```
import numpy as np
import matplotlib.pyplot as plt
import keras
import cv2
from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Dense
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import pickle
import random
import pandas as pd
```

```
Using TensorFlow backend.
```

```
np.random.seed(0)
```

Impoting Data

```
!git clone https://bitbucket.org/jadslim/german-traffic-signs
```

```
Cloning into 'german-traffic-signs'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
```

```
with open('german-traffic-signs/train.p','rb') as f:
    train_data = pickle.load(f)
with open('german-traffic-signs/valid.p','rb') as f:
    val_data = pickle.load(f)
with open('german-traffic-signs/test.p','rb') as f:
    test_data = pickle.load(f)
```

```
X_train, y_train = train_data['features'], train_data['labels']
X_val, y_val = val_data['features'], val_data['labels']
X_test, y_test = test_data['features'], test_data['labels']
```

```
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(34799, 32, 32, 3)
(4410, 32, 32, 3)
(12630, 32, 32, 3)
```

```
assert(X_train.shape[0] == y_train.shape[0]), "The number of images is not equal to the nu
assert(X_val.shape[0] == y_val.shape[0]), "The number of images is not equal to the number
```

```
assert(X_test.shape[0] == y_test.shape[0]), "The number of images is not equal to the numb
assert(X_train.shape[1:] == (32, 32, 3)), "The dimensions of the image is not 32*32*3"
assert(X_val.shape[1:] == (32, 32, 3)), "The dimensions of the image is not 32*32*3"
assert(X_test.shape[1:] == (32, 32, 3)), "The dimensions of the image is not 32*32*3"
```

Data Visualisation

```
data = pd.read_csv('german-traffic-signs/signnames.csv')

num_of_samples = []

cols = 5
num_classes = 43

fig, axs = plt.subplots(nrows = num_classes, ncols = cols, figsize = (5, 50))
fig.tight_layout()

for i in range(cols):
    for j, row in data.iterrows():
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(0, (len(x_selected)-1)), :, :], cmap =
        axs[j][i].axis("off")
        if i == 2:
            axs[j][i].set_title(str(j) + "_" + row["SignName"])
            num_of_samples.append(len(x_selected))
```

0_Speed limit (20km/h)

1_Speed limit (30km/h)

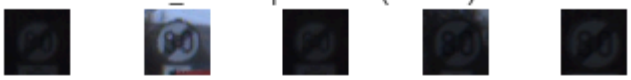
2_Speed limit (50km/h)

3_Speed limit (60km/h)

4_Speed limit (70km/h)

5_Speed limit (80km/h)

6_End of speed limit (80km/h)



7_Speed limit (100km/h)

8_Speed limit (120km/h)

9_No passing

10_No passing for vechiles over 3.5 metric tons

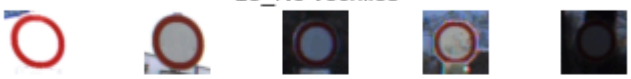
11_Right-of-way at the next intersection

