Neural Networks Project 1: Classification of Parking Lot Images
Group Members: Juliann Weir-Jackson and Vedant Jani

Problem Statement:

The goal of this project was to classify images of a parking lot by two categories , spaces are empty or spaces are not empty.The dataset used was 12416 RGB images all size 640 X 640.The dataset was prelabeled and divided into three sets after downloading , train , validation and test. The sets were divided in a 70:20:10 ratio respectively .Along with the data ,two columns of labels were concatenated to the data column in each csv file.One column named ,Space empty, focusing on if there are empty spots in the image provided , this was identified through the use of 0's and 1's ,1 indicating there are empty spaces and 0 representing there are no empty spaces.The second column ,space occupied, focused on the spaces occupied , 0 representing there are no spaces occupied and 1 representing there are occupied spaces.For the scope of this project, only the space empty column was utilized.

Data Preparation:

Data was uploaded to Google Colab using

$$from\ google.colab\ import$$
$$filesuploaded\ =\ files.upload()$$

Due to the computational power available to us the entire dataset could not be uploaded as it would cause our computers to freeze, so it was decided only seventy percent (70%), 8690 images were used.Due to the amount of RAM given in Google colab it was also decided that we would batch resize the dataset before uploading. An external tool was used to resize all images in the dataset from 640 X 640 to 32 X 32. After images were uploaded functions from the opencv library were used to further preprocess each image.Each image was converted to grayscale and

flattened inorder to convert the multidimensional array to a one dimension array (see Fig.1)

```
[6]   1 trainx = []
      2
      3 for x in loaded_images:
      4   train_img =cv.cvtColor(x, cv.COLOR_BGR2GRAY)
      5   train_img=np.array(train_img).flatten()
      6   trainx.append(train_img)
      7 cv.waitKey(0)
      8 cv.destroyAllWindows()
      9
     10
```

```
[9]   1 X = np.array(trainx)
      2 print(X.shape)
```

```
(8690, 1024)
```

```
[11]  1 from google.colab import files
      2 targetup = files.upload()
```

Choose Files Target.csv

- **Target.csv**(text/csv) - 26085 bytes, last modified: 10/13/2021 - 100% done

```
Saving Target.csv to Target.csv
```

```
[12]  1 Y_t=pd.read_csv('Target.csv')
      2 Y= np.array(Y_t)
      3 print(Y.shape)
```

```
(8690, 1)
```

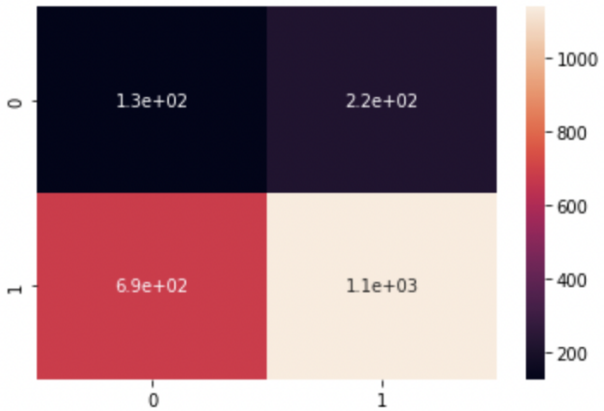
Figure 1: Images showing the preprocessing of data

BenchMarking:

The benchmark technique used was Binary Classification (Logistic regression). Once the benchmarking was executed the training accuracy was 59% and the testing accuracy was 58%.Due to time constraints and loss of data multiple times the maximum number of iterations was set to 100 which leaves the possibility of accuracy improving if the number of iterations was increased.

Performance:

```
Gradient descent failed
The norm of the gradient is 97771.07997884034

Train Classification Report:

              precision    recall  f1-score   support

           0       0.16      0.39      0.22       980
           1       0.85      0.62      0.72      5537

    accuracy                           0.59      6517
   macro avg       0.50      0.51      0.47      6517
weighted avg       0.75      0.59      0.65      6517


Test Classification Report:

              precision    recall  f1-score   support

           0       0.16      0.37      0.22       346
           1       0.84      0.62      0.72      1827

    accuracy                           0.58      2173
   macro avg       0.50      0.50      0.47      2173
weighted avg       0.73      0.58      0.64      2173


Test Confusion Matrix:

<matplotlib.axes._subplots.AxesSubplot at 0x7f88365d94d0>
```



Training and Tuning Hyperparameters:

The hyperparameters selected to be trained were : the learning rate , activation function and loss function.In Order to tune the learning rates two values were selected 0.1 and 0.001. These values were selected after running the network and observing that using values larger than 0.01 would occur in errors and values smaller than 0.001 would take an extensive amount of time.The activation functions were Sigmoid ReLU and ELU and the loss functions used were Cross Entropy on Sum of Squares.For each experiment two of hyperparameters would not change while the remaining would loop through all its possible values.Example:

```
# fit the model to the training data
model = FeedforwardNeuralNetworkSGD([1024, 32, 16, 10], 0.01, 32,'ELU','sum-of-squares',0,0)
model.fit(trainX, trainY, 100, 10)

# print the classification performance
print("Training set accuracy")
predictedY = model.predict(trainX)
predictedY = predictedY.argmax(axis=1)

trainY = trainY.argmax(axis=1)
print(classification_report(trainY, predictedY))

print("Test set accuracy")
predictedY = model.predict(testX)
predictedY = predictedY.argmax(axis=1)

testY = testY.argmax(axis=1)
print(classification_report(testY, predictedY))
```

```
# fit the model to the training data
model = FeedforwardNeuralNetworkSGD([1024, 32, 16, 10], 0.001, 32,'sigmoid','sum-of-squares',0,0)
model.fit(trainX, trainY, 100, 10)

# print the classification performance
print("Training set accuracy")
predictedY = model.predict(trainX)
predictedY = predictedY.argmax(axis=1)

trainY = trainY.argmax(axis=1)
print(classification_report(trainY, predictedY))

print("Test set accuracy")
predictedY = model.predict(testX)
predictedY = predictedY.argmax(axis=1)

testY = testY.argmax(axis=1)
print(classification_report(testY, predictedY))

```

```
# fit the model to the training data
model = FeedforwardNeuralNetworkSGD([1024, 32, 16, 10], 0.01, 32,'ReLU','sum-of-squares',0,0)
model.fit(trainX, trainY, 100, 10)

# print the classification performance
print("Training set accuracy")
predictedY = model.predict(trainX)
predictedY = predictedY.argmax(axis=1)

trainY = trainY.argmax(axis=1)
print(classification_report(trainY, predictedY))

print("Test set accuracy")
predictedY = model.predict(testX)
predictedY = predictedY.argmax(axis=1)

testY = testY.argmax(axis=1)
print(classification_report(testY, predictedY))

```

The best test accuracy was found to be 86% when using ReLU, Sum of Square Sums and 0.001 as the hyperparameters.The worst was 15% when using ReLU, Sum of Square Sums and 0.01 as the hyperparameters.

Training set accuracy

	precision	recall	f1-score	support
0	0.15	1.00	0.26	990
1	0.00	0.00	0.00	5527
6	0.00	0.00	0.00	0
accuracy			0.15	6517
macro avg	0.05	0.33	0.09	6517
weighted avg	0.02	0.15	0.04	6517

Test set accuracy

	precision	recall	f1-score	support
0	0.15	1.00	0.27	336
1	0.00	0.00	0.00	1837
3	0.00	0.00	0.00	0
6	0.00	0.00	0.00	0
accuracy			0.15	2173
macro avg	0.04	0.25	0.07	2173
weighted avg	0.02	0.15	0.04	2173

Image showing worst accuracy produced

```
Epoch = 10       loss = 904.8764878077928
Epoch = 20       loss = 865.5980392102513
Epoch = 30       loss = 893.2996711848843
Epoch = 40       loss = 867.808276363464
Epoch = 50       loss = 853.613021811521
Epoch = 60       loss = 851.1631002358471
Epoch = 70       loss = 852.6373001341469
Epoch = 80       loss = 868.6943319349931
Epoch = 90       loss = 848.8042684601987
Epoch = 100      loss = 853.5934945197731
Training set accuracy
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1016
           1       0.84      1.00      0.92      5501

    accuracy                           0.84      6517
   macro avg       0.42      0.50      0.46      6517
weighted avg       0.71      0.84      0.77      6517

Test set accuracy
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       310
           1       0.86      1.00      0.92      1863

    accuracy                           0.86      2173
   macro avg       0.43      0.50      0.46      2173
weighted avg       0.74      0.86      0.79      2173
```

Image showing best accuracy produced

Conclusion:

In conclusion , we were able to classify whether an image of the parking had empty spaces or not.The accuracy obtained was fair, however it can be drastically improved. It was observed from the loss calculated when executing the feedforward network that we may have been caught in a local minima as loss values were not decreasing but staying around the same range of values.