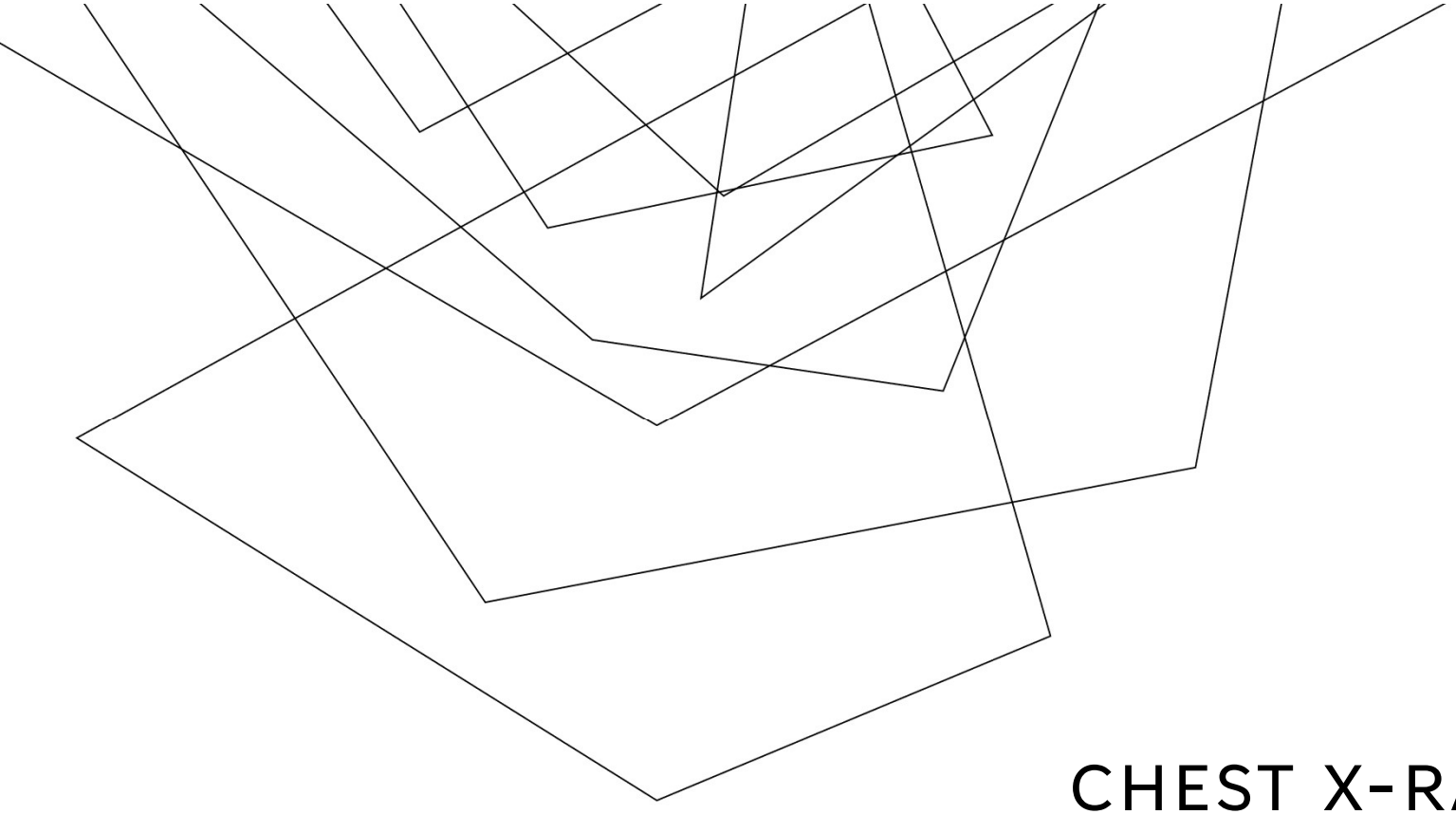


CHEST X-RAY ANALYSIS

Ryan S. Chung

AGENDA

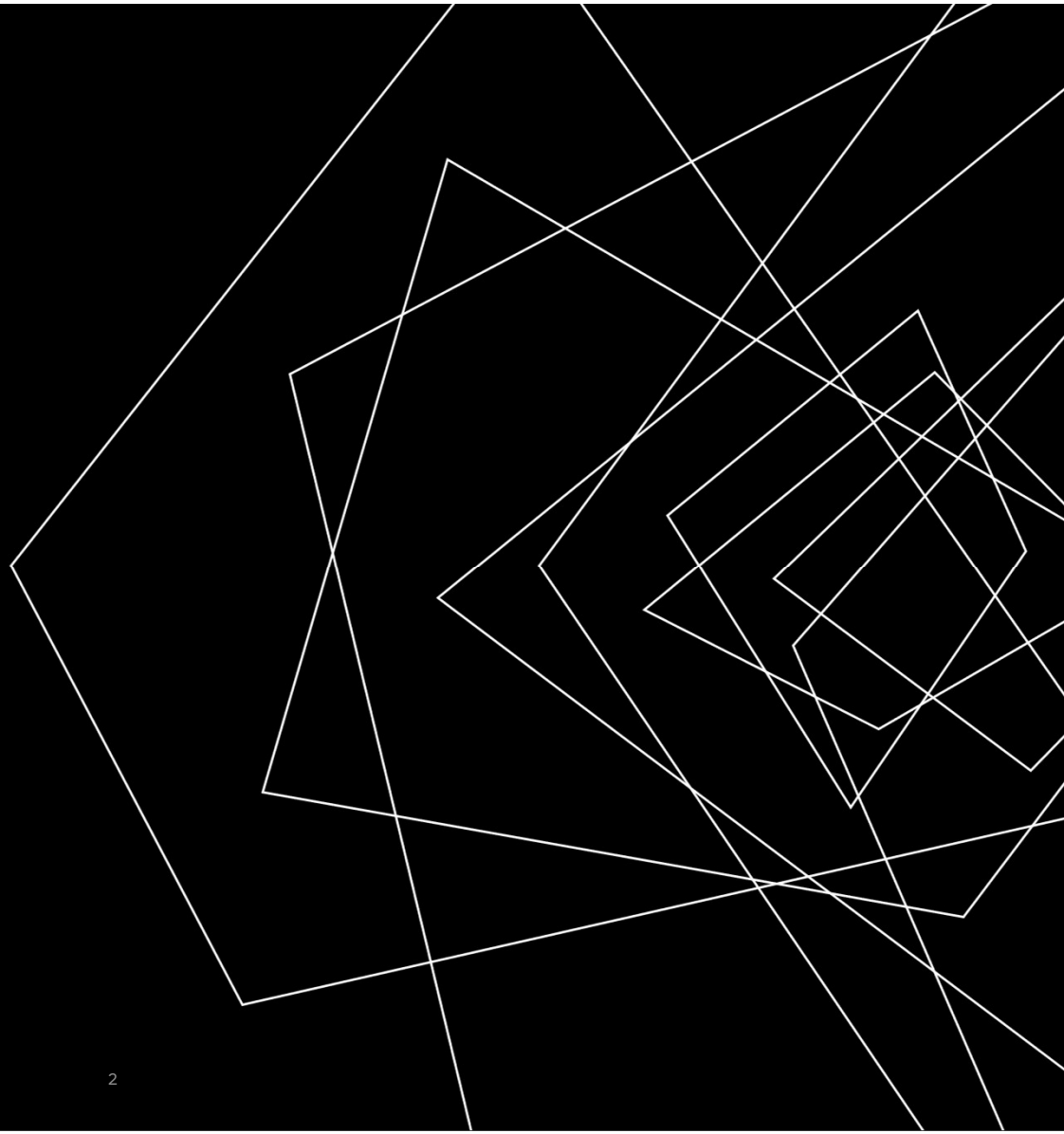
Introduction

Data

Methods

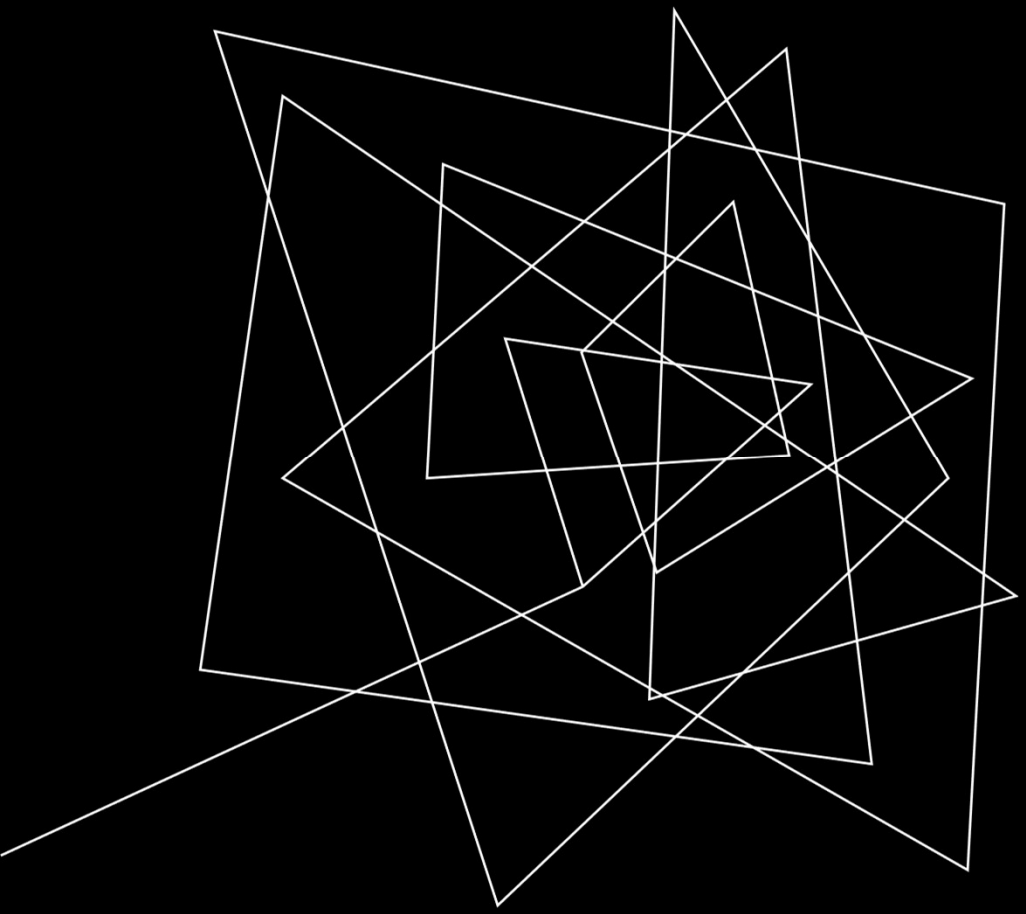
Results

Summary



INTRODUCTION

For this project, the presented business problem was to assist an agency in identifying whether chest x-ray images indicated signs of Pneumonia or not. Our goal for this project was to create an image classification model that would accurately predict whether a provided chest x-ray was healthy or not.



DATA

For this project, a dataset from Kaggle called Chest X-Ray Images (Pneumonia) was used. It should also be mentioned that the initial source of images for the Kaggle dataset were obtained from Mendeley Data which houses thousands of validated OCT and Chest X-Ray images. The data from the