12_Priority road

13_Yield

14_Stop

15_No vechiles



16_Vechiles over 3.5 metric tons prohibited

17_No entry

18_General caution

19_Dangerous curve to the left

20_Dangerous curve to the right

21_Double curve

22_Bumpy road

23_Slippery road

24_Road narrows on the right

25_Road work

26_Traffic signals

27_Pedestrians

28_Children crossing

29_Bicycles crossing

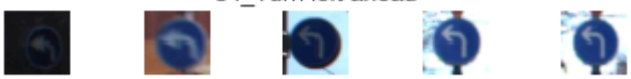
30_Beware of ice/snow

31_Wild animals crossing

32_End of all speed and passing limits

33_Turn right ahead

34_Turn left ahead



35_Ahead only

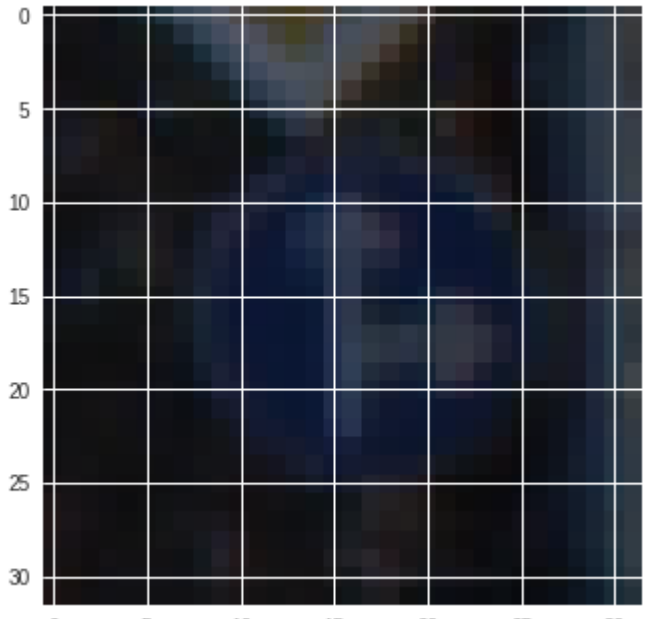

```
print(num_of_samples)
plt.figure(figsize = (12, 4))
plt.bar(range(0, num_classes), num_of_samples)
plt.title("Training Dataset Distribution")
plt.xlabel("Class number")
plt.ylabel("Number of images")
```

```
[180, 1980, 2010, 1260, 1770, 1650, 360, 1290, 1260, 1320, 1800, 1170, 1890, 19
Text(0,0.5,'Number of images')
```

Data Preprocessing

```
plt.imshow(X_train[1000])
plt.axis('off')
print(X_train[1000].shape)
print(y_train[1000])
```

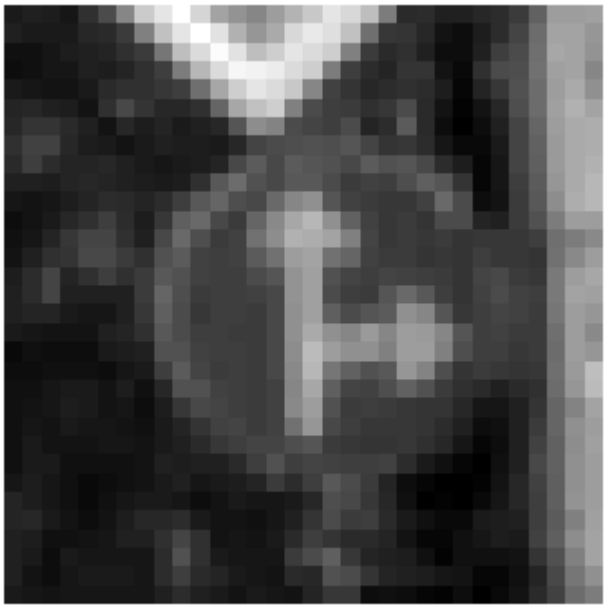
```
(32, 32, 3)
36
```




```
def grayscale(img):
    image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    plt.axis('off')
    return image


img = grayscale(X_train[1000])
plt.imshow(img, cmap = 'gray')
print(img.shape)
```

