Project Part 1

December 3, 2019

0.1 Deep Learning Project Part2

```
Group: 3
Members:
    Juan I. Guimarey 5910690
    Thiago Barbosa PID 5470137
```

Images are going to be preprocessed on this part, model will be built on part 2
In this notebook we apply the following preprocessing steps:

- Create a Startified 13% of data in Validation Set
- Converting the Label 10's to 0's
- Greyscale conversion of image(data) for easy computation
- Normalization of data

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io import loadmat
from skimage import color
from skimage import io
from sklearn.model_selection import train_test_split

%matplotlib inline
plt.rcParams['figure.figsize'] = (16.0, 4.0)
```

LOADING DATA...

[2]:
```
def load_data(path):
    """ Helper function for loading a MAT-File"""
    data = loadmat(path)
    return data['X'], data['y']

X_train, y_train = load_data('C:\\Users\\sergi\\Documents\\CNT 4149\\Final␣
 ↪Project\\train_32x32.mat')
X_test, y_test = load_data('C:\\Users\\sergi\\Documents\\CNT 4149\\Final␣
 ↪Project\\test_32x32.mat')

print("Training Set", X_train.shape, y_train.shape)
```

```
print("Test Set", X_test.shape, y_test.shape)
```

```
Training Set (32, 32, 3, 73257) (73257, 1)
Test Set (32, 32, 3, 26032) (26032, 1)
```

Transposing the the train and test data by converting it from:
(width, height, channels, size) -> (size, width, height, channels)

[3]:
```
# Transpose the image arrays
X_train, y_train = X_train.transpose((3,0,1,2)), y_train[:,0]
X_test, y_test = X_test.transpose((3,0,1,2)), y_test[:,0]

print("Training Set", X_train.shape)
print("Test Set", X_test.shape)
print('')

# Calculate the total number of images
num_images = X_train.shape[0] + X_test.shape[0]

print("Total Number of Images", num_images)
```

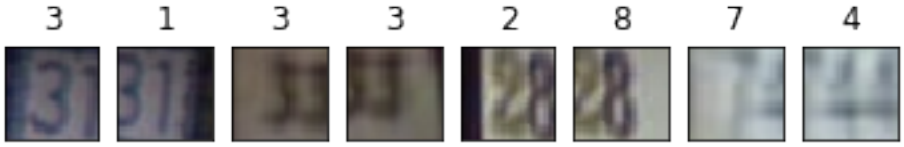
```
Training Set (73257, 32, 32, 3)
Test Set (26032, 32, 32, 3)

Total Number of Images 99289
```

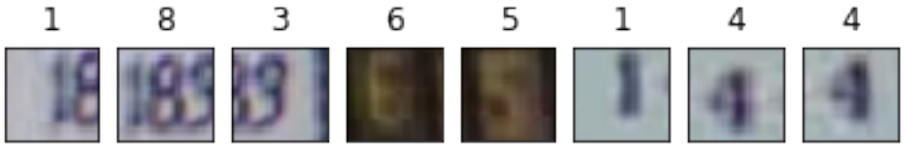
Plotting function

[4]:
```
def plot_images(img, labels, nrows, ncols):
    """ Plot nrows x ncols images
    """
    fig, axes = plt.subplots(nrows, ncols)
    for i, ax in enumerate(axes.flat):
        if img[i].shape == (32, 32, 3):
            ax.imshow(img[i])
        else:
            ax.imshow(img[i,:,:,0])
        ax.set_xticks([]); ax.set_yticks([])
        ax.set_title(labels[i])
```

[5]:
```
# Plot some training set images
plot_images(X_train, y_train, 2, 8)
```



[6]:
```
# Plot some test set images
plot_images(X_test, y_test, 2, 8)
```



Checking unique labels

[7]:
```
print(np.unique(y_train))
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

Plotting data distribution

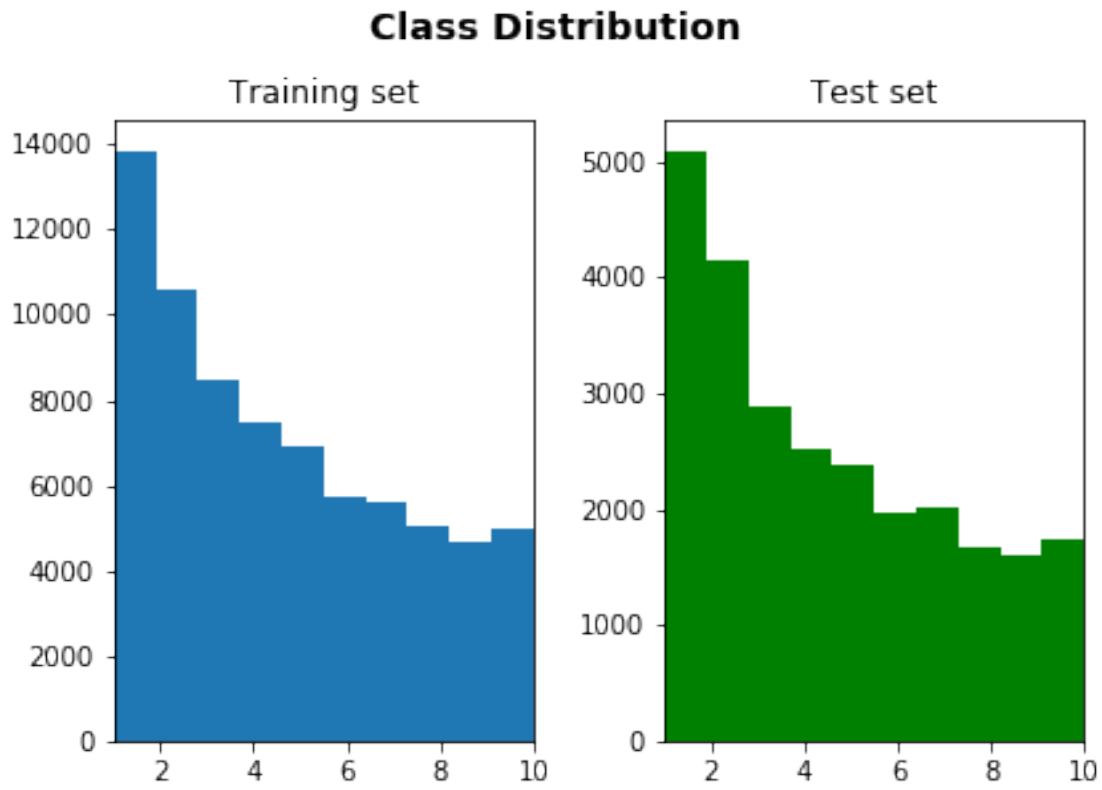
[8]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True)