Kaggle set was well organized which made its use for the completion of this project swift. As stated earlier, the images from the Kaggle dataset consisted of healthy chest X-rays and chest X-rays with indications of Pneumonia.

METHODS

In this project, initially a Multi-Layer Perception model was created. After the MLP was created, we then moved forward to creating a Convolutional Neural Network model. Once the model had been created, it was then run through a grid search using Talos in order to find the best parameters for the model.

RESULTS

INITIAL MLP

```
Train_NORMAL = glob.glob("C:\\Users\\rychu\\Desktop\\2021\\FLT\\P4-Project\\chest_xray\\train\\NORMAL\\*.jpeg")
Train_PNEUMONIA = glob.glob("C:\\Users\\rychu\\Desktop\\2021\\FLT\\P4-Project\\chest_xray\\train\\PNEUMONIA\\*jpeg")
```

```
train_data = []
train_labels = []
```

```
for i in Train_NORMAL:
    image= tf.keras.preprocessing.image.load_img(i, color_mode='grayscale', target_size= (64,64))
    image=np.array(image)
    train_data.append(image)
    train_labels.append(0)

for i in Train_PNEUMONIA:
    image= tf.keras.preprocessing.image.load_img(i, color_mode='grayscale', target_size= (64,64))
    image=np.array(image)
    train_data.append(image)
    train_labels.append(1)
```

```
train_data = np.array(train_data)
train_labels = np.array(train_labels)

# X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
```

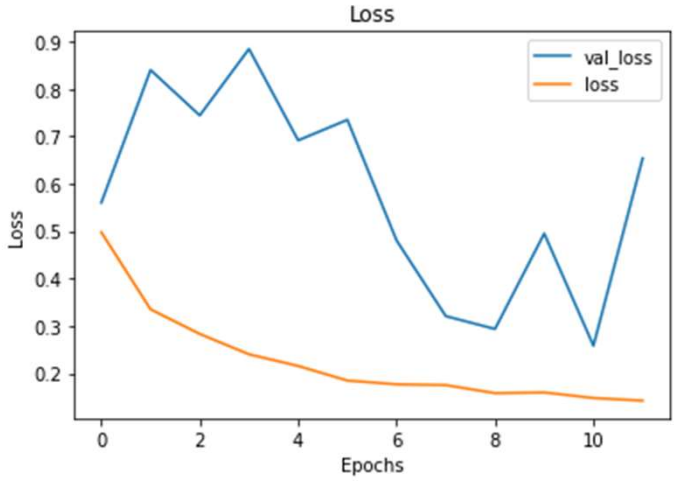
```
Val_NORMAL = glob.glob("C:\\Users\\rychu\\Desktop\\2021\\FLT\\P4-Project\\chest_xray\\val\\NORMAL\\*.jpeg")
Val_PNEUMONIA = glob.glob("C:\\Users\\rychu\\Desktop\\2021\\FLT\\P4-Project\\chest_xray\\val\\PNEUMONIA\\*jpeg")
```

