```
Final Project Code
by Group 3 - Bo Ching Liu,Soo Lee,Guanghao Li,Heng Li, Mengting Tang,
Cheney Wong
```

In [1]:
```
import pandas as pd
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from eye_functions import *
import plotly_express as px
from sklearn.model_selection import train_test_split
from skimage.measure import block_reduce
from skimage.io import imread
```

In [2]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization,
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout
```

In [3]:
```
eyes = pd.read_csv('full_df.csv')
eyes = eyes[(eyes['labels']=="['N']") | (eyes['labels']=="['C']")]
eyes['eyes_emoji']=0
eyes.loc[eyes['labels'] == "['C']", 'eyes_emoji'] = 1
```

In [4]:
```
Y = eyes['eyes_emoji']
X = eyes['filename']
```

In [5]:
```
plotter = pd.DataFrame({'Cataracts' : Y.values.astype(str) , 'eyes' : X.
plotter.eyes.replace([1,0],['Right','Left'],inplace=True)
px.histogram(plotter, x='Cataracts',color='eyes',text_auto=True)
```

In [6]:
```
def image_matrix(sets):
    matrices = []
    for link in sets:
        image = Image.open('preprocessed_images/%s' % link)
        if image.mode != "RGB":
            image = image.convert("RGB")
        orig_array = np.asarray(image)
        resized = block_reduce(orig_array, (2, 2, 1), np.max)
        normed_array = (resized - resized.mean())/resized.std()

        matrices.append(resized)

    stacked_arrays = np.stack(matrices, axis=0)
    return stacked_arrays
```

#

In [7]:
```
#selecing only the right eye. I had hoped that perhaps the orientation c
# right_x = diabetes[diabetes['eyes_side'] ==1].filename
# right_y = diabetes[diabetes['eyes_side'] ==1].eyes_emoji
X1, x_test, Y1, y_test = train_test_split(X,Y,test_size=0.2,train_size=0
```

In [8]:
```
image_train = image_matrix(X1)
```

In [9]:
```
image_test =  image_matrix(x_test)
```

In [10]:
```
print("train X shape: ", X1.shape) # X_train: numpy array with shape: (r
print("train y shape: ", Y1.shape) # y_train: numpy array with shape: (r
#print("test X shape: ", x_test.shape)   # X_test: numpy array with shap
#print("test y shape: ", y_test.shape)   # y_test: numpy array with shap
print("image_train.shape: ", image_train.shape)
```

```
train X shape:  (2532,)
train y shape:  (2532,)
image_train.shape:  (2532, 256, 256, 3)
```

Best NN model

In [11]:
```
model1 = Sequential([
    Flatten(input_shape=(image_train.shape[1:])),# BatchNormalization(),
    Dense(256, activation='relu',kernel_initializer=tf.keras.initializer
    BatchNormalization(momentum = 0.2 , epsilon= 0.001),
    Dense(256, activation='relu',kernel_initializer=tf.keras.initializer
    BatchNormalization(momentum = 0.2 , epsilon= 0.001),
    Dense(256, activation='relu',kernel_initializer=tf.keras.initializer
    BatchNormalization(momentum = 0.2 , epsilon= 0.001),
    Dense(256, activation='relu',kernel_initializer=tf.keras.initializer
    BatchNormalization(momentum = 0.2 , epsilon= 0.001),
    Dense(1, activation='sigmoid')
])

model1.compile( optimizer='adam', loss='binary_crossentropy', metrics=['
```

In [12]:
```
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 196608)            0

 dense (Dense)               (None, 256)               50331904

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dense_1 (Dense)             (None, 256)               65792

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_2 (Dense)             (None, 256)               65792

 batch_normalization_2 (Batc  (None, 256)              1024
 hNormalization)

 dense_3 (Dense)             (None, 256)               65792

 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 dense_4 (Dense)             (None, 1)                 257

