Neural Networks Project 2: Classification of Parking Space Images

Group Members: Juliann Weir-Jackson

Problem Statement:

The goal of this project was to classify images of individual parking spaces as occupied or not occupied. The dataset used contained 3,262 RGB images from a raspberry pi all of size 1296 x 972. The dataset was in a directory format with all images representing free spaces in a folder named Free and all images representing occupied spaces in a folder labeled Full.

Data Preparation:

First a google drive was mounted to after which the image_datset_from_directory function was used to create the dataset.

```
X_data =image_dataset_from_directory(
    directory='/content/gdrive/MyDrive/data',
    image_size=(224,224),
    shuffle=True,
    labels='inferred',
    label_mode='binary',
    color_mode='rgb',
    seed=23,
    batch_size=3262
)
```

All images in the dataset were resized in this step as well to 224 X 224, the dataset was also shuffled and labelled using this function. Because of the size of the dataset all images were loaded in one batch.

Benchmarking

A shallow net was used as the Benchmark. The network contained one convolution layer and used a SoftMax classifier. Once the benchmark was executed a training accuracy of 82.3% was achieved and a validation accuracy of 82.6 percent was achieved. Seeing as this problem is binary the accuracy would already start at 50%.

```
None
[INFO] training network...
Epoch 1/5
13/13 [==============================] - 17s 85ms/step - loss: 5.5259 - acc: 0.5502 - val_loss: 2.6935 - val_acc: 0.5051
Epoch 2/5
13/13 [==============================] - 1s 56ms/step - loss: 1.1020 - acc: 0.6831 - val_loss: 0.4985 - val_acc: 0.7679
Epoch 3/5
13/13 [==============================] - 1s 56ms/step - loss: 0.4646 - acc: 0.8026 - val_loss: 0.4764 - val_acc: 0.8112
Epoch 4/5
13/13 [==============================] - 1s 54ms/step - loss: 0.4228 - acc: 0.8313 - val_loss: 0.4069 - val_acc: 0.8087
Epoch 5/5
13/13 [==============================] - 1s 53ms/step - loss: 0.3559 - acc: 0.8230 - val_loss: 0.3257 - val_acc: 0.8265
<matplotlib.legend.Legend at 0x7fe88e357fd0>
```

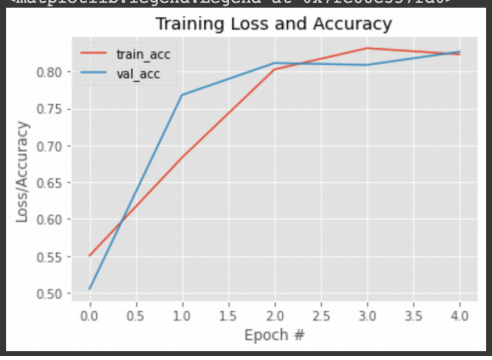


Image showing training and validation accuracy after executing benchmark

Tuning and Tuning Hyperparameters

LeNet architecture was primarily used when tuning. First data was run on a basic LeNet network, using the Adam optimizer.

```
==============================] - 6s 49ms/step - loss: 1.3597 - accuracy: 0.7054 - val_loss: 0.3656 - val_accuracy: 0.8418
/10
==============================] - 4s 42ms/step - loss: 0.3087 - accuracy: 0.8658 - val_loss: 0.2744 - val_accuracy: 0.8903
/10
==============================] - 4s 42ms/step - loss: 0.3226 - accuracy: 0.8677 - val_loss: 0.3956 - val_accuracy: 0.8087
/10
==============================] - 4s 42ms/step - loss: 0.2769 - accuracy: 0.8780 - val_loss: 0.2171 - val_accuracy: 0.9209
/10
==============================] - 4s 42ms/step - loss: 0.1728 - accuracy: 0.9342 - val_loss: 0.1592 - val_accuracy: 0.9413
/10
==============================] - 4s 42ms/step - loss: 0.0929 - accuracy: 0.9629 - val_loss: 0.1015 - val_accuracy: 0.9541
/10
==============================] - 4s 42ms/step - loss: 0.0364 - accuracy: 0.9859 - val_loss: 0.0987 - val_accuracy: 0.9745
/10
==============================] - 4s 42ms/step - loss: 0.0177 - accuracy: 0.9949 - val_loss: 0.1322 - val_accuracy: 0.9668
/10
==============================] - 4s 42ms/step - loss: 0.0199 - accuracy: 0.9936 - val_loss: 0.1818 - val_accuracy: 0.9566
0/10
==============================] - 4s 42ms/step - loss: 0.0140 - accuracy: 0.9974 - val_loss: 0.2046 - val_accuracy: 0.9541
"sequential"
```