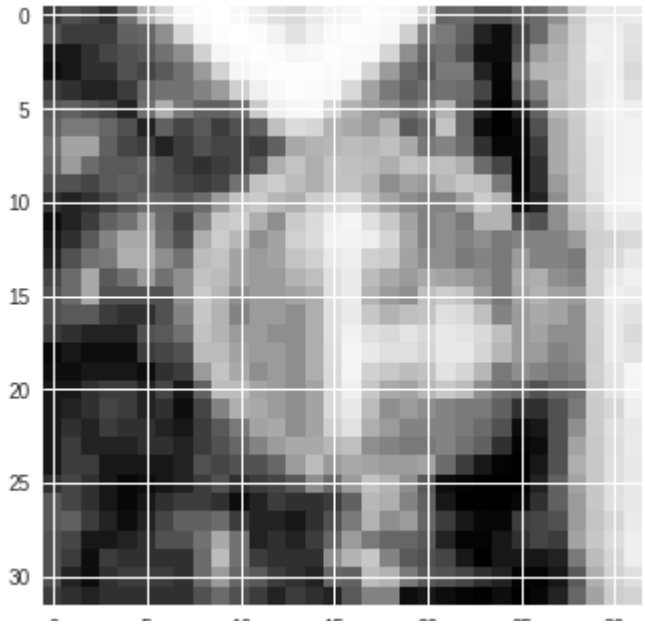
```
(32, 32)
```



```
def equalize(img):
    img = cv2.equalizeHist(img)
    return img
```

```
img = equalize(img)
plt.imshow(img, cmap = 'gray')
plt.axis('off')
print(img.shape)
```

```
(32, 32)
```




```
def preprocessing(img):
    img = grayscale(img)
    img = equalize(img)
    img = img/255
    return img
```

```
X_train = np.array(list(map(preprocessing, X_train)))
X_val = np.array(list(map(preprocessing, X_val)))
X_test = np.array(list(map(preprocessing, X_test)))
```

```
X_train = X_train.reshape(34799, 32, 32, 1)
X_val = X_val.reshape(4410, 32, 32, 1)
X_test = X_test.reshape(12630, 32, 32, 1)


from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(width_shift_range = 0.1,
                  height_shift_range = 0.1,
                   zoom_range = 0.2,
                  shear_range = 0.1,
                  rotation_range = 10)

datagen.fit(X_train)


batches = datagen.flow(X_train, y_train, batch_size = 20)
X_batch, y_batch = next(batches)

fig, axs = plt.subplots(1, 15, figsize = (20, 5))
fig.tight_layout()

for i in range(15):
  axs[i].imshow(X_batch[i].reshape(32, 32))
  axs[i].axis('off')
```

```
y_train = to_categorical(y_train, 43)
y_val = to_categorical(y_val, 43)
y_test = to_categorical(y_test, 43)
```

Neural Network

```
def neural_model():
    model = Sequential()
    model.add(Conv2D(60, (5, 5), input_shape = (32, 32, 1), activation = 'relu'))
    model.add(Conv2D(60, (5, 5), input_shape = (32, 32, 1), activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))

    model.add(Conv2D(30, (3, 3), activation = 'relu'))
    model.add(Conv2D(30, (3, 3), activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2, 2)))

    #model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(500, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation = 'softmax'))
```

```
    model.compile(Adam(lr = 0.001), loss = 'categorical_crossentropy', metrics = ['accurac
    return model


model = neural_model()
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 60)        1560
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 60)        90060
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 60)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 30)        16230
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 30)          8130
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 30)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 480)               0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               240500
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 43)                21543
=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
_________________________________________________________________
None
```

```
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size = 50), steps_per_e
```

```
Epoch 1/10
2000/2000 [==============================] - 59s 29ms/step - loss: 0.8139 - acc
Epoch 2/10
2000/2000 [==============================] - 54s 27ms/step - loss: 0.1934 - acc
Epoch 3/10
2000/2000 [==============================] - 54s 27ms/step - loss: 0.1325 - acc
Epoch 4/10
2000/2000 [==============================] - 54s 27ms/step - loss: 0.1045 - acc
Epoch 5/10
2000/2000 [==============================] - 54s 27ms/step - loss: 0.0901 - acc
Epoch 6/10
2000/2000 [==============================] - 54s 27ms/step - loss: 0.0777 - acc
Epoch 7/10
2000/2000 [==============================] - 54s 27ms/step - loss: 0.0695 - acc
Epoch 8/10
2000/2000 [==============================] - 54s 27ms/step - loss: 0.0642 - acc
Epoch 9/10
2000/2000 [==============================] - 54s 27ms/step - loss: 0.0538 - acc
```