/8
14/14 - 1s - 76ms/step - accuracy: 0.7094 - loss: 1.3435 - recall_9: 0.8603 - val_accuracy: 0.7370 - val_lo
ss: 0.4279 - val_recall_9: 1.0000
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.7868 - loss: 0.4251 - recall_9: 0.9571 - val_accuracy: 0.8429 - val_los
s: 0.3501 - val_recall_9: 0.9905
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.8395 - loss: 0.3596 - recall_9: 0.9547 - val_accuracy: 0.8813 - val_los
s: 0.3099 - val_recall_9: 0.9905
Epoch 4/8
14/14 - 0s - 8ms/step - accuracy: 0.8705 - loss: 0.3263 - recall_9: 0.9532 - val_accuracy: 0.9249 - val_los
s: 0.2864 - val_recall_9: 0.9444
Epoch 5/8
14/14 - 0s - 11ms/step - accuracy: 0.8813 - loss: 0.3179 - recall_9: 0.9504 - val_accuracy: 0.9360 - val_lo
ss: 0.2588 - val_recall_9: 0.9574
Epoch 6/8
14/14 - 0s - 19ms/step - accuracy: 0.8839 - loss: 0.2941 - recall_9: 0.9489 - val_accuracy: 0.9342 - val_lo
ss: 0.2168 - val_recall_9: 0.9763
Epoch 7/8
14/14 - 0s - 8ms/step - accuracy: 0.8901 - loss: 0.2739 - recall_9: 0.9520 - val_accuracy: 0.9342 - val_los
s: 0.2070 - val_recall_9: 0.9408
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.8841 - loss: 0.2911 - recall_9: 0.9333 - val_accuracy: 0.9325 - val_los
s: 0.2068 - val_recall_9: 0.9858
```

In [ ]:

```
# Plot the training and validation loss
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [ ]:
```
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 753us/step - accuracy: 0.9228 - loss: 0.2035 - recall_9: 0.9758
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 687us/step - accuracy: 0.9281 - loss: 0.2091 - recall_9: 0.9770
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - accuracy: 0.9328 - loss: 0.2175 - recall_9: 0.9820
```

Out[ ]: [0.2067713439464569, 0.9325363039970398, 0.9857988357543945]

8) Dropout (0.6)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu'),
                    Dropout(0.6), #dropout rate
                    Dense(1, activation = 'sigmoid')
])


# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 69ms/step - accuracy: 0.6362 - loss: 2.5082 - recall_10: 0.7830 - val_accuracy: 0.7216 - val_l
oss: 0.6281 - val_recall_10: 1.0000
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.7293 - loss: 0.6258 - recall_10: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.5781 - val_recall_10: 1.0000
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.7293 - loss: 0.6107 - recall_10: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.5443 - val_recall_10: 1.0000
Epoch 4/8
14/14 - 0s - 10ms/step - accuracy: 0.7213 - loss: 0.5897 - recall_10: 0.9832 - val_accuracy: 0.7216 - val_l
oss: 0.5010 - val_recall_10: 1.0000
Epoch 5/8
14/14 - 0s - 10ms/step - accuracy: 0.7293 - loss: 0.5522 - recall_10: 1.0000 - val_accuracy: 0.7216 - val_l
oss: 0.4982 - val_recall_10: 1.0000
Epoch 6/8
14/14 - 0s - 10ms/step - accuracy: 0.7293 - loss: 0.5431 - recall_10: 1.0000 - val_accuracy: 0.7216 - val_l
oss: 0.4419 - val_recall_10: 1.0000
Epoch 7/8
14/14 - 0s - 9ms/step - accuracy: 0.7293 - loss: 0.5292 - recall_10: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.4475 - val_recall_10: 1.0000
Epoch 8/8
14/14 - 0s - 9ms/step - accuracy: 0.7293 - loss: 0.5294 - recall_10: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.4582 - val_recall_10: 1.0000
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step - accuracy: 0.7382 - loss: 0.4421 - recall_10: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - accuracy: 0.7420 - loss: 0.4480 - recall_10: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - accuracy: 0.7253 - loss: 0.4624 - recall_10: 1.0000
```

Out[ ]: [0.45819830894470215, 0.7216054797172546, 1.0]

9) Dropout (0.4)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu'),
                    Dropout(0.4), #dropout rate
                    Dense(1, activation = 'sigmoid')
])
```

```
# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 70ms/step - accuracy: 0.6701 - loss: 1.9001 - recall_11: 0.7990 - val_accuracy: 0.4313 - val_l
oss: 0.7622 - val_recall_11: 0.2142
Epoch 2/8
14/14 - 0s - 7ms/step - accuracy: 0.7128 - loss: 0.4871 - recall_11: 0.9567 - val_accuracy: 0.7216 - val_lo
ss: 0.3950 - val_recall_11: 1.0000
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.7728 - loss: 0.3980 - recall_11: 0.9922 - val_accuracy: 0.8548 - val_lo
ss: 0.3659 - val_recall_11: 0.9917
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.8523 - loss: 0.3740 - recall_11: 0.9738 - val_accuracy: 0.8599 - val_lo
ss: 0.3368 - val_recall_11: 0.9929
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.8708 - loss: 0.3558 - recall_11: 0.9707 - val_accuracy: 0.8651 - val_lo
ss: 0.3196 - val_recall_11: 0.9941
Epoch 6/8
14/14 - 0s - 7ms/step - accuracy: 0.8719 - loss: 0.3365 - recall_11: 0.9746 - val_accuracy: 0.8856 - val_lo
ss: 0.3029 - val_recall_11: 0.9941
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.8913 - loss: 0.3185 - recall_11: 0.9750 - val_accuracy: 0.8890 - val_lo
ss: 0.2910 - val_recall_11: 0.9941
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.9035 - loss: 0.3063 - recall_11: 0.9754 - val_accuracy: 0.8984 - val_lo
ss: 0.2786 - val_recall_11: 0.9929
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - accuracy: 0.9005 - loss: 0.2693 - recall_11: 0.9878
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 846us/step - accuracy: 0.8998 - loss: 0.2731 - recall_11: 0.9858
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - accuracy: 0.9010 - loss: 0.2861 - recall_11: 0.9921
```

Out[ ]: [0.2785501480102539, 0.8983774781227112, 0.9928994178771973]

10) Dropout (0.3)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu'),
                    Dropout(0.3), #dropout rate
                    Dense(1, activation = 'sigmoid')
])

# loss and optimizer
```

```python
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 97ms/step - accuracy: 0.6342 - loss: 3.4772 - recall_12: 0.7736 - val_accuracy: 0.7216 - val_l
oss: 1.2599 - val_recall_12: 1.0000
Epoch 2/8
14/14 - 0s - 7ms/step - accuracy: 0.6502 - loss: 0.6893 - recall_12: 0.7455 - val_accuracy: 0.7216 - val_lo
ss: 0.5140 - val_recall_12: 1.0000
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.7774 - loss: 0.4720 - recall_12: 0.9290 - val_accuracy: 0.8284 - val_lo
ss: 0.4109 - val_recall_12: 0.9917
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.8118 - loss: 0.4117 - recall_12: 0.9333 - val_accuracy: 0.8847 - val_lo
ss: 0.3504 - val_recall_12: 0.9645
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.8266 - loss: 0.3771 - recall_12: 0.9438 - val_accuracy: 0.8958 - val_lo
ss: 0.3189 - val_recall_12: 0.9657
Epoch 6/8
14/14 - 0s - 6ms/step - accuracy: 0.8338 - loss: 0.3643 - recall_12: 0.9411 - val_accuracy: 0.8822 - val_lo
ss: 0.3025 - val_recall_12: 0.9858
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.8377 - loss: 0.3464 - recall_12: 0.9473 - val_accuracy: 0.9086 - val_lo
ss: 0.2813 - val_recall_12: 0.9586
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.8449 - loss: 0.3244 - recall_12: 0.9504 - val_accuracy: 0.9120 - val_lo
ss: 0.2624 - val_recall_12: 0.9834
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 671us/step - accuracy: 0.9100 - loss: 0.2552 - recall_12: 0.9754
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 749us/step - accuracy: 0.9146 - loss: 0.2603 - recall_12: 0.9765
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 767us/step - accuracy: 0.9190 - loss: 0.2692 - recall_12: 0.9811
```

Out[ ]: [0.26239901781082153, 0.9120410084724426, 0.9834319353103638]

11) Dropout (0.2)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu'),
                    Dropout(0.2), #dropout rate
                    Dense(1, activation = 'sigmoid')
])


# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 61ms/step - accuracy: 0.5482 - loss: 2.0132 - recall_13: 0.6136 - val_accuracy: 0.7216 - val_l
oss: 0.6721 - val_recall_13: 1.0000
Epoch 2/8
14/14 - 0s - 10ms/step - accuracy: 0.7293 - loss: 0.5788 - recall_13: 1.0000 - val_accuracy: 0.7216 - val_l
oss: 0.5056 - val_recall_13: 1.0000
Epoch 3/8
14/14 - 0s - 11ms/step - accuracy: 0.7293 - loss: 0.5085 - recall_13: 1.0000 - val_accuracy: 0.7216 - val_l
oss: 0.4516 - val_recall_13: 1.0000
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.4677 - recall_13: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.4072 - val_recall_13: 1.0000
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.4404 - recall_13: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.3951 - val_recall_13: 1.0000
Epoch 6/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.4245 - recall_13: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.3573 - val_recall_13: 1.0000
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.4076 - recall_13: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.3565 - val_recall_13: 1.0000
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.3949 - recall_13: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.3266 - val_recall_13: 1.0000
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 686us/step - accuracy: 0.7382 - loss: 0.3144 - recall_13: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 733us/step - accuracy: 0.7420 - loss: 0.3169 - recall_13: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - accuracy: 0.7253 - loss: 0.3315 - recall_13: 1.0000
```

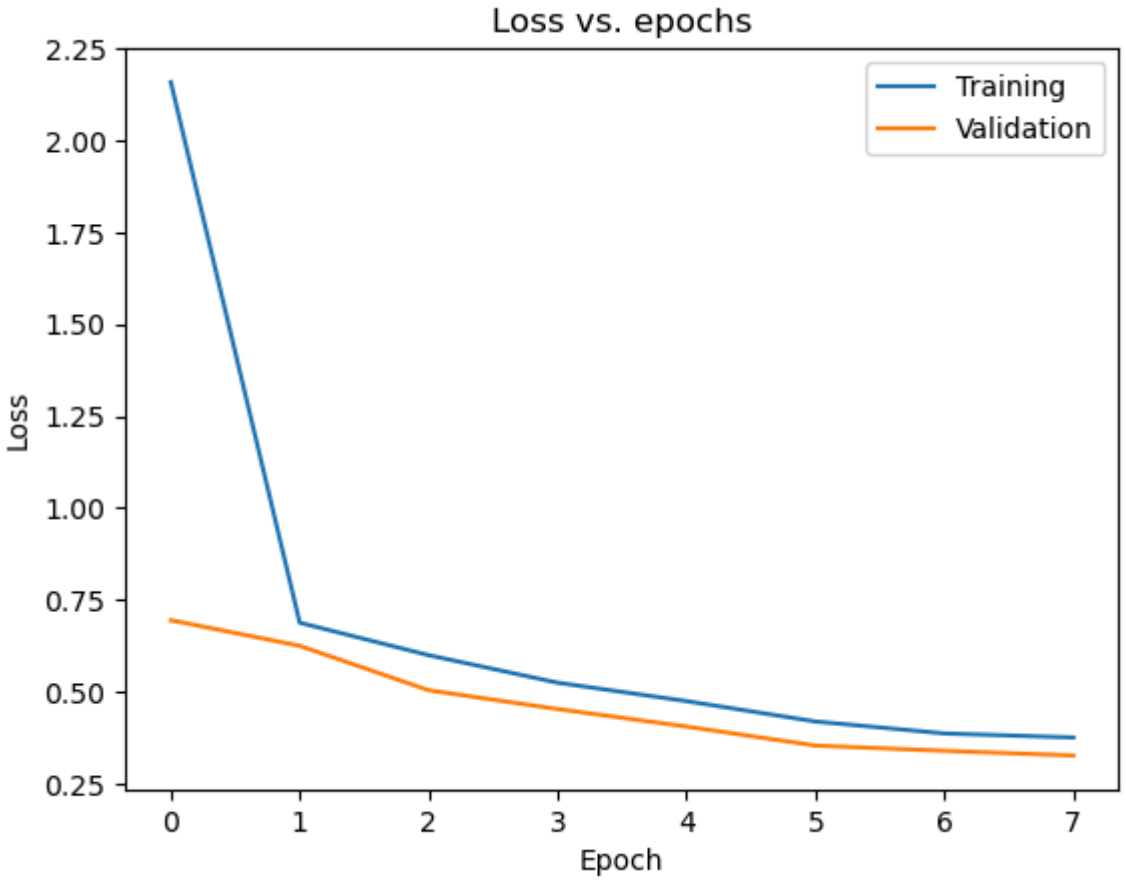
Out[ ]: [0.32661575078964233, 0.7216054797172546, 1.0]

12) Combination (L2=0.001, Dropout 0.2)

In [ ]:
```python
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)), # per
                    Dropout(0.2), #dropout rate
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))
])

# loss and optimizer
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()

model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 71ms/step - accuracy: 0.6630 - loss: 2.1585 - recall_14: 0.8353 - val_accuracy: 0.7216 - val_l
oss: 0.6946 - val_recall_14: 1.0000
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.7452 - loss: 0.6877 - recall_14: 0.9735 - val_accuracy: 0.8420 - val_lo
ss: 0.6249 - val_recall_14: 0.9893
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.8252 - loss: 0.5993 - recall_14: 0.9551 - val_accuracy: 0.8924 - val_lo
ss: 0.5039 - val_recall_14: 0.9692
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.8611 - loss: 0.5245 - recall_14: 0.9387 - val_accuracy: 0.8958 - val_lo
ss: 0.4528 - val_recall_14: 0.9893
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.8824 - loss: 0.4744 - recall_14: 0.9415 - val_accuracy: 0.9214 - val_lo
ss: 0.4052 - val_recall_14: 0.9254
Epoch 6/8
14/14 - 0s - 7ms/step - accuracy: 0.8887 - loss: 0.4190 - recall_14: 0.9489 - val_accuracy: 0.9377 - val_lo
ss: 0.3535 - val_recall_14: 0.9633
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.9103 - loss: 0.3865 - recall_14: 0.9485 - val_accuracy: 0.9223 - val_lo
ss: 0.3393 - val_recall_14: 0.9917
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.9024 - loss: 0.3755 - recall_14: 0.9446 - val_accuracy: 0.9249 - val_lo
ss: 0.3264 - val_recall_14: 0.9917
```




```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9183 - loss: 0.3271 - recall_14: 0.9829
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 896us/step - accuracy: 0.9220 - loss: 0.3319 - recall_14: 0.9820
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - accuracy: 0.9266 - loss: 0.3372 - recall_14: 0.9900
```