As shown in the image after epoch 7 the validation accuracy stagnates. This could mean that overfitting is occurring. This is also shown when the optimizer is changed from Adam to SGD. Not only was overfitting occurring when SGD was used but the accuracy obtained was bad, when using SGD the learning parameter was tuned.

```
Epoch 1/10
49/49 [==============================] - 3s 55ms/step - loss: 0.6867 - accuracy: 0.6141 - val_loss: 0.6836 - val_accuracy
Epoch 2/10
49/49 [==============================] - 2s 48ms/step - loss: 0.6750 - accuracy: 0.6850 - val_loss: 0.6780 - val_accuracy
Epoch 3/10
49/49 [==============================] - 2s 48ms/step - loss: 0.6680 - accuracy: 0.6850 - val_loss: 0.6743 - val_accuracy
Epoch 4/10
49/49 [==============================] - 2s 49ms/step - loss: 0.6630 - accuracy: 0.6850 - val_loss: 0.6711 - val_accuracy
Epoch 5/10
49/49 [==============================] - 2s 49ms/step - loss: 0.6589 - accuracy: 0.6850 - val_loss: 0.6681 - val_accuracy
Epoch 6/10
49/49 [==============================] - 2s 48ms/step - loss: 0.6549 - accuracy: 0.6850 - val_loss: 0.6649 - val_accuracy
Epoch 7/10
49/49 [==============================] - 2s 48ms/step - loss: 0.6512 - accuracy: 0.6850 - val_loss: 0.6618 - val_accuracy
Epoch 8/10
49/49 [==============================] - 2s 49ms/step - loss: 0.6474 - accuracy: 0.6850 - val_loss: 0.6587 - val_accuracy
Epoch 9/10
49/49 [==============================] - 2s 49ms/step - loss: 0.6438 - accuracy: 0.6850 - val_loss: 0.6553 - val_accuracy
Epoch 10/10
49/49 [==============================] - 2s 48ms/step - loss: 0.6401 - accuracy: 0.6850 - val_loss: 0.6521 - val_accuracy
Accuracy for differnt learning rates 0.0001 is 0.6454081535339355
```

```
Epoch 1/10
49/49 [==============================] - 3s 55ms/step - loss: nan - accuracy: 0.3565 - val_loss: nan - val_accuracy: 0.35
Epoch 2/10
49/49 [==============================] - 2s 48ms/step - loss: nan - accuracy: 0.3150 - val_loss: nan - val_accuracy: 0.35
Epoch 3/10
49/49 [==============================] - 2s 48ms/step - loss: nan - accuracy: 0.3150 - val_loss: nan - val_accuracy: 0.35
Epoch 4/10
49/49 [==============================] - 2s 48ms/step - loss: nan - accuracy: 0.3150 - val_loss: nan - val_accuracy: 0.35
Epoch 5/10
49/49 [==============================] - 2s 48ms/step - loss: nan - accuracy: 0.3150 - val_loss: nan - val_accuracy: 0.35
Epoch 6/10
49/49 [==============================] - 2s 48ms/step - loss: nan - accuracy: 0.3150 - val_loss: nan - val_accuracy: 0.35
Epoch 7/10
49/49 [==============================] - 2s 48ms/step - loss: nan - accuracy: 0.3150 - val_loss: nan - val_accuracy: 0.35
Epoch 8/10
49/49 [==============================] - 2s 48ms/step - loss: nan - accuracy: 0.3150 - val_loss: nan - val_accuracy: 0.35
Epoch 9/10
49/49 [==============================] - 2s 48ms/step - loss: nan - accuracy: 0.3150 - val_loss: nan - val_accuracy: 0.35
Epoch 10/10
49/49 [==============================] - 2s 48ms/step - loss: nan - accuracy: 0.3150 - val_loss: nan - val_accuracy: 0.35
Accuracy for differnt learning rates 0.1 is 0.35459184646606445
```

```
Epoch 1/10
49/49 [==============================] - 5s 67ms/step - loss: 0.6539 - accuracy: 0.6767 - val_loss: 0.6492 - val_accuracy
Epoch 2/10
49/49 [==============================] - 2s 49ms/step - loss: 0.6127 - accuracy: 0.6984 - val_loss: 0.6110 - val_accuracy
Epoch 3/10
49/49 [==============================] - 2s 49ms/step - loss: 0.5723 - accuracy: 0.7342 - val_loss: 0.5717 - val_accuracy
Epoch 4/10
49/49 [==============================] - 2s 49ms/step - loss: 0.5337 - accuracy: 0.7591 - val_loss: 0.5804 - val_accuracy
Epoch 5/10
49/49 [==============================] - 2s 48ms/step - loss: 0.5075 - accuracy: 0.7725 - val_loss: 0.5313 - val_accuracy
Epoch 6/10
49/49 [==============================] - 2s 48ms/step - loss: 0.4891 - accuracy: 0.7783 - val_loss: 0.5283 - val_accuracy
Epoch 7/10
49/49 [==============================] - 2s 49ms/step - loss: 0.4935 - accuracy: 0.7706 - val_loss: 0.4962 - val_accuracy
Epoch 8/10
49/49 [==============================] - 2s 49ms/step - loss: 0.4841 - accuracy: 0.7725 - val_loss: 0.4980 - val_accuracy
Epoch 9/10
49/49 [==============================] - 2s 48ms/step - loss: 0.4796 - accuracy: 0.7834 - val_loss: 0.4816 - val_accuracy
Epoch 10/10
49/49 [==============================] - 2s 48ms/step - loss: 0.4686 - accuracy: 0.7802 - val_loss: 0.4953 - val_accuracy
Accuracy for differnt learning rates 0.001 is 0.7780612111091614
```