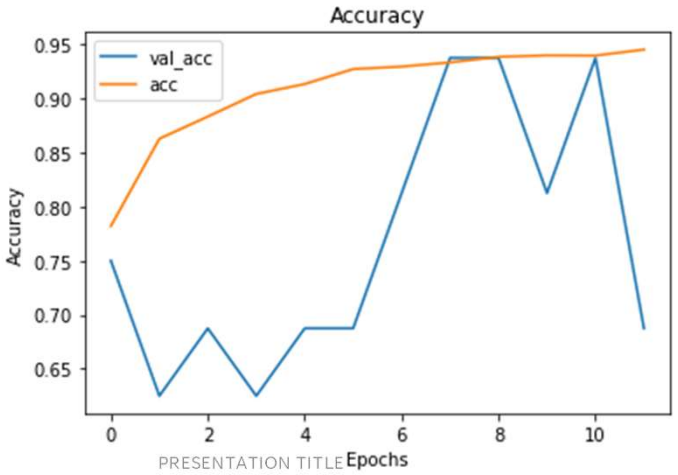
```
val_data = []
val_labels = []
```

```
for i in Val_NORMAL:
    image= tf.keras.preprocessing.image.load_img(i, color_mode='grayscale', target_size= (64,64))
    image=np.array(image)
    val_data.append(image)
    val_labels.append(0)

for i in Val_PNEUMONIA:
    image= tf.keras.preprocessing.image.load_img(i, color_mode='grayscale', target_size= (64,64))
    image=np.array(image)
    val_data.append(image)
    val_labels.append(1)
```

```
val_data = np.array(val_data)
val_labels = np.array(val_labels)
```



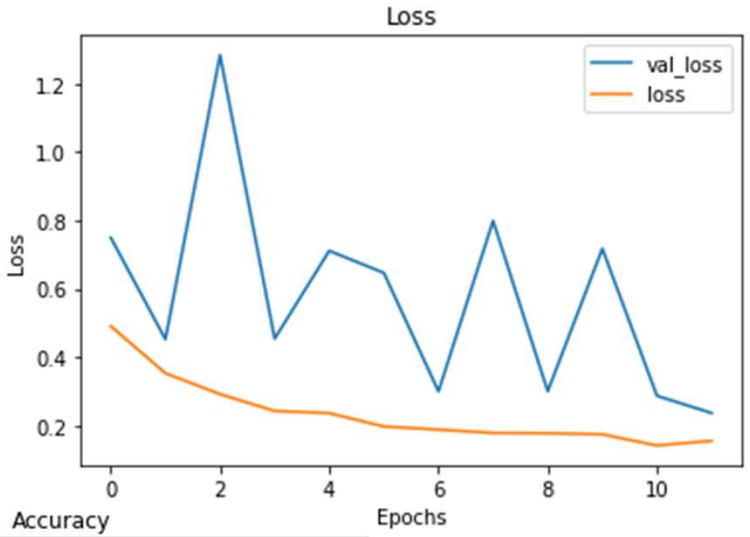


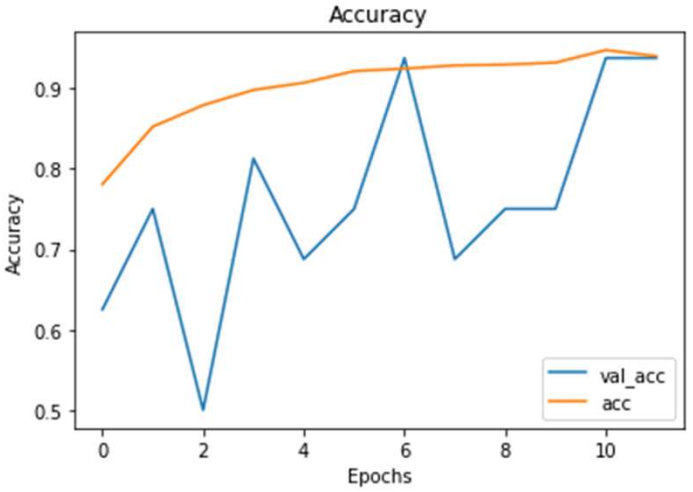
2ND MLP

```
1 Model_2 = Sequential()
2 Model_2.add(Dense(64, activation='tanh', input_shape=(4096,)))
3 Model_2.add(Dense(32, activation='tanh'))
4 Model_2.add(Dense(2, activation='softmax'))
```

```
1 Model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 64)                262208
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 66
=================================================================
Total params: 264,354
Trainable params: 264,354
Non-trainable params: 0
_________________________________________________________________
```