Out[ ]: [0.32641732692718506, 0.924850583076477, 0.9917159676551819]

13) Combination (L2 = 0.002, Dropout = 0.3)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.002)), # per
                    Dropout(0.3), #dropout rate
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.002))
```

```
])


# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 72ms/step - accuracy: 0.7085 - loss: 1.7578 - recall_15: 0.8443 - val_accuracy: 0.8711 - val_l
oss: 0.6228 - val_recall_15: 0.8947
Epoch 2/8
14/14 - 0s - 10ms/step - accuracy: 0.8295 - loss: 0.6308 - recall_15: 0.9446 - val_accuracy: 0.8950 - val_l
oss: 0.5618 - val_recall_15: 0.9751
Epoch 3/8
14/14 - 0s - 9ms/step - accuracy: 0.8776 - loss: 0.5710 - recall_15: 0.9461 - val_accuracy: 0.9180 - val_lo
ss: 0.4788 - val_recall_15: 0.9657
Epoch 4/8
14/14 - 0s - 10ms/step - accuracy: 0.9021 - loss: 0.4789 - recall_15: 0.9500 - val_accuracy: 0.9172 - val_l
oss: 0.4282 - val_recall_15: 0.9905
Epoch 5/8
14/14 - 0s - 10ms/step - accuracy: 0.9066 - loss: 0.4215 - recall_15: 0.9539 - val_accuracy: 0.9103 - val_l
oss: 0.3837 - val_recall_15: 0.9929
Epoch 6/8
14/14 - 0s - 11ms/step - accuracy: 0.9163 - loss: 0.3697 - recall_15: 0.9590 - val_accuracy: 0.8651 - val_l
oss: 0.4162 - val_recall_15: 0.9941
Epoch 7/8
14/14 - 0s - 10ms/step - accuracy: 0.9095 - loss: 0.3677 - recall_15: 0.9555 - val_accuracy: 0.9496 - val_l
oss: 0.2760 - val_recall_15: 0.9763
Epoch 8/8
14/14 - 0s - 9ms/step - accuracy: 0.9194 - loss: 0.3177 - recall_15: 0.9551 - val_accuracy: 0.9411 - val_lo
ss: 0.2644 - val_recall_15: 0.9456
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 930us/step - accuracy: 0.9362 - loss: 0.2708 - recall_15: 0.9440
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9220 - loss: 0.2834 - recall_15: 0.9261
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 756us/step - accuracy: 0.9384 - loss: 0.2766 - recall_15: 0.9471
```

Out[ ]: [0.26442039012908936, 0.9410759806632996, 0.9455621242523193]

14) Combination (L2 = 0.003, Dropout = 0.4)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.003)), # per
                    Dropout(0.4), #dropout rate
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.003))
])
```

```python
# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 80ms/step - accuracy: 0.5895 - loss: 3.0709 - recall_16: 0.6885 - val_accuracy: 0.7216 - val_l
oss: 0.9239 - val_recall_16: 1.0000
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.7256 - loss: 1.0071 - recall_16: 0.9044 - val_accuracy: 0.8770 - val_lo
ss: 0.8440 - val_recall_16: 0.9669
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.7825 - loss: 0.8421 - recall_16: 0.9192 - val_accuracy: 0.8967 - val_lo
ss: 0.6781 - val_recall_16: 0.9550
Epoch 4/8
14/14 - 0s - 16ms/step - accuracy: 0.8195 - loss: 0.6989 - recall_16: 0.9516 - val_accuracy: 0.8224 - val_l
oss: 0.6327 - val_recall_16: 0.9941
Epoch 5/8
14/14 - 0s - 15ms/step - accuracy: 0.8343 - loss: 0.5964 - recall_16: 0.9578 - val_accuracy: 0.9078 - val_l
oss: 0.4877 - val_recall_16: 0.9870
Epoch 6/8
14/14 - 0s - 10ms/step - accuracy: 0.8355 - loss: 0.5342 - recall_16: 0.9575 - val_accuracy: 0.9360 - val_l
oss: 0.4301 - val_recall_16: 0.9858
Epoch 7/8
14/14 - 0s - 9ms/step - accuracy: 0.8614 - loss: 0.4762 - recall_16: 0.9567 - val_accuracy: 0.9394 - val_lo
ss: 0.3604 - val_recall_16: 0.9728
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.8964 - loss: 0.4221 - recall_16: 0.9485 - val_accuracy: 0.8728 - val_lo
ss: 0.4067 - val_recall_16: 0.9941
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - accuracy: 0.8791 - loss: 0.3968 - recall_16: 0.9949
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 771us/step - accuracy: 0.8859 - loss: 0.4038 - recall_16: 0.9973
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 773us/step - accuracy: 0.8694 - loss: 0.4164 - recall_16: 0.9940
```

Out[ ]: [0.4066769480705261, 0.8727583289146423, 0.9940828680992126]

15) Combination (L2=0.004, Dropout = 0.5)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.004)), # per
                    Dropout(0.5), #dropout rate
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.004))
])

# loss and optimizer
```

```python
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 75ms/step - accuracy: 0.6502 - loss: 2.9166 - recall_17: 0.7752 - val_accuracy: 0.7190 - val_l
oss: 1.0659 - val_recall_17: 0.6201
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.7697 - loss: 0.9630 - recall_17: 0.9500 - val_accuracy: 0.8198 - val_lo
ss: 0.8606 - val_recall_17: 0.9941
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.8326 - loss: 0.8273 - recall_17: 0.9500 - val_accuracy: 0.8736 - val_lo
ss: 0.7180 - val_recall_17: 0.9893
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.8500 - loss: 0.7018 - recall_17: 0.9512 - val_accuracy: 0.9146 - val_lo
ss: 0.6156 - val_recall_17: 0.9219
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.8748 - loss: 0.5868 - recall_17: 0.9469 - val_accuracy: 0.9351 - val_lo
ss: 0.4872 - val_recall_17: 0.9728
Epoch 6/8
14/14 - 0s - 7ms/step - accuracy: 0.8813 - loss: 0.5241 - recall_17: 0.9450 - val_accuracy: 0.8745 - val_lo
ss: 0.4753 - val_recall_17: 0.9941
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.8785 - loss: 0.4877 - recall_17: 0.9422 - val_accuracy: 0.9419 - val_lo
ss: 0.3830 - val_recall_17: 0.9586
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.8947 - loss: 0.4309 - recall_17: 0.9418 - val_accuracy: 0.9419 - val_lo
ss: 0.3641 - val_recall_17: 0.9787
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 754us/step - accuracy: 0.9312 - loss: 0.3590 - recall_17: 0.9729
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 906us/step - accuracy: 0.9306 - loss: 0.3691 - recall_17: 0.9690
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 914us/step - accuracy: 0.9448 - loss: 0.3725 - recall_17: 0.9777
```

Out[ ]: [0.364070862531662, 0.9419299960136414, 0.9786982536315918]

16) Combination (L2 = 0.005, Dropout = 0.6)

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.005)), # per
                    Dropout(0.6), #dropout rate
                    Dense(1, activation = 'sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.005))
])


# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(
    x_train, y_train,
    epochs=8,
    batch_size=256,
    validation_data=(x_cv, y_cv),  # Adding validation data
    verbose=2,
)
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 1s - 68ms/step - accuracy: 0.6561 - loss: 2.1460 - recall_18: 0.8263 - val_accuracy: 0.7216 - val_l
oss: 1.1284 - val_recall_18: 1.0000
Epoch 2/8
14/14 - 0s - 7ms/step - accuracy: 0.7367 - loss: 1.1371 - recall_18: 0.9711 - val_accuracy: 0.8770 - val_lo
ss: 1.0098 - val_recall_18: 0.9763
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.7805 - loss: 0.9720 - recall_18: 0.9344 - val_accuracy: 0.9086 - val_lo
ss: 0.8290 - val_recall_18: 0.9467
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.8118 - loss: 0.7972 - recall_18: 0.9301 - val_accuracy: 0.8292 - val_lo
ss: 0.6663 - val_recall_18: 0.9941
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.8135 - loss: 0.6701 - recall_18: 0.9536 - val_accuracy: 0.8898 - val_lo
ss: 0.5688 - val_recall_18: 0.9929
Epoch 6/8
14/14 - 0s - 7ms/step - accuracy: 0.8124 - loss: 0.5919 - recall_18: 0.9539 - val_accuracy: 0.9086 - val_lo
ss: 0.4550 - val_recall_18: 0.9929
Epoch 7/8
14/14 - 0s - 10ms/step - accuracy: 0.8127 - loss: 0.5338 - recall_18: 0.9352 - val_accuracy: 0.9411 - val_l
oss: 0.4040 - val_recall_18: 0.9858
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.8184 - loss: 0.5146 - recall_18: 0.9188 - val_accuracy: 0.8454 - val_lo
ss: 0.4375 - val_recall_18: 0.9941
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 714us/step - accuracy: 0.8620 - loss: 0.4221 - recall_18: 0.9963
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 740us/step - accuracy: 0.8612 - loss: 0.4296 - recall_18: 0.9976
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 744us/step - accuracy: 0.8413 - loss: 0.4441 - recall_18: 0.9940
```

Out[ ]: [0.43752163648605347, 0.8454312682151794, 0.9940828680992126]

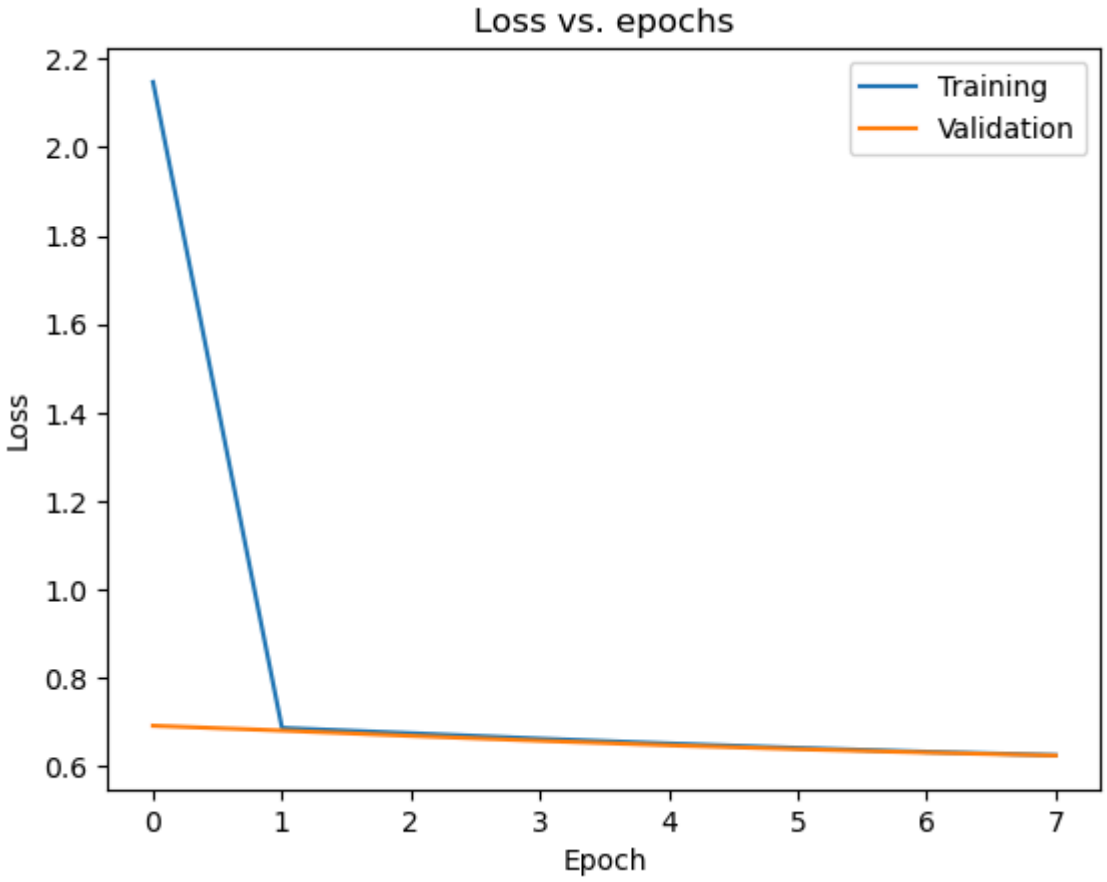
17) Early Stop

In [ ]:
```python
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation = 'relu'),
                    Dense(1, activation = 'sigmoid')
])

# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
```

```
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0.01, patience=5)

history = model.fit(x_train, y_train, epochs = 8, batch_size = 256, verbose = 2, validation_data=(x_cv, y_
                    callbacks=[early_stopping])
```

```
Epoch 1/8
14/14 - 1s - 60ms/step - accuracy: 0.4381 - loss: 2.1469 - recall_19: 0.3649 - val_accuracy: 0.7216 - val_l
oss: 0.6922 - val_recall_19: 1.0000
Epoch 2/8
14/14 - 1s - 38ms/step - accuracy: 0.7293 - loss: 0.6873 - recall_19: 1.0000 - val_accuracy: 0.7216 - val_l
oss: 0.6814 - val_recall_19: 1.0000
Epoch 3/8
14/14 - 0s - 11ms/step - accuracy: 0.7293 - loss: 0.6752 - recall_19: 1.0000 - val_accuracy: 0.7216 - val_l
oss: 0.6694 - val_recall_19: 1.0000
Epoch 4/8
14/14 - 0s - 9ms/step - accuracy: 0.7293 - loss: 0.6630 - recall_19: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.6583 - val_recall_19: 1.0000
Epoch 5/8
14/14 - 0s - 9ms/step - accuracy: 0.7293 - loss: 0.6521 - recall_19: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.6481 - val_recall_19: 1.0000
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.7293 - loss: 0.6421 - recall_19: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.6394 - val_recall_19: 1.0000
Epoch 7/8
14/14 - 0s - 8ms/step - accuracy: 0.7293 - loss: 0.6333 - recall_19: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.6317 - val_recall_19: 1.0000
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.7293 - loss: 0.6258 - recall_19: 1.0000 - val_accuracy: 0.7216 - val_lo
ss: 0.6249 - val_recall_19: 1.0000
```