Images showing the validation and training accuracy of LeNet architecture with SGD and three different learning rates.

In order to reduce overfitting regularization and dropout were then added to the network and each parameter was tuned. The L1 and L2 parameters were tuned using for loops each L1 value was assigned and a for loop was used to test all L2 values.

```
l1=0
for l2 in [0, 0.001, 0.01, 0.1]:

  model = LeNetReg.build(224, 224, 3, 2, l1/trainX.shape[0], l2/trainX.shape[0])
  model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

  maxIt = 10
  H = model.fit(trainX, trainY, validation_split = 0.20, batch_size = 64, epochs = maxIt, verbose = 0)
  dev_accuracy = H.history['val_accuracy'][-1]
  print('Dev accuracy for l1 =', l1, ', l2 =', l2, 'is', dev_accuracy)
l1=0
for l2 in [0, 0.001, 0.01, 0.1]:

  model = LeNetReg.build(224, 224, 3, 2, l1/trainX.shape[0], l2/trainX.shape[0])
  model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
Dev accuracy for l1 = 0.1 , l2 = 0 is 0.9515306353569031
Dev accuracy for l1 = 0.1 , l2 = 0.001 is 0.9362244606018066
Dev accuracy for l1 = 0.1 , l2 = 0.01 is 0.954081654548645
Dev accuracy for l1 = 0.1 , l2 = 0.1 is 0.9668367505073547
```

```
Dev accuracy for l1 = 0.01 , l2 = 0 is 0.9413265585899353
Dev accuracy for l1 = 0.01 , l2 = 0.001 is 0.9107142686843872
Dev accuracy for l1 = 0.01 , l2 = 0.01 is 0.9464285969734192
Dev accuracy for l1 = 0.01 , l2 = 0.1 is 0.9642857313156128
```

```
Dev accuracy for l1 = 0.001 , l2 = 0 is 0.9285714030265808
Dev accuracy for l1 = 0.001 , l2 = 0.001 is 0.954081654548645
Dev accuracy for l1 = 0.001 , l2 = 0.01 is 0.9413265585899353
Dev accuracy for l1 = 0.001 , l2 = 0.1 is 0.9515306353569031
```

```
Dev accuracy for l1 = 0 , l2 = 0 is 0.954081654548645
Dev accuracy for l1 = 0 , l2 = 0.001 is 0.9515306353569031
Dev accuracy for l1 = 0 , l2 = 0.01 is 0.954081654548645
Dev accuracy for l1 = 0 , l2 = 0.1 is 0.9464285969734192
```