3RD MLP

```
Model_3 = Sequential()
Model_3.add(Dense(64, activation='relu', input_shape=(4096,)))
Model_3.add(Dense(32, activation='relu'))
Model_3.add(Dense(2, activation='softmax'))
```

```
Model_3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 64)                262208
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 66
=================================================================
Total params: 264,354
Trainable params: 264,354
Non-trainable params: 0
_________________________________________________________________
```

```
results_train = Model_3.evaluate(train_data, train_labels)
```

```
163/163 [==============================] - 0s 852us/step - loss: 0.1234 - acc: 0.9509
```

```
results_test = Model_3.evaluate(val_data, val_labels)
```

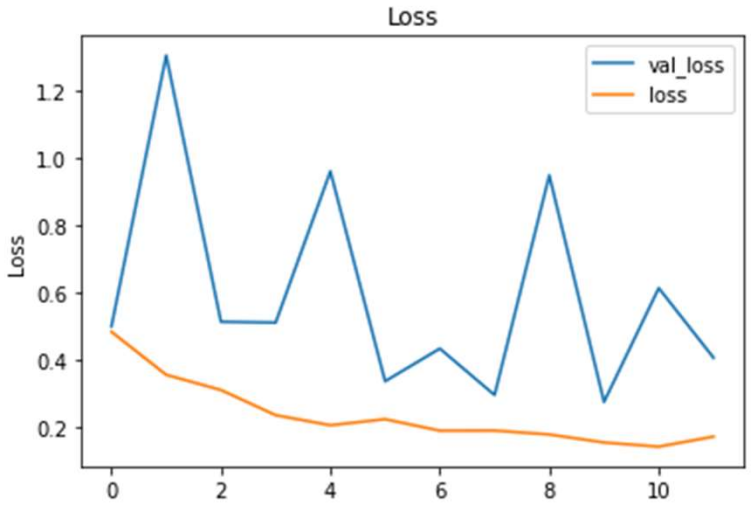
```
1/1 [==============================] - 0s 1ms/step - loss: 0.4065 - acc: 0.8125
```

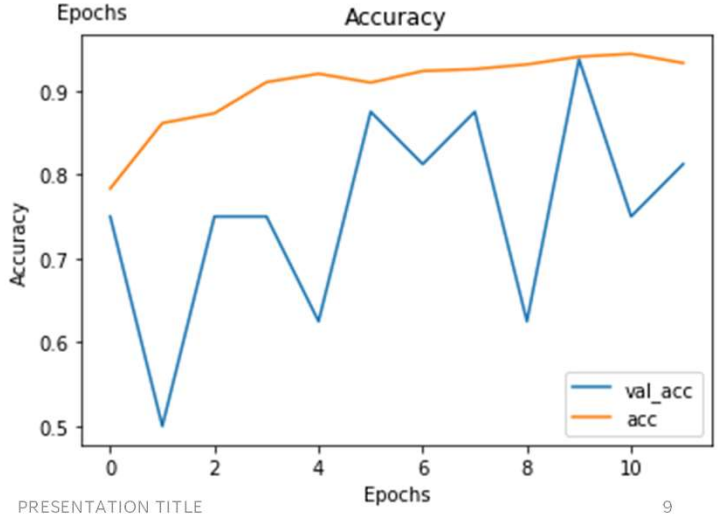
```
results_train
```

```
[0.1233837753534317, 0.9509202241897583]
```

```
results_test
```

```
[0.40651610493659973, 0.8125]
```





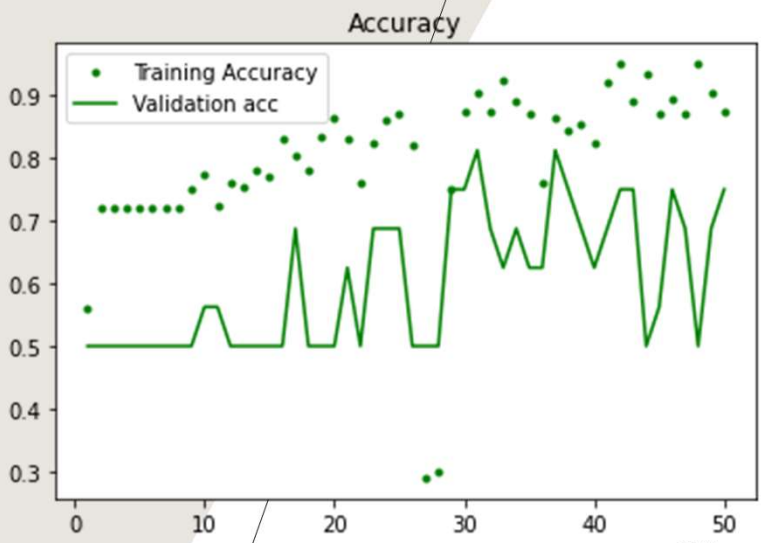
INITIAL CNN

```
from keras import models
from keras import layers
np.random.seed(123)

model = models.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(12288,))) # 2 hidden layers
model.add(layers.Dense(7, activation='relu'))
model.add(layers.Dense(5, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

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

history = model.fit(train_img,
                    train_y,
                    epochs=50,
                    batch_size=32,
                    validation_data=(val_img, val_y))
```




```
results_train = model.evaluate(train_img, train_y)
```

```
7/7 [==============================] - 0s 714us/step - loss: 0.2346 - accuracy: 0.9050
```

```
results_test = model.evaluate(test_img, test_y)
```