fig.suptitle('Class Distribution', fontsize=14, fontweight='bold', y=1.05)
```

```
ax1.hist(y_train, bins=10)
ax1.set_title("Training set")
ax1.set_xlim(1, 10)

ax2.hist(y_test, color='g', bins=10)
ax2.set_title("Test set")

fig.tight_layout()
```



Modifying label 10 to 0

[9]:
```
y_train[y_train == 10] = 0
y_test[y_test == 10] = 0
```

[10]:
```
print(np.unique(y_train))
```

```
[0 1 2 3 4 5 6 7 8 9]
```

Splitting training data and validation data

[11]:
```
#X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,␣
 ↪test_size=0.13, random_state=7, stratify = y_train)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
 ↪13, random_state=7)
```

[12]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True)

fig.suptitle('Class Distribution', fontsize=14, fontweight='bold', y=1.05)

ax1.hist(y_train, bins=10)
ax1.set_title("Training set")
ax1.set_xlim(1, 10)

ax2.hist(y_val, color='g', bins=10)
ax2.set_title("Validation set")

fig.tight_layout()
```

[13]:
```
y_train.shape, y_val.shape, y_test.shape
```

[13]:
```
((63733,), (9524,), (26032,))
```

Function to convert to greyscale

[14]:
```
def rgb2gray(images):
    return np.expand_dims(np.dot(images, [0.2990, 0.5870, 0.1140]), axis=3)
```

[15]:
```
train_greyscale = rgb2gray(X_train).astype(np.float32)
test_greyscale = rgb2gray(X_test).astype(np.float32)
```

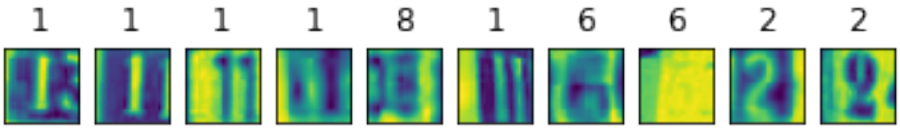
```
val_greyscale = rgb2gray(X_val).astype(np.float32)
print("Training Set", train_greyscale.shape)
print("Validation Set", val_greyscale.shape)
print("Test Set", test_greyscale.shape)
print('')
```

```
Training Set (63733, 32, 32, 1)
Validation Set (9524, 32, 32, 1)
Test Set (26032, 32, 32, 1)
```

Deleting old rgb data

[16]:
```
del X_train, X_test, X_val
```

[17]:
```
plot_images(train_greyscale, y_train, 1, 10)
```



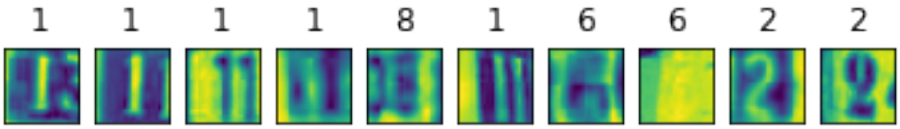
Normalizing the data

[18]:
```
# Calculate the mean on the training data
train_mean = np.mean(train_greyscale, axis=0)

# Calculate the std on the training data
train_std = np.std(train_greyscale, axis=0)

# Subtract it equally from all splits
train_greyscale_norm = (train_greyscale - train_mean) / train_std
test_greyscale_norm = (test_greyscale - train_mean)  / train_std
val_greyscale_norm = (val_greyscale - train_mean) / train_std
```

[19]:
```
plot_images(train_greyscale_norm, y_train, 1, 10)
```



One Hot label encoding

[20]:
```
from sklearn.preprocessing import OneHotEncoder

# Fit the OneHotEncoder
enc = OneHotEncoder().fit(y_train.reshape(-1, 1))

# Transform the label values to a one-hot-encoding scheme
y_train = enc.transform(y_train.reshape(-1, 1)).toarray()
y_test = enc.transform(y_test.reshape(-1, 1)).toarray()
y_val = enc.transform(y_val.reshape(-1, 1)).toarray()

print("Training set", y_train.shape)
print("Validation set", y_val.shape)
print("Test set", y_test.shape)
```

```
Training set (63733, 10)
Validation set (9524, 10)
Test set (26032, 10)

C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\preprocessing\_encoders.py:415: FutureWarning: The handling of
integer data will change in version 0.22. Currently, the categories are
determined based on the range [0, max(values)], while in the future they will be
determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the
categories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
```