The same process was repeated when tuning dropout. The initial list of potential dropout values to be used consisted of larger values the highest being 0.8.Once execution was done using the

initial dropout values it was observed that using smaller values resulted in better training and validation accuracy . The best accuracy achieved when tuning dropout was 95.9% validation accuracy and 97% training accuracy.

```
10
============================] - 20s 86ms/step - loss: 1.8268 - accuracy: 0.6281 - val_loss: 0.5457 - val_accuracy: 0.6480
10
============================] - 3s 68ms/step - loss: 0.4890 - accuracy: 0.7355 - val_loss: 0.4079 - val_accuracy: 0.7934
10
============================] - 3s 67ms/step - loss: 0.3583 - accuracy: 0.8371 - val_loss: 0.3052 - val_accuracy: 0.8495
10
============================] - 3s 67ms/step - loss: 0.3246 - accuracy: 0.8581 - val_loss: 0.2510 - val_accuracy: 0.8954
10
============================] - 3s 68ms/step - loss: 0.2181 - accuracy: 0.9073 - val_loss: 0.1961 - val_accuracy: 0.9158
10
============================] - 3s 69ms/step - loss: 0.1606 - accuracy: 0.9367 - val_loss: 0.1646 - val_accuracy: 0.9362
10
============================] - 3s 68ms/step - loss: 0.1313 - accuracy: 0.9482 - val_loss: 0.1919 - val_accuracy: 0.9362
10
============================] - 3s 67ms/step - loss: 0.1002 - accuracy: 0.9636 - val_loss: 0.1697 - val_accuracy: 0.9490
10
============================] - 3s 68ms/step - loss: 0.0944 - accuracy: 0.9661 - val_loss: 0.1759 - val_accuracy: 0.9413
/10
============================] - 3s 68ms/step - loss: 0.0840 - accuracy: 0.9706 - val_loss: 0.1623 - val_accuracy: 0.9592
racy for dropout percentages [0.1, 0.2, 0.2, 0.2] is 0.9591836929321289
```

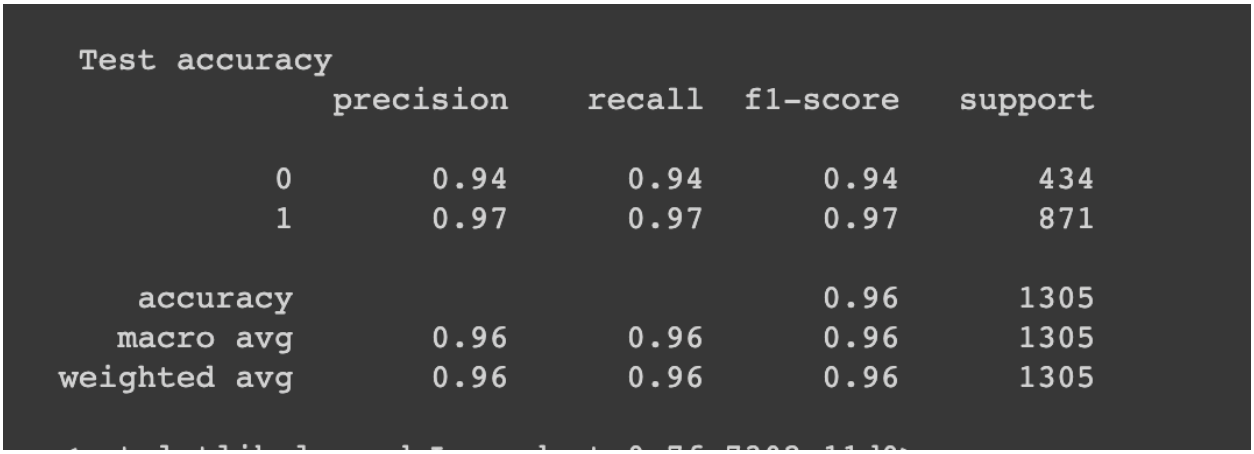
This reduced the overfitting that occured. In order to improve accuracy changes were made to the architecture. Another fully connected layer was added which significantly increased accuracy.

```
ining network...

============================] - 22s 313ms/step - loss: 4.4139 - accuracy: 0.5987 - val_loss: 0.9157 - val_accuracy: 0.7092

============================] - 3s 216ms/step - loss: 0.8868 - accuracy: 0.6754 - val_loss: 0.8506 - val_accuracy: 0.7551

============================] - 3s 216ms/step - loss: 0.7709 - accuracy: 0.7304 - val_loss: 0.7813 - val_accuracy: 0.7704

============================] - 3s 217ms/step - loss: 0.6754 - accuracy: 0.7853 - val_loss: 0.6393 - val_accuracy: 0.7730

============================] - 3s 215ms/step - loss: 0.5536 - accuracy: 0.8511 - val_loss: 0.5348 - val_accuracy: 0.8597

============================] - 3s 214ms/step - loss: 0.4606 - accuracy: 0.8824 - val_loss: 0.4264 - val_accuracy: 0.9107

============================] - 3s 242ms/step - loss: 0.3893 - accuracy: 0.9246 - val_loss: 0.4029 - val_accuracy: 0.9209

============================] - 3s 215ms/step - loss: 0.3486 - accuracy: 0.9367 - val_loss: 0.3641 - val_accuracy: 0.9362

============================] - 3s 218ms/step - loss: 0.3439 - accuracy: 0.9297 - val_loss: 0.3718 - val_accuracy: 0.9184
0
============================] - 3s 215ms/step - loss: 0.3053 - accuracy: 0.9514 - val_loss: 0.3120 - val_accuracy: 0.9541
```