In [ ]:
```
# Plot the training and validation loss

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [ ]:
```
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
model.evaluate(x_cv, y_cv)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 716us/step - accuracy: 0.7382 - loss: 0.6184 - recall_19: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 707us/step - accuracy: 0.7420 - loss: 0.6169 - recall_19: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 740us/step - accuracy: 0.7253 - loss: 0.6235 - recall_19: 1.0000
```

Out[ ]:
```
[0.6249336004257202, 0.7216054797172546, 1.0]
```

hw5 start

Custome Weights and bias initializers

In [ ]:
```
import tensorflow.keras.backend as K
model = Sequential([
                    Flatten(input_shape = (64, 64, 1)),
                    Dense(64, activation='relu', kernel_initializer='random_uniform', bias_initializer='ze
                    Dense(1, activation='sigmoid', kernel_initializer='he_uniform', bias_initializer='ones
])
model.summary()
model.weights
```

Model: "sequential_20"

Layer (type)	Output Shape	Param #
flatten_20 (Flatten)	(None, 4096)	0
dense_39 (Dense)	(None, 64)	262,208
dense_40 (Dense)	(None, 1)	65

Total params: 262,273 (1.00 MB)

Trainable params: 262,273 (1.00 MB)

Non-trainable params: 0 (0.00 B)

Out[ ]:
```
[<KerasVariable shape=(4096, 64), dtype=float32, path=sequential_20/dense_39/kernel>,
 <KerasVariable shape=(64,), dtype=float32, path=sequential_20/dense_39/bias>,
 <KerasVariable shape=(64, 1), dtype=float32, path=sequential_20/dense_40/kernel>,
 <KerasVariable shape=(1,), dtype=float32, path=sequential_20/dense_40/bias>]
```

In [ ]:
```
model = Sequential([
                    Flatten(input_shape = (64, 64, 1))
])

model.add(Dense(64,
```

```
                kernel_initializer=tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05),
                bias_initializer=tf.keras.initializers.Constant(value=0.4),
                activation='relu'))

model.add(Dense(1,
                kernel_initializer=tf.keras.initializers.HeUniform(seed=None),
                bias_initializer=tf.keras.initializers.Constant(value=0.4),
                activation='sigmoid'))
```

In [ ]:
```
model.summary()
```

Model: "sequential_21"

Layer (type)	Output Shape	Param #
flatten_21 (Flatten)	(None, 4096)	0
dense_41 (Dense)	(None, 64)	262,208
dense_42 (Dense)	(None, 1)	65

Total params: 262,273 (1.00 MB)

Trainable params: 262,273 (1.00 MB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
model.weights
```

Out[ ]:
```
[<KerasVariable shape=(4096, 64), dtype=float32, path=sequential_21/dense_41/kernel>,
 <KerasVariable shape=(64,), dtype=float32, path=sequential_21/dense_41/bias>,
 <KerasVariable shape=(64, 1), dtype=float32, path=sequential_21/dense_42/kernel>,
 <KerasVariable shape=(1,), dtype=float32, path=sequential_21/dense_42/bias>]
```

In [ ]:
```
def my_init(shape, dtype=None):
    return K.random_normal(shape, dtype=dtype)

model.add(Dense(1, kernel_initializer=my_init))
```

In [ ]:
```
model.summary()
```

Model: "sequential_21"

Layer (type)	Output Shape	Param #
flatten_21 (Flatten)	(None, 4096)	0
dense_41 (Dense)	(None, 64)	262,208
dense_42 (Dense)	(None, 1)	65
dense_43 (Dense)	(None, 1)	2

Total params: 262,275 (1.00 MB)

Trainable params: 262,275 (1.00 MB)

Non-trainable params: 0 (0.00 B)

Batch normalization

```
    momentum=0.99, # default is 0.99
    epsilon=0.001, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.05), # default is
    beta_initializer='zeros'
    gamma_initializer=tf.keras.initializers.Constant(value=0.9)
```

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout

model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
    Dense(64, activation="relu"),
    BatchNormalization(),  # <- Batch normalization layer 1
    Dense(256, activation='relu'),
    BatchNormalization(),  # <- Batch normalization layer 32
    Dense(1, activation='sigmoid')
])

model.add(tf.keras.layers.BatchNormalization(
```

```
    momentum=0.99, # default is 0.99
    epsilon=0.001, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.05), # default is beta_initiali
    gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is gamma_initializer='ones'
))

# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(x_train, y_train, epochs = 8, batch_size = 256, verbose = 2, validation_data=(x_cv, y_

model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 2s - 153ms/step - accuracy: 0.6689 - loss: 4.1000 - recall_20: 0.5831 - val_accuracy: 0.2784 - val_
loss: 11.6309 - val_recall_20: 0.0000e+00
Epoch 2/8
14/14 - 0s - 9ms/step - accuracy: 0.8221 - loss: 1.9759 - recall_20: 0.7810 - val_accuracy: 0.7319 - val_lo
ss: 3.3570 - val_recall_20: 1.0000
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.8551 - loss: 1.5196 - recall_20: 0.8431 - val_accuracy: 0.7259 - val_lo
ss: 4.2472 - val_recall_20: 1.0000
Epoch 4/8
14/14 - 0s - 9ms/step - accuracy: 0.8804 - loss: 1.1717 - recall_20: 0.8962 - val_accuracy: 0.2810 - val_lo
ss: 11.5099 - val_recall_20: 0.0036
Epoch 5/8
14/14 - 0s - 9ms/step - accuracy: 0.8673 - loss: 1.2001 - recall_20: 0.8985 - val_accuracy: 0.7233 - val_lo
ss: 4.3616 - val_recall_20: 1.0000
Epoch 6/8
14/14 - 0s - 10ms/step - accuracy: 0.9081 - loss: 0.5805 - recall_20: 0.9594 - val_accuracy: 0.7216 - val_l
oss: 4.4383 - val_recall_20: 1.0000
Epoch 7/8
14/14 - 0s - 13ms/step - accuracy: 0.9106 - loss: 0.5660 - recall_20: 0.9582 - val_accuracy: 0.7242 - val_l
oss: 4.3020 - val_recall_20: 1.0000
Epoch 8/8
14/14 - 0s - 14ms/step - accuracy: 0.9266 - loss: 0.4453 - recall_20: 0.9641 - val_accuracy: 0.8173 - val_l
oss: 2.3028 - val_recall_20: 0.9953
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 954us/step - accuracy: 0.8196 - loss: 2.2529 - recall_20: 0.9957
```

Out[ ]: [2.302755355834961, 0.8172501921653748, 0.9952662587165833]

In [ ]:
```
model.evaluate(x_test, y_test)
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 926us/step - accuracy: 0.8314 - loss: 1.9782 - recall_20: 0.9929
```

Out[ ]: [1.9685423374176025, 0.8336177468299866, 0.994226336479187]

Batch Normalization

```
momentum=0.98, # default is 0.99
epsilon=0.002, #default is 0.001
axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.06), # default is
beta_initializer='zeros'
gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is
gamma_initializer='ones'
```

In [ ]:
```
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
    Dense(64, activation="relu"),
    BatchNormalization(),  # <- Batch normalization layer 1
    Dense(256, activation='relu'),
    BatchNormalization(),  # <- Batch normalization layer 32
    Dense(1, activation='sigmoid')
])

model.add(tf.keras.layers.BatchNormalization(
    momentum=0.98, # default is 0.99
    epsilon=0.002, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.06), # default is beta_initiali
    gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is gamma_initializer='ones'
))

# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])
```

```
history = model.fit(x_train, y_train, epochs = 8, batch_size = 256, verbose = 2, validation_data=(x_cv, y_

model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 3s - 193ms/step - accuracy: 0.6744 - loss: 3.6784 - recall_21: 0.6066 - val_accuracy: 0.2784 - val_
loss: 11.5368 - val_recall_21: 0.0000e+00
Epoch 2/8
14/14 - 0s - 11ms/step - accuracy: 0.7672 - loss: 2.5863 - recall_21: 0.7260 - val_accuracy: 0.7293 - val_l
oss: 4.0956 - val_recall_21: 1.0000
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.8158 - loss: 2.0471 - recall_21: 0.8091 - val_accuracy: 0.8437 - val_lo
ss: 1.7204 - val_recall_21: 0.9550
Epoch 4/8
14/14 - 0s - 8ms/step - accuracy: 0.8824 - loss: 1.1899 - recall_21: 0.8798 - val_accuracy: 0.7216 - val_lo
ss: 4.4383 - val_recall_21: 1.0000
Epoch 5/8
14/14 - 0s - 8ms/step - accuracy: 0.7603 - loss: 2.9104 - recall_21: 0.7693 - val_accuracy: 0.7216 - val_lo
ss: 4.4383 - val_recall_21: 1.0000
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.8599 - loss: 1.2319 - recall_21: 0.8907 - val_accuracy: 0.7216 - val_lo
ss: 0.9996 - val_recall_21: 1.0000
Epoch 7/8
14/14 - 0s - 8ms/step - accuracy: 0.9112 - loss: 0.7091 - recall_21: 0.9411 - val_accuracy: 0.8933 - val_lo
ss: 0.4304 - val_recall_21: 0.9657
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.8901 - loss: 0.7256 - recall_21: 0.9399 - val_accuracy: 0.8924 - val_lo
ss: 0.4344 - val_recall_21: 0.9680
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - accuracy: 0.8917 - loss: 0.4880 - recall_21: 0.9642
```

Out[ ]: [0.43441614508628845, 0.8923996686935425, 0.9680473208427429]

Batch Normalization

```
momentum=0.97, # default is 0.99
epsilon=0.003, #default is 0.001
axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.07), # default is
beta_initializer='zeros'
```

```
    gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is
    gamma_initializer='ones'
```

In [ ]:

```
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
    Dense(64, activation="relu"),
    BatchNormalization(),  # <- Batch normalization layer 1
    Dense(256, activation='relu'),
    BatchNormalization(),  # <- Batch normalization layer 32
    Dense(1, activation='sigmoid')
])

model.add(tf.keras.layers.BatchNormalization(
    momentum=0.97, # default is 0.99
    epsilon=0.003, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.07), # default is beta_initiali
    gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is gamma_initializer='ones'
))

# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(x_train, y_train, epochs = 8, batch_size = 256, verbose = 2, validation_data=(x_cv, y_
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 2s - 112ms/step - accuracy: 0.7179 - loss: 3.3619 - recall_22: 0.6468 - val_accuracy: 0.7950 - val_
loss: 0.5643 - val_recall_22: 0.9964
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.8796 - loss: 1.3805 - recall_22: 0.8528 - val_accuracy: 0.9061 - val_lo
ss: 0.6083 - val_recall_22: 0.9112
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.8241 - loss: 1.8636 - recall_22: 0.8220 - val_accuracy: 0.2784 - val_lo
ss: 11.6309 - val_recall_22: 0.0000e+00
Epoch 4/8
14/14 - 0s - 9ms/step - accuracy: 0.8133 - loss: 1.8567 - recall_22: 0.8368 - val_accuracy: 0.2784 - val_lo
ss: 11.5815 - val_recall_22: 0.0000e+00
Epoch 5/8
14/14 - 0s - 8ms/step - accuracy: 0.8711 - loss: 1.1311 - recall_22: 0.8977 - val_accuracy: 0.8531 - val_lo
ss: 1.5858 - val_recall_22: 0.8260
Epoch 6/8
14/14 - 0s - 9ms/step - accuracy: 0.9026 - loss: 0.7570 - recall_22: 0.9301 - val_accuracy: 0.8847 - val_lo
ss: 0.3710 - val_recall_22: 0.9870
Epoch 7/8
14/14 - 0s - 11ms/step - accuracy: 0.9098 - loss: 0.6710 - recall_22: 0.9450 - val_accuracy: 0.4876 - val_l
oss: 6.3399 - val_recall_22: 0.2935
Epoch 8/8
14/14 - 0s - 13ms/step - accuracy: 0.8878 - loss: 0.7979 - recall_22: 0.9379 - val_accuracy: 0.4278 - val_l
oss: 7.4314 - val_recall_22: 0.2083
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4141 - loss: 7.3998 - recall_22: 0.1933
```

Out[ ]: [7.431393146514893, 0.427839457988739, 0.20828402042388916]

Batch Normalization

```
momentum=0.96, # default is 0.99
epsilon=0.004, #default is 0.001
axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.08), # default is
beta_initializer='zeros'
gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is
gamma_initializer='ones'
```

In [ ]:
```
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
```

```
    Dense(64, activation="relu"),
    BatchNormalization(),  # <- Batch normalization layer 1
    Dense(256, activation='relu'),
    BatchNormalization(),  # <- Batch normalization layer 32
    Dense(1, activation='sigmoid')
])

model.add(tf.keras.layers.BatchNormalization(
    momentum=0.96, # default is 0.99
    epsilon=0.004, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.08), # default is beta_initiali
    gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is gamma_initializer='ones'
))

# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(x_train, y_train, epochs = 8, batch_size = 256, verbose = 2, validation_data=(x_cv, y_
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 2s - 115ms/step - accuracy: 0.6596 - loss: 3.9039 - recall_24: 0.5777 - val_accuracy: 0.7216 - val_
loss: 0.7754 - val_recall_24: 1.0000
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.7643 - loss: 2.8523 - recall_24: 0.7279 - val_accuracy: 0.7225 - val_lo
ss: 0.9846 - val_recall_24: 1.0000
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.8192 - loss: 1.9890 - recall_24: 0.8044 - val_accuracy: 0.7583 - val_lo
ss: 2.5076 - val_recall_24: 0.9953
Epoch 4/8
14/14 - 0s - 8ms/step - accuracy: 0.8876 - loss: 0.9444 - recall_24: 0.9126 - val_accuracy: 0.7216 - val_lo
ss: 4.4383 - val_recall_24: 1.0000
Epoch 5/8
14/14 - 0s - 8ms/step - accuracy: 0.8597 - loss: 1.4545 - recall_24: 0.8864 - val_accuracy: 0.2784 - val_lo
ss: 11.6309 - val_recall_24: 0.0000e+00
Epoch 6/8
14/14 - 0s - 7ms/step - accuracy: 0.8013 - loss: 1.5077 - recall_24: 0.8677 - val_accuracy: 0.4193 - val_lo
ss: 8.5588 - val_recall_24: 0.1953
Epoch 7/8
14/14 - 0s - 8ms/step - accuracy: 0.8799 - loss: 0.6908 - recall_24: 0.9450 - val_accuracy: 0.7310 - val_lo
ss: 0.6772 - val_recall_24: 1.0000
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.9081 - loss: 0.4827 - recall_24: 0.9660 - val_accuracy: 0.7225 - val_lo
ss: 0.7065 - val_recall_24: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 856us/step - accuracy: 0.7258 - loss: 0.6992 - recall_24: 1.0000
```

Out[ ]: `[0.7065058350563049, 0.7224594354629517, 1.0]`

Batch Normalization

```
momentum=0.95, # default is 0.99
epsilon=0.005, #default is 0.001
axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.09), # default is
beta_initializer='zeros'
gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is
gamma_initializer='ones'
```

In [ ]:
```
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
```

```
    Dense(64, activation="relu"),
    BatchNormalization(),  # <- Batch normalization layer 1
    Dense(256, activation='relu'),
    BatchNormalization(),  # <- Batch normalization layer 32
    Dense(1, activation='sigmoid')
])

model.add(tf.keras.layers.BatchNormalization(
    momentum=0.95, # default is 0.99
    epsilon=0.005, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.09), # default is beta_initiali
    gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is gamma_initializer='ones'
))


# loss and optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(x_train, y_train, epochs = 8, batch_size = 256, verbose = 2, validation_data=(x_cv, y_
model.evaluate(x_cv, y_cv)
```

```
Epoch 1/8
14/14 - 2s - 120ms/step - accuracy: 0.5981 - loss: 4.6793 - recall_25: 0.4973 - val_accuracy: 0.2784 - val_
loss: 11.6309 - val_recall_25: 0.0000e+00
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.6459 - loss: 3.8882 - recall_25: 0.5578 - val_accuracy: 0.2784 - val_lo
ss: 11.6309 - val_recall_25: 0.0000e+00
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.6900 - loss: 3.1693 - recall_25: 0.6120 - val_accuracy: 0.2784 - val_lo
ss: 11.6309 - val_recall_25: 0.0000e+00
Epoch 4/8
14/14 - 0s - 25ms/step - accuracy: 0.6624 - loss: 3.7597 - recall_25: 0.5999 - val_accuracy: 0.2784 - val_l
oss: 11.6309 - val_recall_25: 0.0000e+00
Epoch 5/8
14/14 - 0s - 9ms/step - accuracy: 0.6576 - loss: 3.7388 - recall_25: 0.6163 - val_accuracy: 0.2784 - val_lo
ss: 11.6309 - val_recall_25: 0.0000e+00
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.6897 - loss: 3.2806 - recall_25: 0.6561 - val_accuracy: 0.2784 - val_lo
ss: 11.6309 - val_recall_25: 0.0000e+00
Epoch 7/8
14/14 - 0s - 8ms/step - accuracy: 0.6647 - loss: 3.3384 - recall_25: 0.6553 - val_accuracy: 0.2784 - val_lo
ss: 11.6309 - val_recall_25: 0.0000e+00
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.7017 - loss: 2.7985 - recall_25: 0.6827 - val_accuracy: 0.2792 - val_lo
ss: 11.5436 - val_recall_25: 0.0012
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.2750 - loss: 11.6478 - recall_25: 3.4378e-04
```

Out[ ]: [11.54360294342041, 0.27924850583076477, 0.001183431944809854]

Model with batch normalization and dropout (Default)

In [ ]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout

model = Sequential([
    # Ensure the input images are flattened
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
    Dense(64, activation="relu"),
    BatchNormalization(),  # Normalize after the activation
    Dropout(0.5),
    Dense(256, activation='relu'),
```

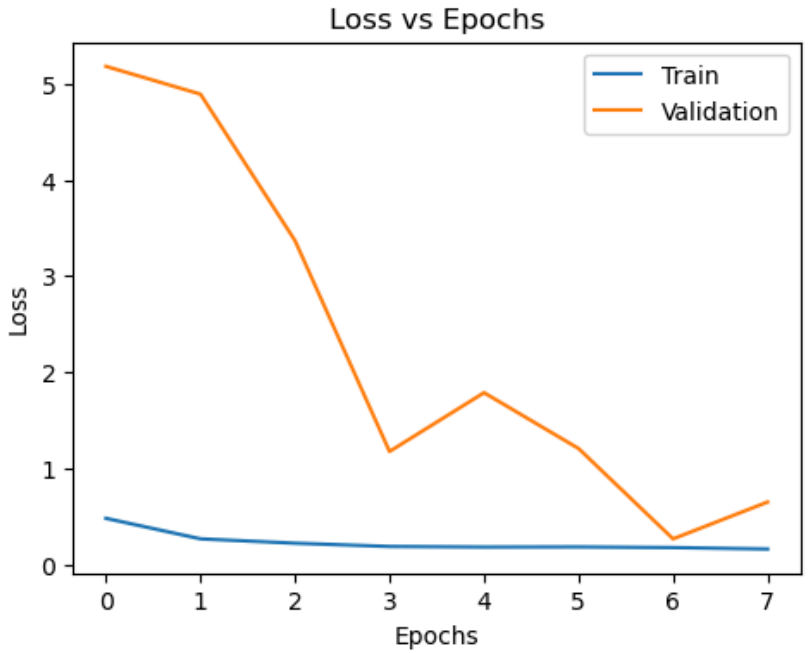
```
    BatchNormalization(),  # Normalize after the activation
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # This outputs the probability for one class
])

from tensorflow.keras.metrics import Recall

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
model.evaluate(x_cv, y_cv)
```

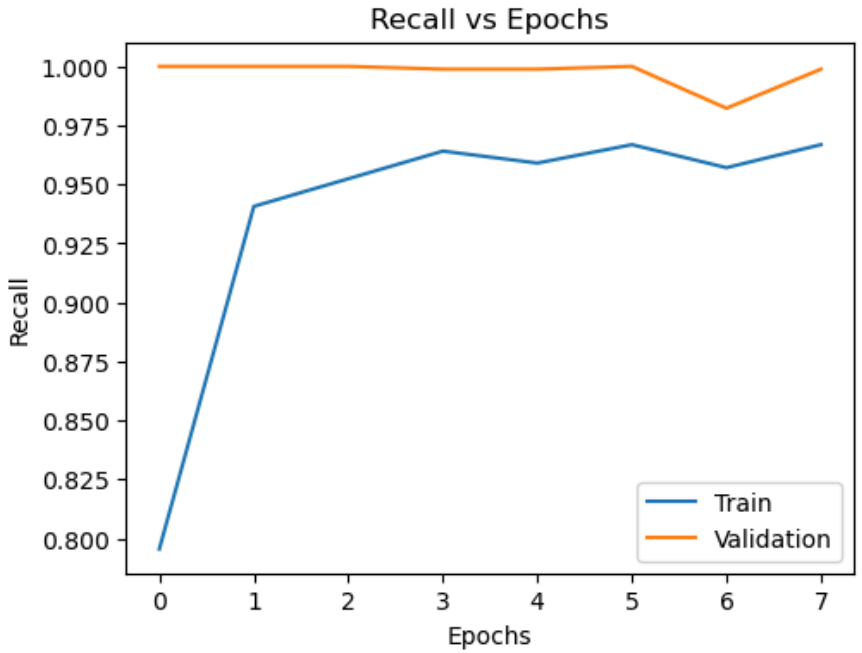
```
Epoch 1/8
14/14 - 2s - 110ms/step - accuracy: 0.8044 - loss: 0.4803 - recall_28: 0.7955 - val_accuracy: 0.7216 - val_
loss: 5.1790 - val_recall_28: 1.0000
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.9058 - loss: 0.2670 - recall_28: 0.9407 - val_accuracy: 0.7216 - val_lo
ss: 4.8901 - val_recall_28: 1.0000
Epoch 3/8
14/14 - 0s - 9ms/step - accuracy: 0.9132 - loss: 0.2218 - recall_28: 0.9524 - val_accuracy: 0.7216 - val_lo
ss: 3.3677 - val_recall_28: 1.0000
Epoch 4/8
14/14 - 0s - 10ms/step - accuracy: 0.9325 - loss: 0.1878 - recall_28: 0.9641 - val_accuracy: 0.7541 - val_l
oss: 1.1753 - val_recall_28: 0.9988
Epoch 5/8
14/14 - 0s - 11ms/step - accuracy: 0.9357 - loss: 0.1819 - recall_28: 0.9590 - val_accuracy: 0.7344 - val_l
oss: 1.7866 - val_recall_28: 0.9988
Epoch 6/8
14/14 - 0s - 11ms/step - accuracy: 0.9354 - loss: 0.1832 - recall_28: 0.9668 - val_accuracy: 0.7498 - val_l
oss: 1.2061 - val_recall_28: 1.0000
Epoch 7/8
14/14 - 0s - 10ms/step - accuracy: 0.9331 - loss: 0.1763 - recall_28: 0.9571 - val_accuracy: 0.9018 - val_l
oss: 0.2669 - val_recall_28: 0.9822
Epoch 8/8
14/14 - 0s - 10ms/step - accuracy: 0.9434 - loss: 0.1613 - recall_28: 0.9668 - val_accuracy: 0.8079 - val_l
oss: 0.6499 - val_recall_28: 0.9988
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 979us/step - accuracy: 0.8076 - loss: 0.6765 - recall_28: 0.9987
```

Out[ ]: [0.6498734951019287, 0.807856559753418, 0.9988165497779846]

In [ ]:
```
# Plot the learning curves

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

frame = pd.DataFrame(history.history)
epochs = np.arange(len(frame))

fig = plt.figure(figsize=(12,4))

# Loss plot
ax = fig.add_subplot(121)
ax.plot(epochs, frame['loss'], label="Train")
ax.plot(epochs, frame['val_loss'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.set_title("Loss vs Epochs")
ax.legend()

# Accuracy plot
ax = fig.add_subplot(122)
ax.plot(epochs, frame['recall_28'], label="Train")
ax.plot(epochs, frame['val_recall_28'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Recall")
ax.set_title("Recall vs Epochs")
ax.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x2fb8cbf10>
```





Model with batch normalization and dropout (change dropout rate, batch normalization)

In [ ]:

```
model = Sequential([
    # Ensure the input images are flattened
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
    Dense(64, activation="relu"),
    BatchNormalization(),  # <- Batch normalization layer 1
    Dropout(0.4),
    BatchNormalization(), # <- Batch normalization layer 2
    Dropout(0.4),
    Dense(256, activation='relu'),
    #BatchNormalization(),
    #Dense(1)
])

model.add(tf.keras.layers.BatchNormalization(
    momentum=0.95, # default is 0.99
```

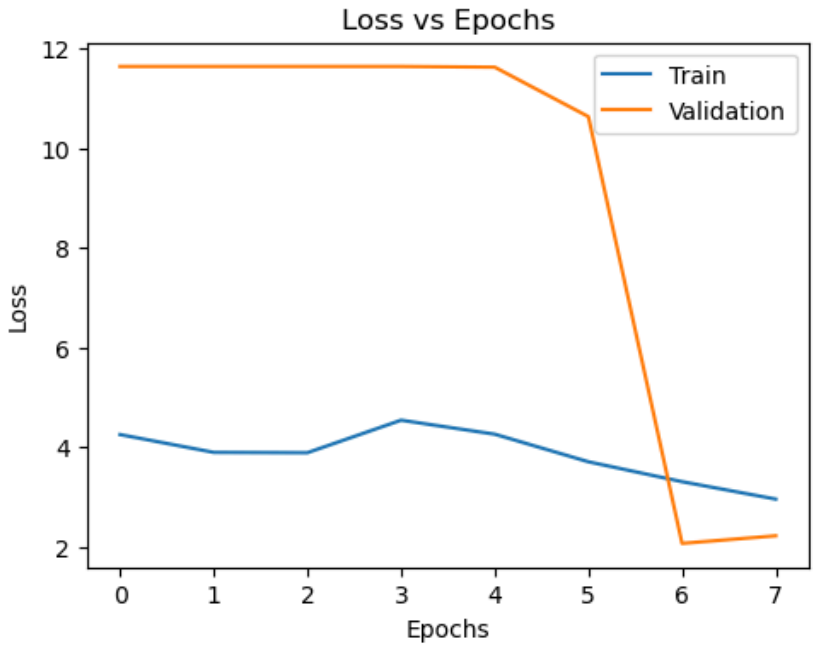
```
    epsilon=0.005, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.05), # default is beta_initiali
    gamma_initializer=tf.keras.initializers.Constant(value=0.9) # default is gamma_initializer='ones'
))

model.add(Dense(1))

from tensorflow.keras.metrics import Recall

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
model.evaluate(x_cv, y_cv)
```