=================================================================
Total params: 50,533,633
Trainable params: 50,531,585
Non-trainable params: 2,048
_________________________________________________________________
```

In [13]:
```
model1.compile( optimizer='adam', loss='binary_crossentropy', metrics=['
```

In [14]:
```
checkpoint_filepath = '~\checkpoints'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='loss',
    mode='min',
    save_best_only=True)
```

In [15]:
```
history1 = model1.fit(image_train, Y1, epochs = 72, batch_size = 64, ver
```

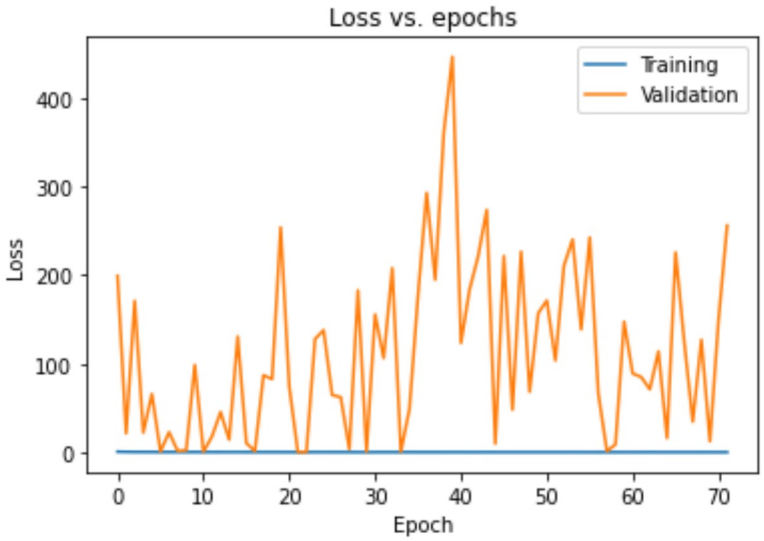
```
Epoch 1/72
32/32 - 15s - loss: 0.6558 - binary_accuracy: 0.6612 - recall_1: 0.512
8 - val_loss: 199.2352 - val_binary_accuracy: 0.7929 - val_recall_1:
0.5128 - 15s/epoch - 468ms/step
Epoch 2/72
32/32 - 13s - loss: 0.4363 - binary_accuracy: 0.8504 - recall_1: 0.338
5 - val_loss: 21.2846 - val_binary_accuracy: 0.8363 - val_recall_1: 0.
5128 - 13s/epoch - 419ms/step
Epoch 3/72
32/32 - 13s - loss: 0.3329 - binary_accuracy: 0.8968 - recall_1: 0.276
9 - val_loss: 171.0624 - val_binary_accuracy: 0.8994 - val_recall_1:
0.2821 - 13s/epoch - 409ms/step
Epoch 4/72
32/32 - 13s - loss: 0.2878 - binary_accuracy: 0.9057 - recall_1: 0.276
9 - val_loss: 22.2475 - val_binary_accuracy: 0.8521 - val_recall_1: 0.
4872 - 13s/epoch - 405ms/step
Epoch 5/72
32/32 - 13s - loss: 0.2729 - binary_accuracy: 0.9052 - recall_1: 0.225
6 - val_loss: 65.8326 - val_binary_accuracy: 0.8777 - val_recall_1: 0.
```

In [16]:
```
print(history1.history.keys())
```

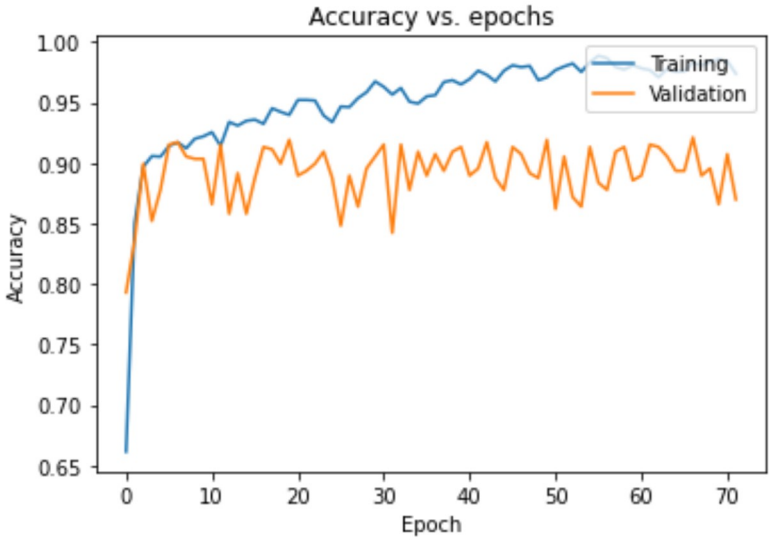
```
dict_keys(['loss', 'binary_accuracy', 'recall_1', 'val_loss', 'val_bin
ary_accuracy', 'val_recall_1'])
```

In [17]:
```
# Plot the training and validation loss

plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```

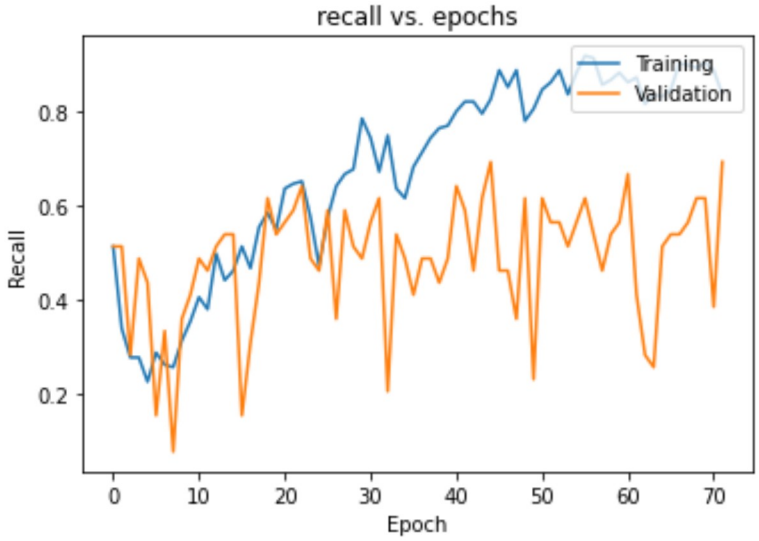


In [18]:
```
plt.plot(history1.history['binary_accuracy'])
plt.plot(history1.history['val_binary_accuracy'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [19]:
```
# Make a plot for the recall

plt.plot(list(history1.history.values())[2])
plt.plot(list(history1.history.values())[5])
plt.title('recall vs. epochs')
plt.ylabel('Recall')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



CNN

In [20]:
```
Y1 = Y1.values.reshape((-1,1))
```

In [21]:
```
model = Sequential([
            Conv2D(filters = 5, kernel_size = (4,4), activation = 'relu'
            MaxPooling2D(pool_size=2, strides=(2,2)), #2
            Conv2D(filters = 6, kernel_size = (4,4), activation = 'relu'
            MaxPooling2D(pool_size=2, strides=(2,2)), #4
            Flatten(), #5
            Dense(256, activation='relu',kernel_initializer=tf.keras.ini
            BatchNormalization(momentum = 0.2 , epsilon=0.001), # <- Bat
            Dense(256, activation='relu',kernel_initializer=tf.keras.ini
            BatchNormalization(momentum = 0.2 , epsilon=0.001),
            Dense(256, activation='relu',kernel_initializer=tf.keras.ini
            BatchNormalization(momentum = 0.2 , epsilon=0.001),
            Dense(256, activation='relu',kernel_initializer=tf.keras.ini
            BatchNormalization(momentum = 0.2 , epsilon=0.001), # <- Bat
            Dense(1, activation = 'sigmoid') #output
])
```