Another convolution layer was added which resulted in a training accuracy of 92 % and a validation accuracy of 90% after execution.

```
==============================] - 9s 353ms/step - loss: 3.5548 - accuracy: 0.6217 - val_loss: 1.2711 - val_accuracy: 0.6633
10
==============================] - 3s 212ms/step - loss: 1.1684 - accuracy: 0.7246 - val_loss: 1.0975 - val_accuracy: 0.7347
10
==============================] - 3s 211ms/step - loss: 1.0031 - accuracy: 0.7776 - val_loss: 0.9819 - val_accuracy: 0.7704
10
==============================] - 3s 212ms/step - loss: 0.8921 - accuracy: 0.8192 - val_loss: 0.9213 - val_accuracy: 0.8138
10
==============================] - 3s 210ms/step - loss: 0.8041 - accuracy: 0.8601 - val_loss: 0.8137 - val_accuracy: 0.8571
10
==============================] - 3s 211ms/step - loss: 0.7198 - accuracy: 0.8843 - val_loss: 0.7383 - val_accuracy: 0.8929
10
==============================] - 3s 210ms/step - loss: 0.7014 - accuracy: 0.8856 - val_loss: 0.7939 - val_accuracy: 0.8240
10
==============================] - 3s 213ms/step - loss: 0.6675 - accuracy: 0.8907 - val_loss: 0.6884 - val_accuracy: 0.8954
10
==============================] - 3s 214ms/step - loss: 0.6118 - accuracy: 0.9067 - val_loss: 0.6485 - val_accuracy: 0.9107
/10
==============================] - 3s 210ms/step - loss: 0.5547 - accuracy: 0.9240 - val_loss: 0.6445 - val_accuracy: 0.9005
sequential"
```

The best performance was obtained after adding a fully connected layer. The test data was then executed on this network.



Test accuracy

	precision	recall	f1-score	support
0	0.94	0.94	0.94	434
1	0.97	0.97	0.97	871
accuracy			0.96	1305
macro avg	0.96	0.96	0.96	1305
weighted avg	0.96	0.96	0.96	1305

Conclusion:

In conclusion, we were able to classify whether an image of the parking space was occupied or not. However, the performance could be increased by increasing the number of epochs each network runs. After running the test data, it was also observed that the dataset was imbalanced which may have influenced the performance of each neural network. The confusion matrix produced, indicates that 410 of the predictions made were true positives, 850 were true negative only 26 were false positive and 24 were false negatives

```
<matplotlib.legend.Legend at 0x7fe7302c11d0>
```