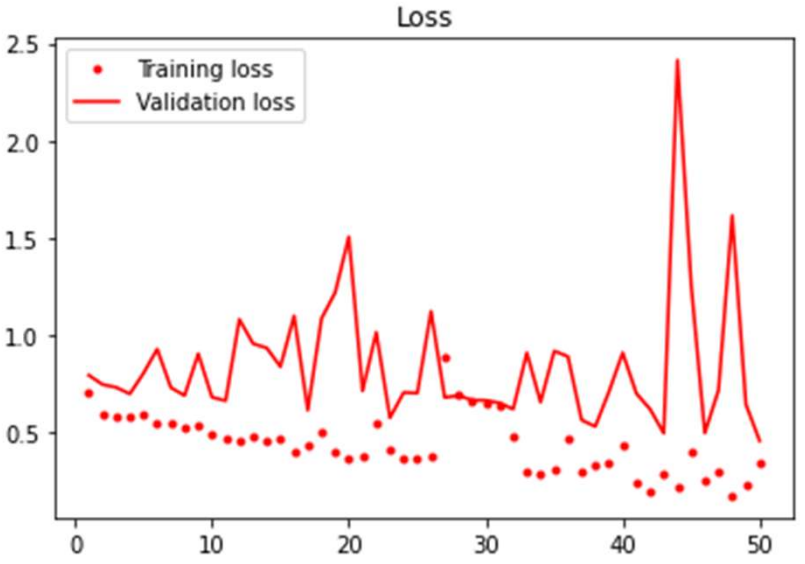
```
6/6 [==============================] - 0s 750us/step - loss: 0.3340 - accuracy: 0.8722
```

```
results_train
```

```
[0.23455727100372314, 0.9049999713897705]
```

```
results_test
```

```
[0.3339858949184418, 0.8722222447395325]
```



2ND CNN

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

model.add(layers.Conv2D(32, (4, 4), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

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

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

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

history = model.fit(train_images,
                    train_y,
                    epochs=30,
                    batch_size=32,
                    validation_data=(val_images, val_y))
```

```python
results_train = model.evaluate(train_images, train_y)
```

```
7/7 [==============================] - 0s 6ms/step - loss: 0.5448 - acc: 0.7200
```

```python
results_test = model.evaluate(test_images, test_y)
```

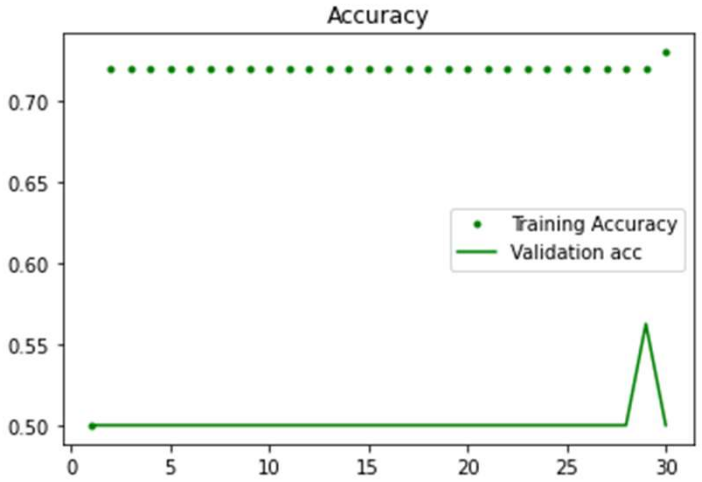
```
6/6 [==============================] - 0s 6ms/step - loss: 0.5798 - acc: 0.6889
```

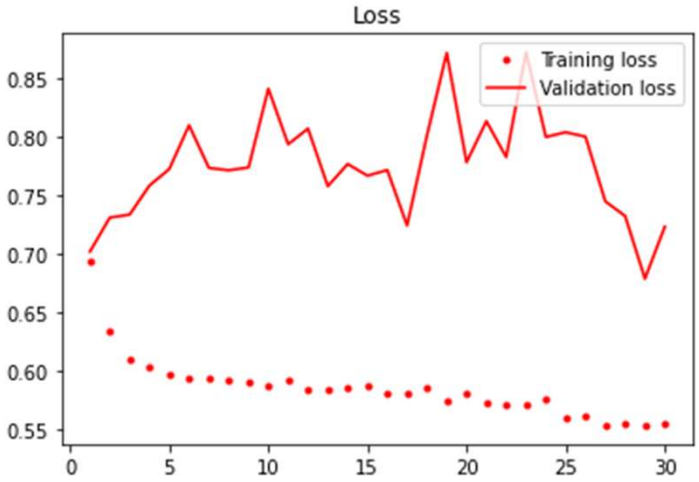
```python
results_train
```

