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

```
Cloning into 'german-traffic-signs'...
Unpacking objects: 100% (6/6), done.
```

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

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

```
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:
    valid_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_valid,y_valid = valid_data['features'],valid_data['labels']
X_test,y_test = test_data['features'],test_data['labels']
```

```
print(X_train.shape)
print(X_valid.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 number of labels"
assert(X_valid.shape[0]==y_valid.shape[0]),"The number of images is not equal to the number of labels"
assert(X_test.shape[0]==y_test.shape[0]),"The number of images is not equal to the number of labels"
assert(X_train.shape[1:]==(32,32,3)),"The dimensions of the images are not 32 x 32 x 3"
assert(X_valid.shape[1:]==(32,32,3)),"The dimensions of the images are not 32 x 32 x 3"
assert(X_test.shape[1:]==(32,32,3)),"The dimensions of the images are not 32 x 32 x 3"
```

```
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=plt.get_cmap('gray'))
      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

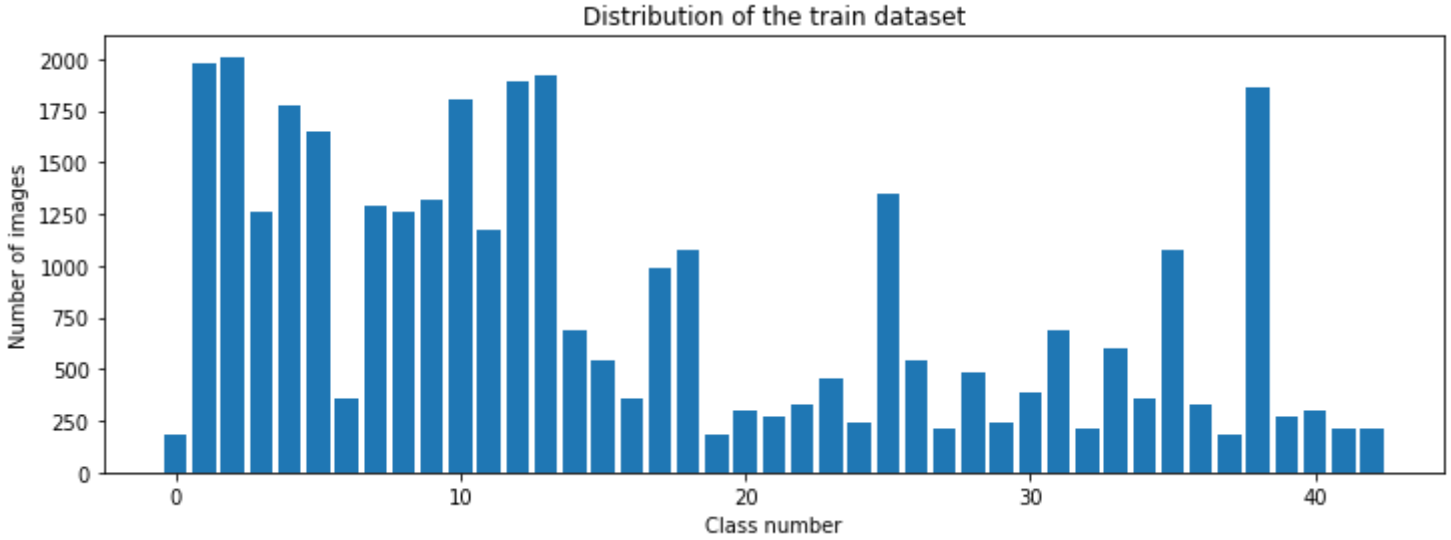
35-Ahead only

36-Go straight or right

37-Go straight or left


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

```
[180, 1980, 2010, 1260, 1770, 1650, 360, 1290, 1260, 1320, 1800, 1170, 1890, 1920, 690, 540, 360, 990, 1080, 180, 300, 270, 330, 450, 240
```



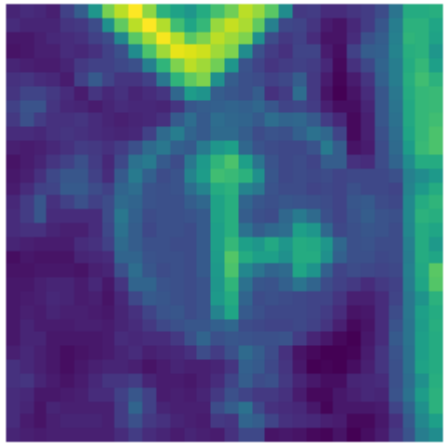

```
import cv2

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

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

```
def grayscale(img):
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    return img
```

```
img = grayscale(X_train[1000])
plt.imshow(img)
plt.axis('off')
print(img.shape)
```