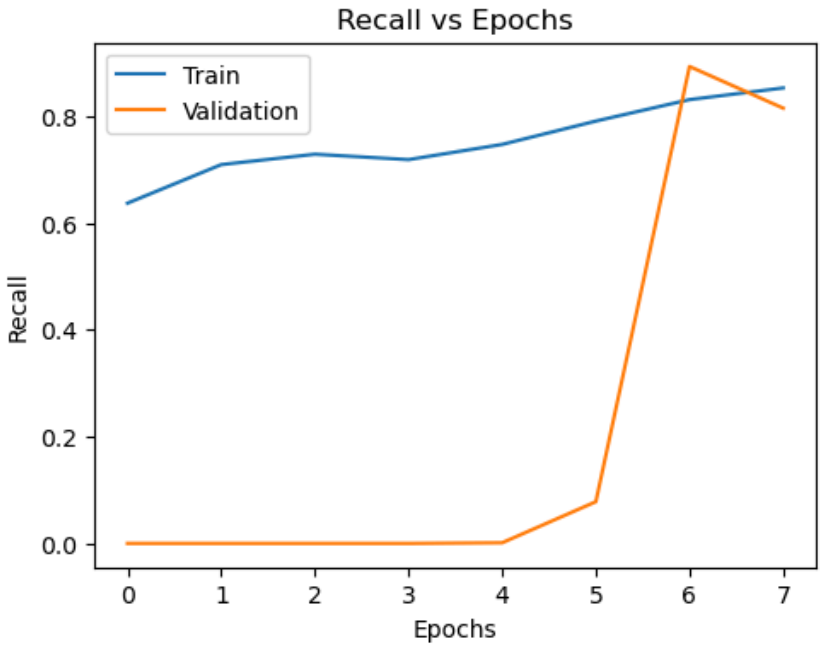
```
Epoch 1/8
14/14 - 3s - 181ms/step - accuracy: 0.6630 - loss: 4.2579 - recall_31: 0.6374 - val_accuracy: 0.2784 - val_
loss: 11.6309 - val_recall_31: 0.0000e+00
Epoch 2/8
14/14 - 0s - 10ms/step - accuracy: 0.7205 - loss: 3.9013 - recall_31: 0.7096 - val_accuracy: 0.2784 - val_l
oss: 11.6309 - val_recall_31: 0.0000e+00
Epoch 3/8
14/14 - 0s - 10ms/step - accuracy: 0.7282 - loss: 3.8942 - recall_31: 0.7291 - val_accuracy: 0.2784 - val_l
oss: 11.6309 - val_recall_31: 0.0000e+00
Epoch 4/8
14/14 - 0s - 13ms/step - accuracy: 0.6951 - loss: 4.5455 - recall_31: 0.7190 - val_accuracy: 0.2784 - val_l
oss: 11.6309 - val_recall_31: 0.0000e+00
Epoch 5/8
14/14 - 0s - 17ms/step - accuracy: 0.7125 - loss: 4.2655 - recall_31: 0.7475 - val_accuracy: 0.2792 - val_l
oss: 11.6171 - val_recall_31: 0.0012
Epoch 6/8
14/14 - 0s - 10ms/step - accuracy: 0.7461 - loss: 3.7129 - recall_31: 0.7912 - val_accuracy: 0.3348 - val_l
oss: 10.6258 - val_recall_31: 0.0781
Epoch 7/8
14/14 - 0s - 9ms/step - accuracy: 0.7743 - loss: 3.3141 - recall_31: 0.8314 - val_accuracy: 0.8523 - val_lo
ss: 2.0780 - val_recall_31: 0.8935
Epoch 8/8
14/14 - 0s - 10ms/step - accuracy: 0.7931 - loss: 2.9653 - recall_31: 0.8532 - val_accuracy: 0.8480 - val_l
oss: 2.2301 - val_recall_31: 0.8154
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8449 - loss: 2.3294 - recall_31: 0.8093
```

Out[ ]:
```
[2.2300636768341064, 0.8479931950569153, 0.8153846263885498]
```

In [ ]:
```
frame = pd.DataFrame(history.history)
epochs = np.arange(len(frame))

fig = plt.figure(figsize=(12,4))

# Loss plot
ax = fig.add_subplot(121)
ax.plot(epochs, frame['loss'], label="Train")
ax.plot(epochs, frame['val_loss'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.set_title("Loss vs Epochs")
ax.legend()

# Accuracy plot
ax = fig.add_subplot(122)
ax.plot(epochs, frame['recall_31'], label="Train")
ax.plot(epochs, frame['val_recall_31'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Recall")
ax.set_title("Recall vs Epochs")
ax.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x2fbb41ad0>
```





In [ ]:

```
model = Sequential([
    # Ensure the input images are flattened
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
    Dense(64, activation="relu"),
    BatchNormalization(),  # <- Batch normalization layer 1
    Dropout(0.5),
    BatchNormalization(), # <- Batch normalization layer 2
    Dropout(0.5),
    Dense(256, activation='relu'),
    #BatchNormalization(),
    #Dense(1)
])

model.add(tf.keras.layers.BatchNormalization(
    momentum=0.90, # default is 0.99
    epsilon=0.010, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.06), # default is beta_initiali
    gamma_initializer=tf.keras.initializers.Constant(value=0.8) # default is gamma_initializer='ones'
))
```

```
model.add(Dense(1))

from tensorflow.keras.metrics import Recall

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
model.evaluate(x_cv, y_cv)
```

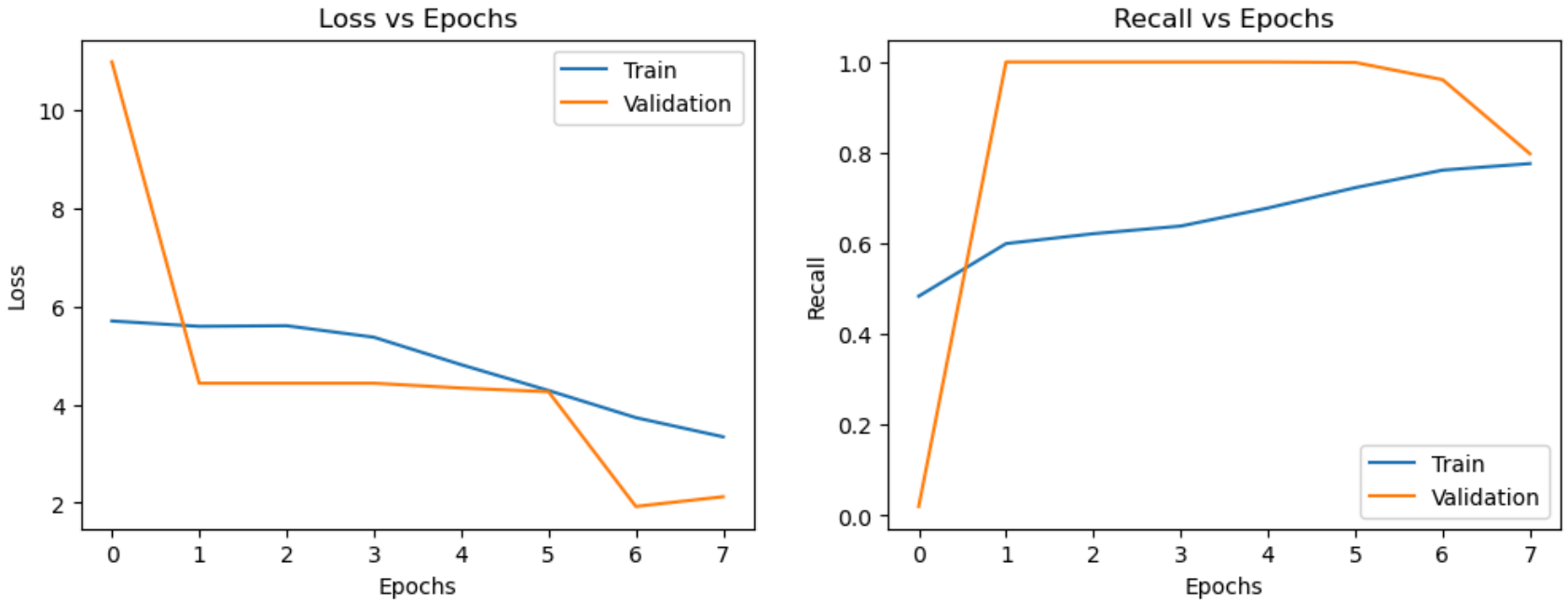
```
Epoch 1/8
14/14 - 2s - 128ms/step - accuracy: 0.4999 - loss: 5.7078 - recall_32: 0.4836 - val_accuracy: 0.2929 - val_
loss: 10.9913 - val_recall_32: 0.0201
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.5702 - loss: 5.5987 - recall_32: 0.5995 - val_accuracy: 0.7216 - val_lo
ss: 4.4383 - val_recall_32: 1.0000
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.5841 - loss: 5.6108 - recall_32: 0.6214 - val_accuracy: 0.7216 - val_lo
ss: 4.4383 - val_recall_32: 1.0000
Epoch 4/8
14/14 - 0s - 8ms/step - accuracy: 0.6043 - loss: 5.3764 - recall_32: 0.6382 - val_accuracy: 0.7216 - val_lo
ss: 4.4383 - val_recall_32: 1.0000
Epoch 5/8
14/14 - 0s - 8ms/step - accuracy: 0.6405 - loss: 4.8157 - recall_32: 0.6780 - val_accuracy: 0.7233 - val_lo
ss: 4.3398 - val_recall_32: 1.0000
Epoch 6/8
14/14 - 0s - 9ms/step - accuracy: 0.6823 - loss: 4.2899 - recall_32: 0.7229 - val_accuracy: 0.7267 - val_lo
ss: 4.2648 - val_recall_32: 0.9988
Epoch 7/8
14/14 - 0s - 8ms/step - accuracy: 0.7131 - loss: 3.7360 - recall_32: 0.7615 - val_accuracy: 0.8574 - val_lo
ss: 1.9247 - val_recall_32: 0.9609
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.7512 - loss: 3.3447 - recall_32: 0.7760 - val_accuracy: 0.8232 - val_lo
ss: 2.1234 - val_recall_32: 0.7976
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 868us/step - accuracy: 0.8049 - loss: 2.2658 - recall_32: 0.7752
```

Out[ ]: [2.123356580734253, 0.8232280015945435, 0.7976331114768982]

In [ ]:
```
frame = pd.DataFrame(history.history)
epochs = np.arange(len(frame))

fig = plt.figure(figsize=(12,4))
```

```
# Loss plot
ax = fig.add_subplot(121)
ax.plot(epochs, frame['loss'], label="Train")
ax.plot(epochs, frame['val_loss'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.set_title("Loss vs Epochs")
ax.legend()

# Accuracy plot
ax = fig.add_subplot(122)
ax.plot(epochs, frame['recall_32'], label="Train")
ax.plot(epochs, frame['val_recall_32'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Recall")
ax.set_title("Recall vs Epochs")
ax.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x2fbd9d550>



In [ ]:

```
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
    Dense(64, activation="relu"),
    BatchNormalization(),  # <- Batch normalization layer 1
    Dropout(0.5),
    BatchNormalization(), # <- Batch normalization layer 2
    Dropout(0.5),
    Dense(256, activation='relu'),
    #BatchNormalization(),
    #Dense(1)
])
model.add(tf.keras.layers.BatchNormalization(
    momentum=0.92, # default is 0.99
    epsilon=0.015, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.07), # default is beta_initiali
    gamma_initializer=tf.keras.initializers.Constant(value=0.7) # default is gamma_initializer='ones'
))
model.add(Dense(1))
from tensorflow.keras.metrics import Recall

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
model.evaluate(x_cv, y_cv)
```

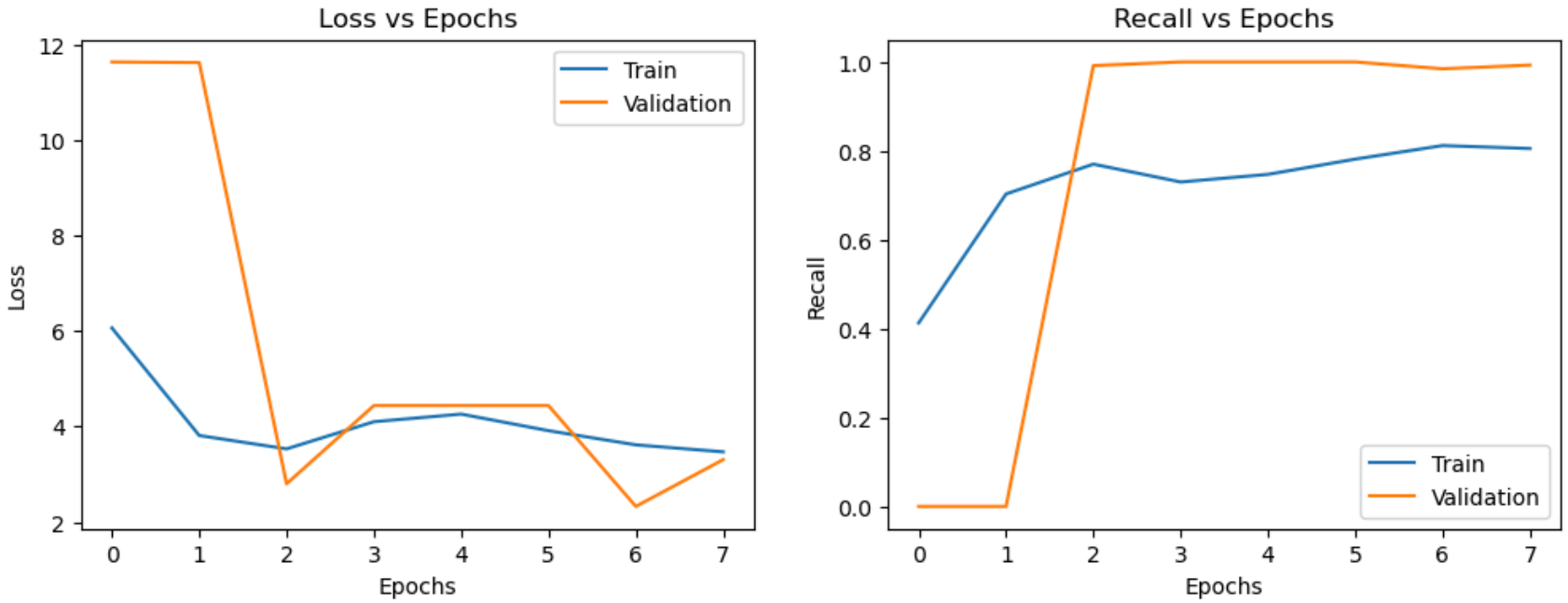
```
Epoch 1/8
14/14 - 2s - 124ms/step - accuracy: 0.5238 - loss: 6.0623 - recall_33: 0.4130 - val_accuracy: 0.2784 - val_
loss: 11.6309 - val_recall_33: 0.0000e+00
Epoch 2/8
14/14 - 0s - 10ms/step - accuracy: 0.6948 - loss: 3.8113 - recall_33: 0.7030 - val_accuracy: 0.2784 - val_l
oss: 11.6179 - val_recall_33: 0.0000e+00
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.7350 - loss: 3.5317 - recall_33: 0.7701 - val_accuracy: 0.8130 - val_lo
ss: 2.8001 - val_recall_33: 0.9917
Epoch 4/8
14/14 - 0s - 8ms/step - accuracy: 0.6943 - loss: 4.1001 - recall_33: 0.7299 - val_accuracy: 0.7216 - val_lo
ss: 4.4383 - val_recall_33: 1.0000
Epoch 5/8
14/14 - 0s - 8ms/step - accuracy: 0.6900 - loss: 4.2599 - recall_33: 0.7471 - val_accuracy: 0.7216 - val_lo
ss: 4.4383 - val_recall_33: 1.0000
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.7171 - loss: 3.9125 - recall_33: 0.7814 - val_accuracy: 0.7216 - val_lo
ss: 4.4383 - val_recall_33: 1.0000
Epoch 7/8
14/14 - 0s - 10ms/step - accuracy: 0.7447 - loss: 3.6160 - recall_33: 0.8119 - val_accuracy: 0.8326 - val_l
oss: 2.3269 - val_recall_33: 0.9846
Epoch 8/8
14/14 - 0s - 11ms/step - accuracy: 0.7501 - loss: 3.4697 - recall_33: 0.8052 - val_accuracy: 0.7805 - val_l
oss: 3.3060 - val_recall_33: 0.9929
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 959us/step - accuracy: 0.7885 - loss: 3.2486 - recall_33: 0.9929
```

Out[ ]:
```
[3.3059511184692383, 0.7805294394493103, 0.9928994178771973]
```

In [ ]:
```python
frame = pd.DataFrame(history.history)
epochs = np.arange(len(frame))