```
[0.5448285341262817, 0.7200000286102295]
```

```python
results_test
```

```
[0.579835832118988, 0.6888889074325562]
```





TUNING WITH TALOS

```python
def dense_network(x_train, y_train, x_test, y_test, params):

    model = models.Sequential()

    # hidden layers
    model.add(layers.Conv2D(32, (3, 3), activation=params['activation1'], input_shape=(64 ,64,  3)))
    model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Conv2D(32, (4, 4), activation=params['activation2']))
    model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Conv2D(64, (3, 3), activation=params['activation3']))
    model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Flatten())

    model.add(layers.Dense(64, activation=params['activation4']))
    model.add(layers.Dropout(params['dropout']))

    # output layer
    model.add(layers.Dense(1, activation='sigmoid'))

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

    out = model.fit(train_images,
                        train_y,
                        epochs=30,
                        batch_size=32,
                        validation_data=(val_images, val_y))
    return out, model
```

```python
params = {'dropout': [0.1, 0.3, 0.5],
          'optimizer': ['adam', 'sgd'],
          'activation1': ['relu', 'tanh'],
          'activation2': ['relu', 'tanh'],
          'activation3': ['relu', 'tanh'],
          'activation4': ['relu', 'tanh'],}
```

```python
results = talos.Scan(X_train, y_train, params=params, model = dense_network, experiment_name='grid')
```

```python
df.sort_values('val_acc', ascending=False)
```

	round_epochs	loss	acc	val_loss	val_acc	activation1	activation2	activation3	activation4	dropout	optimizer
14	30	0.045584	0.990	0.355963	0.9375	relu	relu	tanh	relu	0.3	adam
90	30	0.015954	1.000	0.637691	0.8750	tanh	tanh	tanh	tanh	0.1	adam
46	30	0.038556	0.990	0.404017	0.8750	relu	tanh	tanh	tanh	0.5	adam
32	30	0.088186	0.965	0.428396	0.8750	relu	tanh	relu	tanh	0.3	adam
16	30	0.124078	0.950	0.396825	0.8750	relu	relu	tanh	relu	0.5	adam
...	...	...	...	...	...	...	...	...	...	...	...
45	30	0.424899	0.795	0.986641	0.5000	relu	tanh	tanh	tanh	0.3	sgd
17	30	0.538083	0.745	0.735461	0.5000	relu	relu	tanh	relu	0.5	sgd
71	30	0.479030	0.765	0.899852	0.5000	tanh	relu	tanh	tanh	0.5	sgd
15	30	0.543857	0.745	0.854714	0.5000	relu	relu	tanh	relu	0.3	sgd
73	30	0.517773	0.745	0.777227	0.5000	tanh	tanh	relu	relu	0.1	sgd

FINAL MODEL

```python
model = models.Sequential()

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

model.add(layers.Conv2D(32, (4, 4), activation='tanh'))
model.add(layers.MaxPooling2D((2, 2)))

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

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))

# output layer
model.add(layers.Dense(1, activation='sigmoid'))

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

history = model.fit(train_images,
                        train_y,
                        epochs=40,
                        batch_size=32,
                        validation_data=(val_images, val_y))
```

```python
results_train = model.evaluate(train_images, train_y)
```

```
7/7 [==============================] - 0s 7ms/step - loss: 8.9096e-04 - acc: 1.0000
```