In [22]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 253, 253, 5)       245

 max_pooling2d (MaxPooling2D  (None, 126, 126, 5)      0
 )

 conv2d_1 (Conv2D)           (None, 123, 123, 6)       486

 max_pooling2d_1 (MaxPooling  (None, 61, 61, 6)        0
 2D)

 flatten_1 (Flatten)         (None, 22326)             0

 dense_5 (Dense)             (None, 256)               5715712

 batch_normalization_4 (Batc  (None, 256)              1024
 hNormalization)

 dense_6 (Dense)             (None, 256)               65792

 batch_normalization_5 (Batc  (None, 256)              1024
 hNormalization)

 dense_7 (Dense)             (None, 256)               65792

 batch_normalization_6 (Batc  (None, 256)              1024
 hNormalization)

 dense_8 (Dense)             (None, 256)               65792

 batch_normalization_7 (Batc  (None, 256)              1024
 hNormalization)

 dense_9 (Dense)             (None, 1)                 257

=================================================================
Total params: 5,918,172
Trainable params: 5,916,124
Non-trainable params: 2,048
_________________________________________________________________
```

In [23]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['bi
```

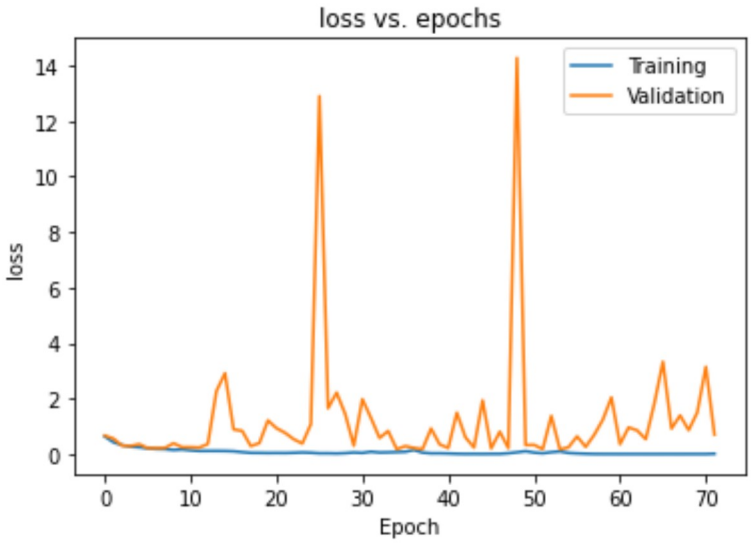
In [24]:
```
checkpoint_filepath = '~/checkpoints'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)
```

In [25]:
```
history = model.fit(image_train, Y1, epochs = 72, batch_size = 64, verbo
```

```
Epoch 1/72
32/32 - 13s - loss: 0.6368 - binary_accuracy: 0.6706 - recall_2: 0.564
1 - val_loss: 0.6663 - val_binary_accuracy: 0.7495 - val_recall_2: 0.5
385 - 13s/epoch - 409ms/step
Epoch 2/72
32/32 - 12s - loss: 0.4276 - binary_accuracy: 0.8617 - recall_2: 0.384
6 - val_loss: 0.5601 - val_binary_accuracy: 0.8718 - val_recall_2: 0.3
846 - 12s/epoch - 368ms/step
Epoch 3/72
32/32 - 12s - loss: 0.3241 - binary_accuracy: 0.8988 - recall_2: 0.297
4 - val_loss: 0.3080 - val_binary_accuracy: 0.8895 - val_recall_2: 0.4
872 - 12s/epoch - 367ms/step
Epoch 4/72
32/32 - 12s - loss: 0.2658 - binary_accuracy: 0.9096 - recall_2: 0.338
5 - val_loss: 0.2895 - val_binary_accuracy: 0.8757 - val_recall_2: 0.4
103 - 12s/epoch - 368ms/step
Epoch 5/72
32/32 - 11s - loss: 0.2482 - binary_accuracy: 0.9156 - recall_2: 0.328
2 - val_loss: 0.3552 - val_binary_accuracy: 0.8679 - val_recall_2: 0.6
```