fig = plt.figure(figsize=(12,4))

# Loss plot
ax = fig.add_subplot(121)
ax.plot(epochs, frame['loss'], label="Train")
ax.plot(epochs, frame['val_loss'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.set_title("Loss vs Epochs")
ax.legend()
```

```
# Accuracy plot
ax = fig.add_subplot(122)
ax.plot(epochs, frame['recall_33'], label="Train")
ax.plot(epochs, frame['val_recall_33'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Recall")
ax.set_title("Recall vs Epochs")
ax.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x2fd3cf150>



In [ ]:
```
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # This line flattens the 64x64x1 images to a vector
    Dense(64, activation="relu"),
    BatchNormalization(),  # <- Batch normalization layer 1
    Dropout(0.5),
    BatchNormalization(), # <- Batch normalization layer 2
    Dropout(0.5),
    Dense(256, activation='relu'),
    #BatchNormalization(),
    #Dense(1)
])
```

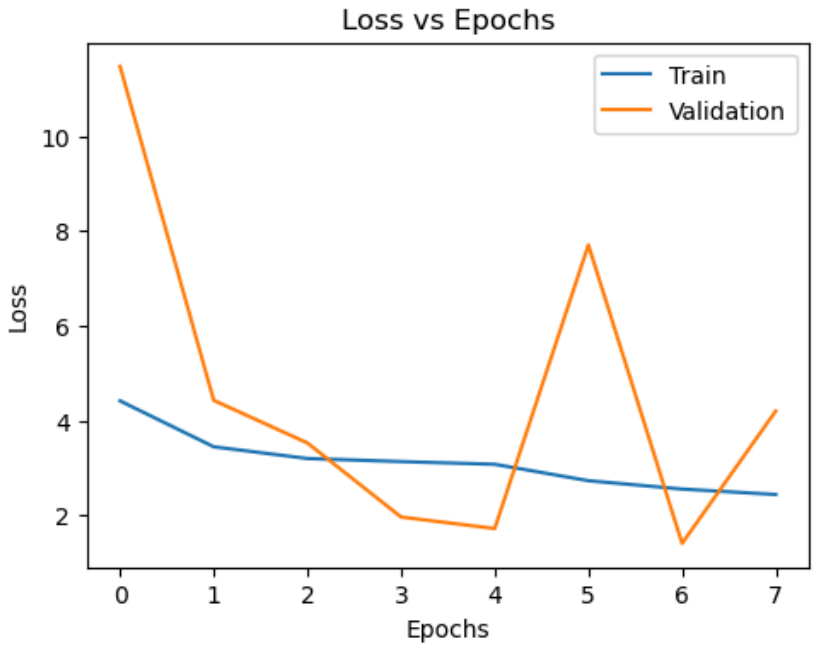
```
model.add(tf.keras.layers.BatchNormalization(
    momentum=0.97, # default is 0.99
    epsilon=0.006, #default is 0.001
    axis = -1, #default is -1 (meaning the channel dimension is the last dimension)
    beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.04), # default is beta_initiali
    gamma_initializer=tf.keras.initializers.Constant(value=0.5) # default is gamma_initializer='ones'
))
model.add(Dense(1))
from tensorflow.keras.metrics import Recall

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.005),
              loss='binary_crossentropy',
              metrics=['accuracy', Recall()])

history = model.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
model.evaluate(x_cv, y_cv)
```

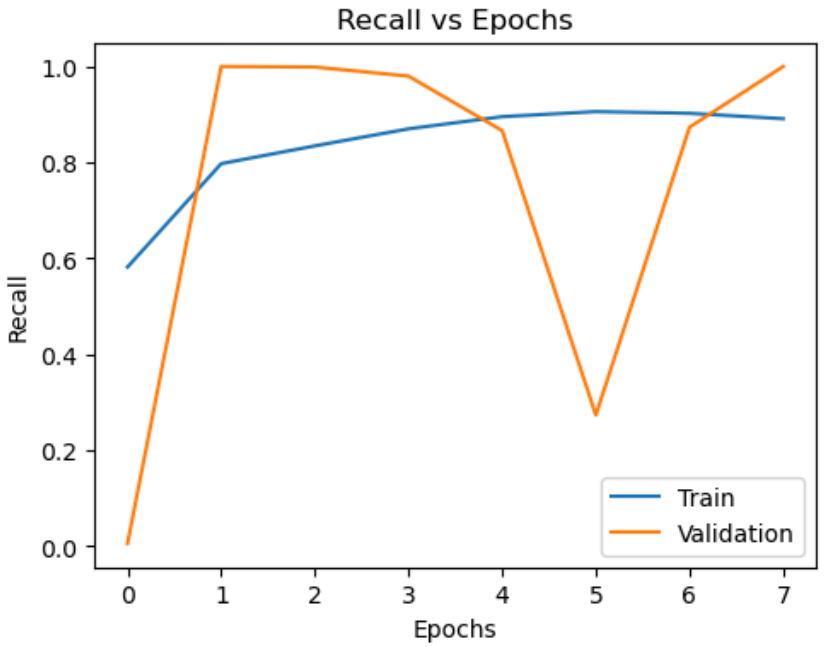
```
Epoch 1/8
14/14 - 2s - 128ms/step - accuracy: 0.5827 - loss: 4.4150 - recall_34: 0.5820 - val_accuracy: 0.2827 - val_
loss: 11.4764 - val_recall_34: 0.0059
Epoch 2/8
14/14 - 0s - 9ms/step - accuracy: 0.7148 - loss: 3.4450 - recall_34: 0.7974 - val_accuracy: 0.7216 - val_lo
ss: 4.4264 - val_recall_34: 1.0000
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.7424 - loss: 3.1977 - recall_34: 0.8345 - val_accuracy: 0.7643 - val_lo
ss: 3.5264 - val_recall_34: 0.9988
Epoch 4/8
14/14 - 0s - 10ms/step - accuracy: 0.7626 - loss: 3.1328 - recall_34: 0.8700 - val_accuracy: 0.8591 - val_l
oss: 1.9620 - val_recall_34: 0.9799
Epoch 5/8
14/14 - 0s - 8ms/step - accuracy: 0.7726 - loss: 3.0758 - recall_34: 0.8954 - val_accuracy: 0.8693 - val_lo
ss: 1.7191 - val_recall_34: 0.8663
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.7979 - loss: 2.7290 - recall_34: 0.9059 - val_accuracy: 0.4757 - val_lo
ss: 7.7090 - val_recall_34: 0.2734
Epoch 7/8
14/14 - 0s - 8ms/step - accuracy: 0.8064 - loss: 2.5527 - recall_34: 0.9024 - val_accuracy: 0.8839 - val_lo
ss: 1.4051 - val_recall_34: 0.8734
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.8144 - loss: 2.4380 - recall_34: 0.8911 - val_accuracy: 0.7310 - val_lo
ss: 4.2003 - val_recall_34: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 922us/step - accuracy: 0.7326 - loss: 4.1681 - recall_34: 1.0000
```

Out[ ]:
```
[4.2002644538879395, 0.7309991717338562, 1.0]
```

In [ ]:
```
frame = pd.DataFrame(history.history)
epochs = np.arange(len(frame))

fig = plt.figure(figsize=(12,4))

# Loss plot
ax = fig.add_subplot(121)
ax.plot(epochs, frame['loss'], label="Train")
ax.plot(epochs, frame['val_loss'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.set_title("Loss vs Epochs")
ax.legend()

# Accuracy plot
ax = fig.add_subplot(122)
ax.plot(epochs, frame['recall_34'], label="Train")
ax.plot(epochs, frame['val_recall_34'], label="Validation")
ax.set_xlabel("Epochs")
ax.set_ylabel("Recall")
ax.set_title("Recall vs Epochs")
ax.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x2fefccb50>
```





Batch Normalization, dropout, and L2 regularization

baseline with L2 regularization

In [ ]:
```
model1 = Sequential([
    Flatten(input_shape=(64, 64, 1)),
    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))
])

model1.compile(
     optimizer = tf.keras.optimizers.Adam(learning_rate=0.005),
     loss = 'binary_crossentropy',
     metrics = ['accuracy', Recall()]
 )
history1 = model1.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
```

```
Epoch 1/8
14/14 - 1s - 70ms/step - accuracy: 0.5346 - loss: 2.7369 - recall_35: 0.5820 - val_accuracy: 0.7216 - val_l
oss: 0.7329 - val_recall_35: 1.0000
Epoch 2/8
14/14 - 0s - 9ms/step - accuracy: 0.7495 - loss: 0.7393 - recall_35: 0.9953 - val_accuracy: 0.7276 - val_lo
ss: 0.6577 - val_recall_35: 1.0000
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.8124 - loss: 0.6094 - recall_35: 0.9883 - val_accuracy: 0.8044 - val_lo
ss: 0.5711 - val_recall_35: 0.9941
Epoch 4/8
14/14 - 0s - 9ms/step - accuracy: 0.8377 - loss: 0.5475 - recall_35: 0.9415 - val_accuracy: 0.8232 - val_lo
ss: 0.5172 - val_recall_35: 0.9941
Epoch 5/8
14/14 - 0s - 14ms/step - accuracy: 0.8876 - loss: 0.4469 - recall_35: 0.9532 - val_accuracy: 0.8907 - val_l
oss: 0.4197 - val_recall_35: 0.9893
Epoch 6/8
14/14 - 0s - 13ms/step - accuracy: 0.9035 - loss: 0.3968 - recall_35: 0.9555 - val_accuracy: 0.9266 - val_l
oss: 0.3534 - val_recall_35: 0.9751
Epoch 7/8
14/14 - 0s - 10ms/step - accuracy: 0.9220 - loss: 0.3454 - recall_35: 0.9649 - val_accuracy: 0.9419 - val_l
oss: 0.3132 - val_recall_35: 0.9692
Epoch 8/8
14/14 - 0s - 9ms/step - accuracy: 0.9314 - loss: 0.3129 - recall_35: 0.9672 - val_accuracy: 0.9342 - val_lo
ss: 0.2992 - val_recall_35: 0.9893
```

L2 regularization + Dropout

In [ ]:

```
model2 = Sequential([
    Flatten(input_shape=(64, 64, 1)),
    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    Dropout(0.2),
    Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))
])

model2.compile(
     optimizer = tf.keras.optimizers.Adam(learning_rate=0.005),
     loss = 'binary_crossentropy',
     metrics = ['accuracy', Recall()]
 )
history2 = model2.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
```

```
Epoch 1/8
14/14 - 1s - 68ms/step - accuracy: 0.6132 - loss: 2.6087 - recall_36: 0.7311 - val_accuracy: 0.7216 - val_l
oss: 0.6820 - val_recall_36: 1.0000
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.7717 - loss: 0.6659 - recall_36: 0.9161 - val_accuracy: 0.8719 - val_lo
ss: 0.5902 - val_recall_36: 0.8982
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.8543 - loss: 0.5524 - recall_36: 0.9571 - val_accuracy: 0.8890 - val_lo
ss: 0.5039 - val_recall_36: 0.9716
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.8841 - loss: 0.4915 - recall_36: 0.9524 - val_accuracy: 0.9137 - val_lo
ss: 0.4369 - val_recall_36: 0.9751
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.8924 - loss: 0.4485 - recall_36: 0.9477 - val_accuracy: 0.8915 - val_lo
ss: 0.4309 - val_recall_36: 0.9941
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.8930 - loss: 0.4296 - recall_36: 0.9536 - val_accuracy: 0.9231 - val_lo
ss: 0.3731 - val_recall_36: 0.9905
Epoch 7/8
14/14 - 0s - 9ms/step - accuracy: 0.9063 - loss: 0.3984 - recall_36: 0.9590 - val_accuracy: 0.9385 - val_lo
ss: 0.3339 - val_recall_36: 0.9503
Epoch 8/8
14/14 - 0s - 10ms/step - accuracy: 0.9192 - loss: 0.3579 - recall_36: 0.9590 - val_accuracy: 0.9488 - val_l
oss: 0.3136 - val_recall_36: 0.9787
```