```python
results_val = model.evaluate(val_images, val_y)
```

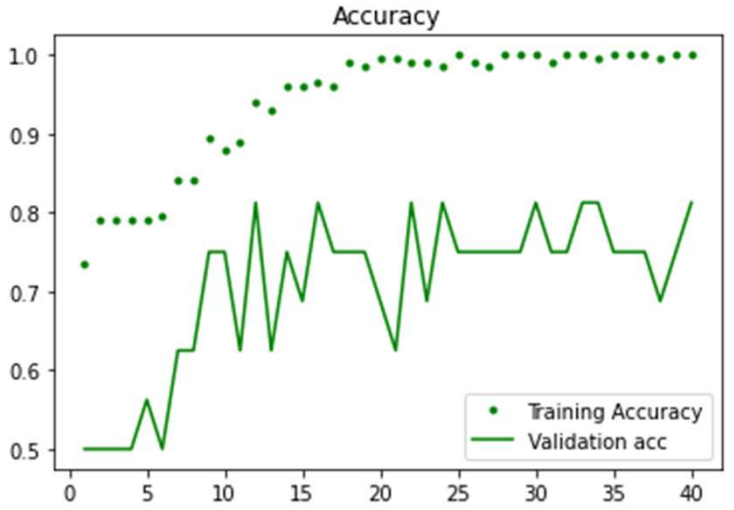
```
1/1 [==============================] - 0s 499us/step - loss: 1.7688 - acc: 0.8125
```

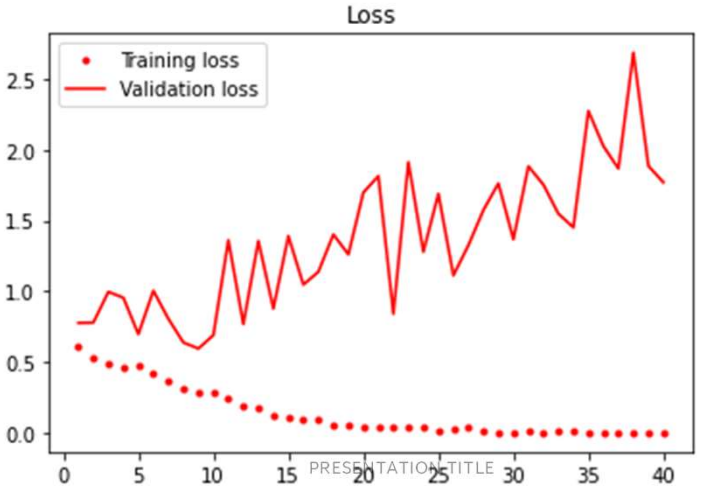
```python
results_train
```

```
[0.0008909601019695401, 1.0]
```

```python
results_val
```

```
[1.7688368558883667, 0.8125]
```





CONCLUSION

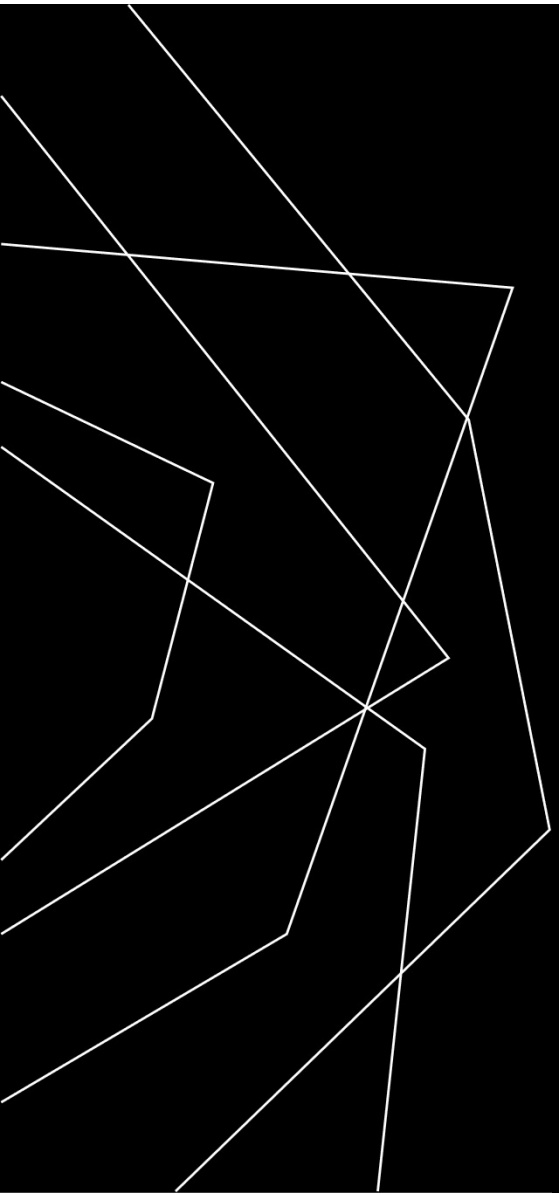
1. Our final MLP model had an accuracy of 95% on the training set of images.

2. The final MLP model had an accuracy of 81% on the validation set of images.

3. After being run through a grid search, the final Convolutional Neural Network had an accuracy of 100% on the training set of images.

4. After being run through a grid search, the final Convolutional Neural Network had an accuracy of 81% on the training set of images.

QUESTIONS?



THANK YOU