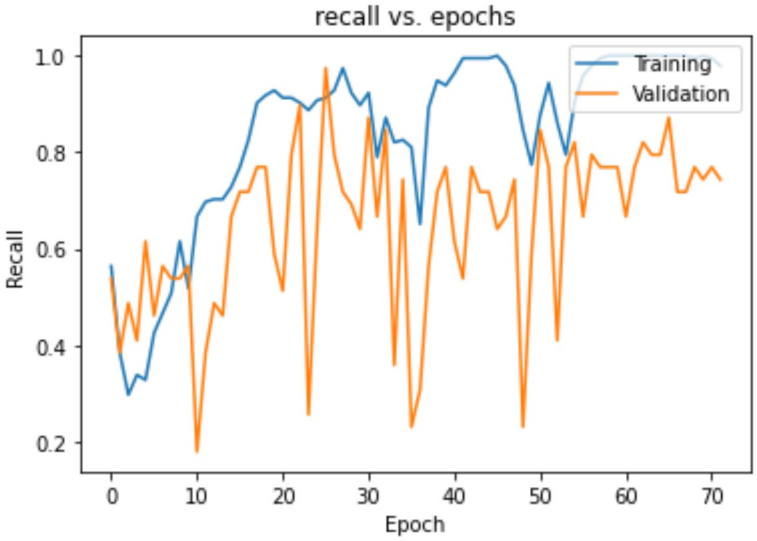
In [26]:

```
plt.plot(list(history.history.values())[0])
plt.plot(list(history.history.values())[3])
plt.title('loss vs. epochs')
plt.ylabel('loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```

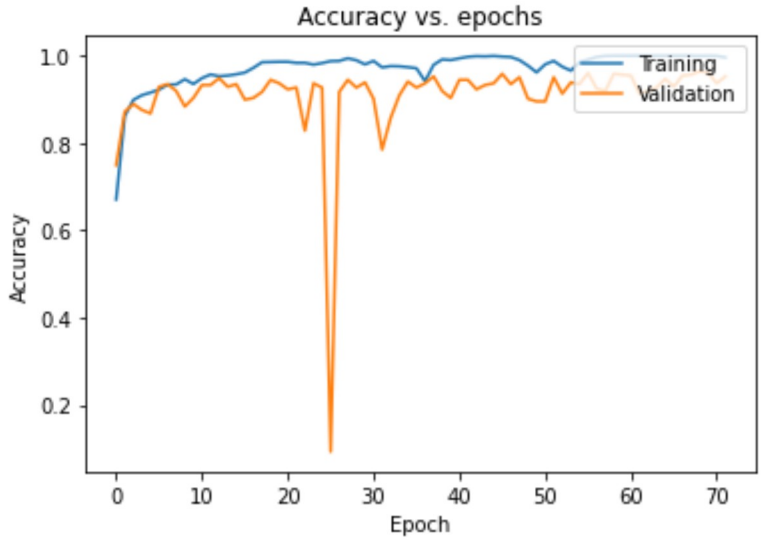


In [27]:

```
plt.plot(list(history.history.values())[2])
plt.plot(list(history.history.values())[5])
plt.title('recall vs. epochs')
plt.ylabel('Recall')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [28]:
```
plt.plot(history.history['binary_accuracy'])
plt.plot(history.history['val_binary_accuracy'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [29]:
```
CNN_predictions = model.predict(image_test)
```

```
20/20 [==============================] - 1s 40ms/step
```

In [30]:
```
NN_predictions = model1.predict(image_test)
```

```
20/20 [==============================] - 0s 20ms/step
```

In [31]:
```
from sklearn.metrics import confusion_matrix
cnn_cm = confusion_matrix(np.round(CNN_predictions.flatten()).astype(int
nn_cm = confusion_matrix(np.round(NN_predictions.flatten()).astype(int),
print(cnn_cm)
print(nn_cm)
```

```
[[547  16]
 [ 28  43]]
[[502  23]
 [ 73  36]]
```