L2 Regularization + Batch Normilization

In [ ]:

```
model3 = Sequential([
    Flatten(input_shape=(64, 64, 1)),
    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    BatchNormalization(),
    Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))
])
model3.compile(
     optimizer = tf.keras.optimizers.Adam(learning_rate=0.005),
     loss = 'binary_crossentropy',
     metrics = ['accuracy', Recall()]
 )
history3 = model3.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
```

```
Epoch 1/8
14/14 - 1s - 85ms/step - accuracy: 0.8210 - loss: 0.6218 - recall_37: 0.8095 - val_accuracy: 0.7635 - val_l
oss: 1.4499 - val_recall_37: 0.9988
Epoch 2/8
14/14 - 0s - 7ms/step - accuracy: 0.9234 - loss: 0.4693 - recall_37: 0.9469 - val_accuracy: 0.9385 - val_lo
ss: 0.4038 - val_recall_37: 0.9456
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.9294 - loss: 0.4088 - recall_37: 0.9563 - val_accuracy: 0.9394 - val_lo
ss: 0.3682 - val_recall_37: 0.9917
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.9428 - loss: 0.3474 - recall_37: 0.9660 - val_accuracy: 0.9360 - val_lo
ss: 0.3335 - val_recall_37: 0.9302
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.9456 - loss: 0.3034 - recall_37: 0.9711 - val_accuracy: 0.9436 - val_lo
ss: 0.3130 - val_recall_37: 0.9905
Epoch 6/8
14/14 - 0s - 7ms/step - accuracy: 0.9536 - loss: 0.2712 - recall_37: 0.9723 - val_accuracy: 0.9539 - val_lo
ss: 0.2621 - val_recall_37: 0.9858
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.9479 - loss: 0.2576 - recall_37: 0.9770 - val_accuracy: 0.9513 - val_lo
ss: 0.2652 - val_recall_37: 0.9586
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.9499 - loss: 0.2454 - recall_37: 0.9750 - val_accuracy: 0.8164 - val_lo
ss: 0.4916 - val_recall_37: 0.9988
```

L2 Regularization + Batch Normilization + Dropout

In [ ]:

```
model4 = Sequential([
    Flatten(input_shape=(64, 64, 1)),
    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    BatchNormalization(),
    Dropout(0.2),
    Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))
])
model4.compile(
     optimizer = tf.keras.optimizers.Adam(learning_rate=0.005),
     loss = 'binary_crossentropy',
     metrics = ['accuracy', Recall()]
 )
history4 = model4.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
```

```
Epoch 1/8
14/14 - 1s - 90ms/step - accuracy: 0.8249 - loss: 0.5662 - recall_38: 0.8064 - val_accuracy: 0.8369 - val_l
oss: 0.7658 - val_recall_38: 0.9964
Epoch 2/8
14/14 - 0s - 8ms/step - accuracy: 0.9229 - loss: 0.4486 - recall_38: 0.9450 - val_accuracy: 0.8514 - val_lo
ss: 0.6457 - val_recall_38: 0.8012
Epoch 3/8
14/14 - 0s - 33ms/step - accuracy: 0.9391 - loss: 0.3923 - recall_38: 0.9617 - val_accuracy: 0.9206 - val_l
oss: 0.4179 - val_recall_38: 0.9941
Epoch 4/8
14/14 - 0s - 16ms/step - accuracy: 0.9377 - loss: 0.3543 - recall_38: 0.9660 - val_accuracy: 0.9146 - val_l
oss: 0.3903 - val_recall_38: 0.9917
Epoch 5/8
14/14 - 0s - 9ms/step - accuracy: 0.9371 - loss: 0.3335 - recall_38: 0.9696 - val_accuracy: 0.9095 - val_lo
ss: 0.3922 - val_recall_38: 0.8852
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.9422 - loss: 0.2962 - recall_38: 0.9727 - val_accuracy: 0.6789 - val_lo
ss: 0.8949 - val_recall_38: 0.5550
Epoch 7/8
14/14 - 0s - 8ms/step - accuracy: 0.9402 - loss: 0.2847 - recall_38: 0.9590 - val_accuracy: 0.7515 - val_lo
ss: 0.6538 - val_recall_38: 0.6568
Epoch 8/8
14/14 - 0s - 7ms/step - accuracy: 0.9476 - loss: 0.2595 - recall_38: 0.9731 - val_accuracy: 0.9411 - val_lo
ss: 0.2837 - val_recall_38: 0.9314
```

Inccreased Complexity with L2, Batch Normalization, and Dropout

In [ ]:

```
model5 = Sequential([
    Flatten(input_shape=(64, 64, 1)),
    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    BatchNormalization(),
    Dropout(0.2),
    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    BatchNormalization(),
    Dropout(0.2),
    Dense(1, activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))
])

model5.compile(
     optimizer = tf.keras.optimizers.Adam(learning_rate=0.005),
     loss = 'binary_crossentropy',
```

```
    metrics = ['accuracy', Recall()]
)
history5 = model5.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
```

```
Epoch 1/8
14/14 - 2s - 138ms/step - accuracy: 0.8332 - loss: 0.6327 - recall_39: 0.8337 - val_accuracy: 0.7216 - val_
loss: 4.7364 - val_recall_39: 1.0000
Epoch 2/8
14/14 - 0s - 9ms/step - accuracy: 0.9354 - loss: 0.4769 - recall_39: 0.9520 - val_accuracy: 0.7216 - val_lo
ss: 5.0628 - val_recall_39: 1.0000
Epoch 3/8
14/14 - 0s - 8ms/step - accuracy: 0.9365 - loss: 0.4334 - recall_39: 0.9606 - val_accuracy: 0.7412 - val_lo
ss: 1.7459 - val_recall_39: 1.0000
Epoch 4/8
14/14 - 0s - 11ms/step - accuracy: 0.9490 - loss: 0.3754 - recall_39: 0.9711 - val_accuracy: 0.7242 - val_l
oss: 1.8264 - val_recall_39: 1.0000
Epoch 5/8
14/14 - 0s - 8ms/step - accuracy: 0.9471 - loss: 0.3331 - recall_39: 0.9672 - val_accuracy: 0.7558 - val_lo
ss: 1.0954 - val_recall_39: 0.9988
Epoch 6/8
14/14 - 0s - 8ms/step - accuracy: 0.9533 - loss: 0.2953 - recall_39: 0.9731 - val_accuracy: 0.7344 - val_lo
ss: 1.3104 - val_recall_39: 1.0000
Epoch 7/8
14/14 - 0s - 8ms/step - accuracy: 0.9513 - loss: 0.2690 - recall_39: 0.9715 - val_accuracy: 0.7259 - val_lo
ss: 1.2235 - val_recall_39: 1.0000
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.9607 - loss: 0.2427 - recall_39: 0.9762 - val_accuracy: 0.3535 - val_lo
ss: 2.1300 - val_recall_39: 0.1041
```

Evaluation of Models

In [ ]:

```
accuracies = {
    "Combination 1": history1.history['accuracy'][-1],
    "Combination 2": history2.history['accuracy'][-1],
    "Combination 3": history3.history['accuracy'][-1],
    "Combination 4": history4.history['accuracy'][-1],
    "Combination 5": history5.history['accuracy'][-1]
}

import pandas as pd

# Create a DataFrame to display the results
```

```
df_results = pd.DataFrame.from_dict(accuracies, orient='index', columns=['Training Accuracy'])
print(df_results)
```

```
              Training Accuracy
Combination 1           0.931398
Combination 2           0.919157
Combination 3           0.949900
Combination 4           0.947623
Combination 5           0.960717
```

In [ ]:
```
recall = {
    "Combination 1": history1.history['recall_35'][-1],
    "Combination 2": history2.history['recall_36'][-1],
    "Combination 3": history3.history['recall_37'][-1],
    "Combination 4": history4.history['recall_38'][-1],
    "Combination 5": history5.history['recall_39'][-1]
}

import pandas as pd

# Create a DataFrame to display the results
df_results = pd.DataFrame.from_dict(recall, orient='index', columns=['Training Recall'])
print(df_results)
```

```
              Training Recall
Combination 1         0.967213
Combination 2         0.959016
Combination 3         0.975020
Combination 4         0.973068
Combination 5         0.976190
```

In [ ]:
```
val_accuracies = {
    "Combination 1": history1.history['val_accuracy'][-1],
    "Combination 2": history2.history['val_accuracy'][-1],
    "Combination 3": history3.history['val_accuracy'][-1],
    "Combination 4": history4.history['val_accuracy'][-1],
    "Combination 5": history5.history['val_accuracy'][-1]
}

import pandas as pd

# Create a DataFrame to display the results
```

```
df_results = pd.DataFrame.from_dict(val_accuracies, orient='index', columns=['Validation Accuracy'])
print(df_results)
```

```
              Validation Accuracy
Combination 1             0.934244
Combination 2             0.948762
Combination 3             0.816396
Combination 4             0.941076
Combination 5             0.353544
```

In [ ]:
```
val_recall = {
    "Combination 1": history1.history['val_recall_35'][-1],
    "Combination 2": history2.history['val_recall_36'][-1],
    "Combination 3": history3.history['val_recall_37'][-1],
    "Combination 4": history4.history['val_recall_38'][-1],
    "Combination 5": history5.history['val_recall_39'][-1]
}

import pandas as pd

# Create a DataFrame to display the results
df_results = pd.DataFrame.from_dict(val_recall, orient='index', columns=['Validation Recall'])
print(df_results)
```

```
              Validation Recall
Combination 1           0.989349
Combination 2           0.978698
Combination 3           0.998817
Combination 4           0.931361
Combination 5           0.104142
```

Evaluate Models

Model 1

In [ ]:
```
model1.evaluate(x_train,y_train)
model1.evaluate(x_cv,y_cv)
model1.evaluate(x_test,y_test)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 993us/step - accuracy: 0.9253 - loss: 0.3001 - recall_35: 0.9819
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 847us/step - accuracy: 0.9295 - loss: 0.3112 - recall_35: 0.9833
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 739us/step - accuracy: 0.9230 - loss: 0.3062 - recall_35: 0.9760
```

Out[ ]:
```
[0.3048883080482483, 0.9274743795394897, 0.9815242290496826]
```

In [ ]:
```python
y_pred = model1.predict(x_train)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_train, y_pred_classes)
print(cm)

y_pred = model1.predict(x_cv)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_cv, y_pred_classes)
print(cm)

y_pred = model1.predict(x_test)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
print(cm)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
[[ 732  219]
 [  41 2521]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step
[[258  68]
 [  9 836]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 615us/step
[[237  69]
 [ 16 850]]
```

Model 2

In [ ]:
```python
model2.evaluate(x_train,y_train)
model2.evaluate(x_cv,y_cv)
```

```
model2.evaluate(x_test,y_test)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - accuracy: 0.9330 - loss: 0.3177 - recall_36: 0.9709
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - accuracy: 0.9487 - loss: 0.3257 - recall_36: 0.9752
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - accuracy: 0.9301 - loss: 0.3254 - recall_36: 0.9650
```

Out[ ]: [0.3241880536079407, 0.9334471225738525, 0.9665126800537109]

In [ ]:
```
y_pred = model2.predict(x_train)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_train, y_pred_classes)
print(cm)

y_pred = model2.predict(x_cv)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_cv, y_pred_classes)
print(cm)

y_pred = model2.predict(x_test)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
print(cm)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 582us/step
[[ 789  162]
 [  71 2491]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 608us/step
[[284  42]
 [ 18 827]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step
[[257  49]
 [ 29 837]]
```

Model 3

In [ ]:
```
model3.evaluate(x_train,y_train)
model3.evaluate(x_cv,y_cv)
model3.evaluate(x_test,y_test)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 862us/step - accuracy: 0.8351 - loss: 0.4680 - recall_37: 0.9999
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 754us/step - accuracy: 0.8101 - loss: 0.5117 - recall_37: 0.9987
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8360 - loss: 0.4948 - recall_37: 1.0000
```

Out[ ]:
```
[0.4813782274723053, 0.8378839492797852, 1.0]
```

In [ ]:
```
y_pred = model3.predict(x_train)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_train, y_pred_classes)
print(cm)

y_pred = model3.predict(x_cv)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_cv, y_pred_classes)
print(cm)

y_pred = model3.predict(x_test)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
print(cm)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
[[ 346  605]
 [   1 2561]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 697us/step
[[112 214]
 [  1 844]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step
[[116 190]
 [  0 866]]
```

Model 4

In [ ]:
```
model4.evaluate(x_train,y_train)
model4.evaluate(x_cv,y_cv)
model4.evaluate(x_test,y_test)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 845us/step - accuracy: 0.9386 - loss: 0.2701 - recall_38: 0.9256
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 755us/step - accuracy: 0.9364 - loss: 0.2983 - recall_38: 0.9280
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 867us/step - accuracy: 0.9052 - loss: 0.3031 - recall_38: 0.8944
```

Out[ ]:
```
[0.2956961393356323, 0.9155290126800537, 0.9041570425033569]
```

In [ ]:
```
y_pred = model4.predict(x_train)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_train, y_pred_classes)
print(cm)

y_pred = model4.predict(x_cv)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_cv, y_pred_classes)
print(cm)

y_pred = model4.predict(x_test)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
print(cm)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
[[ 923   28]
 [ 197 2365]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step
[[315  11]
 [ 58 787]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step
[[290  16]
 [ 83 783]]
```

Model 5

In [ ]:
```
model5.evaluate(x_train,y_train)
model5.evaluate(x_cv,y_cv)
model5.evaluate(x_test,y_test)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - accuracy: 0.3444 - loss: 2.1184 - recall_39: 0.1120
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 928us/step - accuracy: 0.3400 - loss: 2.1852 - recall_39: 0.0900
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.3407 - loss: 2.1916 - recall_39: 0.1113
```

Out[ ]:
```
[2.1281349658966064, 0.34897610545158386, 0.11893764138221741]
```

In [ ]:
```
y_pred = model5.predict(x_train)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_train, y_pred_classes)
print(cm)

y_pred = model5.predict(x_cv)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_cv, y_pred_classes)
print(cm)

y_pred = model5.predict(x_test)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
print(cm)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
[[ 950    1]
 [2278  284]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step
[[326   0]
 [757  88]]
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step
[[306   0]
 [763 103]]
```

HW 6

Q1: TensorFlow Model with best accuracy - Changing NN model

In [ ]:

```
# Define the model
model = Sequential([
    Flatten(input_shape=(64, 64, 1)),  # Input shape adjusted to 64x64x1
    Dense(64,
          kernel_initializer=tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05),
          bias_initializer=tf.keras.initializers.Constant(value=0.4),
          activation='relu'),
    Dense(1,
          kernel_initializer=tf.keras.initializers.HeUniform(seed=None),
          bias_initializer=tf.keras.initializers.Constant(value=0.4),
          activation='sigmoid')
])

model.compile(
     optimizer = tf.keras.optimizers.Adam(learning_rate=0.005),
     loss = 'binary_crossentropy',
     metrics = ['accuracy', Recall()]
 )
history = model.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
```

```
model.evaluate(x_cv, y_cv)
model.evaluate(x_test, y_test)
```

```
Epoch 1/8
/Users/andrewgatchalian/anaconda3/lib/python3.11/site-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
14/14 - 1s - 72ms/step - accuracy: 0.6985 - loss: 3.3007 - recall_40: 0.9290 - val_accuracy: 0.7216 - val_loss: 0.6250 - val_recall_40: 1.0000
Epoch 2/8
14/14 - 0s - 11ms/step - accuracy: 0.7293 - loss: 0.5604 - recall_40: 1.0000 - val_accuracy: 0.7216 - val_loss: 0.4997 - val_recall_40: 1.0000
Epoch 3/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.4524 - recall_40: 1.0000 - val_accuracy: 0.7216 - val_loss: 0.4118 - val_recall_40: 1.0000
Epoch 4/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.3990 - recall_40: 1.0000 - val_accuracy: 0.7216 - val_loss: 0.3800 - val_recall_40: 1.0000
Epoch 5/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.3675 - recall_40: 1.0000 - val_accuracy: 0.7216 - val_loss: 0.3570 - val_recall_40: 1.0000
Epoch 6/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.3465 - recall_40: 1.0000 - val_accuracy: 0.7216 - val_loss: 0.3385 - val_recall_40: 1.0000
Epoch 7/8
14/14 - 0s - 7ms/step - accuracy: 0.7293 - loss: 0.3269 - recall_40: 1.0000 - val_accuracy: 0.7216 - val_loss: 0.3201 - val_recall_40: 1.0000
Epoch 8/8
14/14 - 0s - 8ms/step - accuracy: 0.7293 - loss: 0.3148 - recall_40: 1.0000 - val_accuracy: 0.7216 - val_loss: 0.3001 - val_recall_40: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - accuracy: 0.7253 - loss: 0.3058 - recall_40: 1.0000
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 934us/step - accuracy: 0.7420 - loss: 0.2954 - recall_40: 1.0000
```

Out[ ]: `[0.2965989410877228, 0.7389078736305237, 1.0]`

Q2: CNN model

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Sequential, load_model
```

```
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, Dropout, Conv2D, MaxPooling2D
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.preprocessing import image

model = Sequential([
    Conv2D(filters=6, kernel_size=(5,5), strides=(1,1), padding='valid', activation = 'relu', input_shape=
    MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'),
    Conv2D(filters=6, kernel_size=(5,5), strides=(1,1), padding='valid', activation = 'relu'),
    MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'),
    Flatten(),
    Dense(120, activation = 'relu'),
    BatchNormalization(),
    Dropout(.2),
    Dense(84, activation = 'relu'),
    BatchNormalization(),
    Dropout(.2),
    Dense(1,activation = 'sigmoid')
])

model.compile(
     optimizer = tf.keras.optimizers.Adam(learning_rate=0.005),
     loss = 'binary_crossentropy',
     metrics = ['accuracy', Recall()]
 )
history = model.fit(x_train, y_train, epochs=8,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)

model.evaluate(x_cv, y_cv)
model.evaluate(x_test, y_test)
```

```
Epoch 1/8
```

```
/Users/andrewgatchalian/anaconda3/lib/python3.11/site-packages/keras/src/layers/convolutional/base_conv.py:
107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential model
s, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
14/14 - 4s - 311ms/step - accuracy: 0.8400 - loss: 0.3798 - recall_41: 0.8275 - val_accuracy: 0.7216 - val_
loss: 0.6166 - val_recall_41: 1.0000
Epoch 2/8
14/14 - 3s - 208ms/step - accuracy: 0.9379 - loss: 0.1851 - recall_41: 0.9551 - val_accuracy: 0.7216 - val_
loss: 0.7373 - val_recall_41: 1.0000
Epoch 3/8
14/14 - 2s - 168ms/step - accuracy: 0.9488 - loss: 0.1509 - recall_41: 0.9696 - val_accuracy: 0.8198 - val_
loss: 0.3383 - val_recall_41: 0.9964
Epoch 4/8
14/14 - 3s - 189ms/step - accuracy: 0.9499 - loss: 0.1332 - recall_41: 0.9676 - val_accuracy: 0.9240 - val_
loss: 0.2827 - val_recall_41: 0.9917
Epoch 5/8
14/14 - 2s - 160ms/step - accuracy: 0.9559 - loss: 0.1224 - recall_41: 0.9738 - val_accuracy: 0.8138 - val_
loss: 0.3301 - val_recall_41: 0.9976
Epoch 6/8
14/14 - 2s - 157ms/step - accuracy: 0.9496 - loss: 0.1357 - recall_41: 0.9746 - val_accuracy: 0.8762 - val_
loss: 0.2566 - val_recall_41: 0.9953
Epoch 7/8
14/14 - 3s - 187ms/step - accuracy: 0.9599 - loss: 0.1128 - recall_41: 0.9719 - val_accuracy: 0.7498 - val_
loss: 0.4185 - val_recall_41: 1.0000
Epoch 8/8
14/14 - 2s - 163ms/step - accuracy: 0.9545 - loss: 0.1157 - recall_41: 0.9738 - val_accuracy: 0.8651 - val_
loss: 0.2617 - val_recall_41: 0.9964
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8492 - loss: 0.2763 - recall_41: 0.9971
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8693 - loss: 0.2636 - recall_41: 0.9965
```

Out[ ]: `[0.25585824251174927, 0.8805460929870605, 0.9976905584335327]`

VGG Model

VGG is computationally expensive so we use lower # epochs. Also load the data as RGB

In [ ]:
```
import os
import numpy as np
import cv2
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
# import
```

```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
from PIL import Image
from scipy import ndimage
#from lr_utils import load_dataset

%matplotlib inline
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.preprocessing import image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, Dropout, Conv2D, MaxPooling2D
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.preprocessing import image
```

In [ ]:

```
# Function to load images and labels in grayscale
def load_images_and_labels(folder):
    images = []
    labels = []

    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        if filename.startswith('NORMAL'):
            labels.append('NORMAL')
        elif filename.startswith('VIRUS') or filename.startswith('BACTERIA'):
            labels.append('PNEUMONIA')
        else:
            # Skip unrelated files
            continue

        if os.path.isfile(img_path):
            # Read image as grayscale
```

```
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            if img is not None:
                images.append(img)

    return images, labels
```

In [ ]:
```
# Load images and labels from folder
folder_path = "/Users/andrewgatchalian/Documents/UCI MSBA 24/Spring Quarter/Deep Learning/Project/xray_data
images, labels = load_images_and_labels(folder_path)
```

In [ ]:
```
# Convert labels to numerical values
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
```

In [ ]:
```python
import cv2
import numpy as np
from tensorflow.keras.applications.resnet50 import preprocess_input

def preprocess_images_for_rgb(images, size=(64, 64)):
    processed_images = []
    for img in images:
        if img is None:
            continue  # Skip None entries if any

        # Resize images to the specified size
        resized_img = cv2.resize(img, size)

        # Convert grayscale image to pseudo-RGB by duplicating the grayscale channel across three channels
        if len(resized_img.shape) == 2:  # Checking if the image is grayscale
            resized_img = cv2.cvtColor(resized_img, cv2.COLOR_GRAY2RGB)

        # Apply ResNet50-specific preprocessing which includes scaling pixel values
        preprocessed_img = preprocess_input(resized_img)

        processed_images.append(preprocessed_img)

    return np.array(processed_images)

# Usage example
# Assuming 'images' is a list of loaded images (as numpy arrays)
processed_images = preprocess_images_for_rgb(images)
```

In [ ]:
```python
# Split the data into training, cross-validation, and test sets
x_train, x_test, y_train, y_test = train_test_split(processed_images, labels_encoded, test_size=0.2, train_
x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, test_size=0.25, train_size=0.75, random_s
```

In [ ]:
```python
vgg_model = tf.keras.applications.vgg16.VGG16(weights='imagenet', include_top=False, input_shape=(64, 64, 3
print(type(vgg_model))
vgg_model.summary()
#Input layer [(None, 224, 224, 3)]
#Output layer (None, 1000)
#two FC/dense layers: fc1 (Dense)(None, 4096), and fc2 (Dense) (None, 4096)
#params: Trainable params: 3,504,872
```

```
<class 'keras.src.models.functional.Functional'>
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 64, 64, 3)	0
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1,792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36,928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 32, 32, 128)	73,856
block2_conv2 (Conv2D)	(None, 32, 32, 128)	147,584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
block3_conv1 (Conv2D)	(None, 16, 16, 256)	295,168
block3_conv2 (Conv2D)	(None, 16, 16, 256)	590,080
block3_conv3 (Conv2D)	(None, 16, 16, 256)	590,080
block3_pool (MaxPooling2D)	(None, 8, 8, 256)	0
block4_conv1 (Conv2D)	(None, 8, 8, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 4, 4, 512)	0
block5_conv1 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 2, 2, 512)	0

Total params: 14,714,688 (56.13 MB)

Trainable params: 14,714,688 (56.13 MB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
# build Sequential model, iterate over VGG but omit the last layer (output layer)
model1 = Sequential()
for layer in vgg_model.layers:
    model1.add(layer)
model1.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1,792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36,928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 32, 32, 128)	73,856
block2_conv2 (Conv2D)	(None, 32, 32, 128)	147,584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
block3_conv1 (Conv2D)	(None, 16, 16, 256)	295,168
block3_conv2 (Conv2D)	(None, 16, 16, 256)	590,080
block3_conv3 (Conv2D)	(None, 16, 16, 256)	590,080
block3_pool (MaxPooling2D)	(None, 8, 8, 256)	0
block4_conv1 (Conv2D)	(None, 8, 8, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 4, 4, 512)	0
block5_conv1 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 2, 2, 512)	0

Total params: 14,714,688 (56.13 MB)

Trainable params: 14,714,688 (56.13 MB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
for layer in model1.layers:
    layer.trainable = False
model1.add(Flatten())
#model2.add(Dense(64, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))
model1.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1,792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36,928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 32, 32, 128)	73,856
block2_conv2 (Conv2D)	(None, 32, 32, 128)	147,584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
block3_conv1 (Conv2D)	(None, 16, 16, 256)	295,168
block3_conv2 (Conv2D)	(None, 16, 16, 256)	590,080
block3_conv3 (Conv2D)	(None, 16, 16, 256)	590,080
block3_pool (MaxPooling2D)	(None, 8, 8, 256)	0
block4_conv1 (Conv2D)	(None, 8, 8, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 4, 4, 512)	0
block5_conv1 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 1)	2,049

Total params: 14,716,737 (56.14 MB)

Trainable params: 2,049 (8.00 KB)

Non-trainable params: 14,714,688 (56.13 MB)

In [ ]:
```
from tensorflow.keras.metrics import Recall

model1.compile(
     optimizer = tf.keras.optimizers.Adam(learning_rate=0.005),
     loss = 'binary_crossentropy',
     metrics = ['accuracy', Recall()]
 )
```

In [ ]:
```
history = model1.fit(x_train, y_train, epochs=3,  validation_data=(x_cv, y_cv), batch_size=256,verbose=2)
```

```
Epoch 1/3
14/14 - 97s - 7s/step - accuracy: 0.9197 - loss: 1.4831 - recall: 0.9481 - val_accuracy: 0.9325 - val_loss:
1.3808 - val_recall: 0.9467
Epoch 2/3
14/14 - 82s - 6s/step - accuracy: 0.9451 - loss: 0.8711 - recall: 0.9606 - val_accuracy: 0.9394 - val_loss:
0.9881 - val_recall: 0.9562
Epoch 3/3
```

In [ ]:
```
model1.evaluate(x_train, y_train)
# Predict the training data
y_pred = model1.predict(x_train)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

from sklearn.metrics import confusion_matrix

# Confusion matrix
cm = confusion_matrix(y_train, y_pred_classes)
print(cm)
```

```
110/110 ━━━━━━━━━━━━━━━━━━━━ 62s 557ms/step - accuracy: 0.9555 - loss: 0.4853 - recall: 0.9887
110/110 ━━━━━━━━━━━━━━━━━━━━ 61s 552ms/step
[[ 821  130]
 [  30 2532]]
```

In [ ]:
```
model1.evaluate(x_cv, y_cv)
# Predict the training data
y_pred = model1.predict(x_cv)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output
```

```
# Confusion matrix
cm = confusion_matrix(y_cv, y_pred_classes)
print(cm)
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 20s 552ms/step - accuracy: 0.9406 - loss: 0.8787 - recall: 0.9689
37/37 ━━━━━━━━━━━━━━━━━━━━ 21s 582ms/step
[[281  45]
 [ 22 823]]
```

In [ ]:
```
model1.evaluate(x_test, y_test)
# Predict the training data
y_pred = model1.predict(x_test)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary output

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
print(cm)
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 22s 599ms/step - accuracy: 0.9424 - loss: 0.7597 - recall: 0.9757
37/37 ━━━━━━━━━━━━━━━━━━━━ 23s 621ms/step
[[262  44]
 [ 23 843]]
```