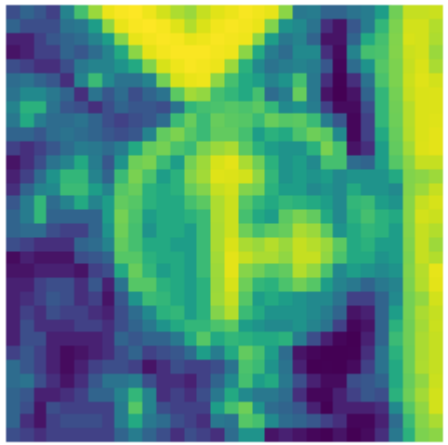
```
(32, 32)
```



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

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

```
(32, 32)
```

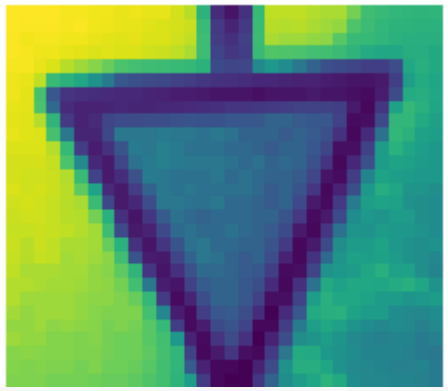


```
def preprocessing(img):
  img = grayscale(img)
  img = equalize(img)
  img = img/255
  return img
```

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

```
plt.imshow(X_train[random.randint(0,len(X_train)-1)])
plt.axis('off')
print(X_train.shape)
```

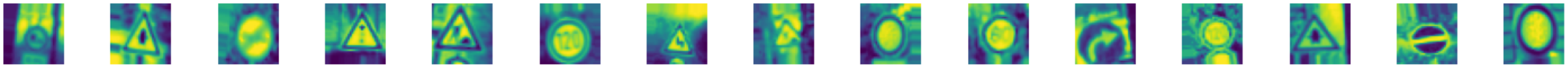
```
(34799, 32, 32)
```



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

```
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=15)
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')
```



```
print(X_batch.shape)
```

```
(15, 32, 32, 1)
```

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

```
def modified_model():
  model = Sequential()

  model.add(Conv2D(60,(5,5),input_shape=(32,32,1),activation='relu'))
  model.add(Conv2D(60,(5,5),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(Flatten())
  model.add(Dense(500,activation='relu'))
  # model.add(Dropout(0.5))
  model.add(Dense(num_classes,activation='softmax'))

  model.compile(Adam(learning_rate=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
  return model
```

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

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 28, 28, 60)        1560

 conv2d_9 (Conv2D)           (None, 24, 24, 60)        90060

 max_pooling2d_4 (MaxPooling  (None, 12, 12, 60)       0
```

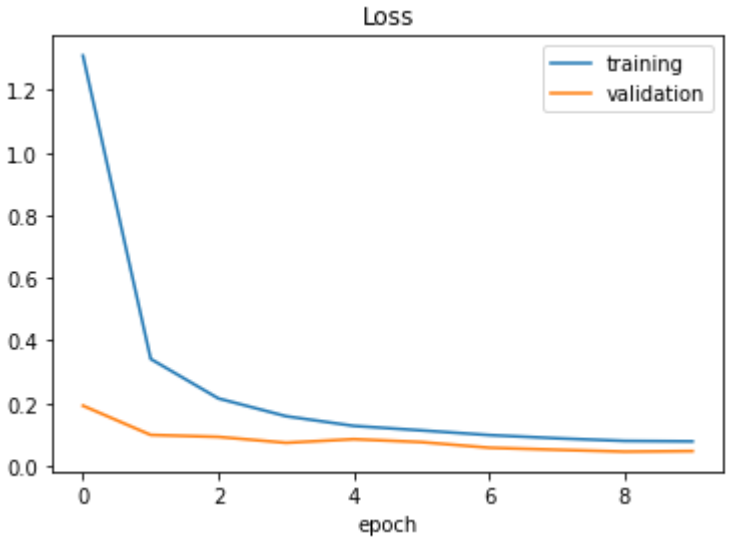
```
 2D)

 conv2d_10 (Conv2D)          (None, 10, 10, 30)        16230

 conv2d_11 (Conv2D)          (None, 8, 8, 30)          8130

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 30)         0
 2D)

 flatten_2 (Flatten)         (None, 480)               0

 dense_4 (Dense)             (None, 500)               240500

 dense_5 (Dense)             (None, 43)                21543

=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
_________________________________________________________________
None
```

```
history = model.fit(datagen.flow(X_train,y_train,batch_size=50),epochs=10,validation_data=(X_valid, y_valid),verbose=1,shuffle=1)
```

```
Epoch 1/10
696/696 [==============================] - 14s 19ms/step - loss: 1.3104 - accuracy: 0.6236 - val_loss: 0.1919 - val_accuracy: 0.9429
Epoch 2/10
696/696 [==============================] - 13s 19ms/step - loss: 0.3413 - accuracy: 0.8949 - val_loss: 0.0985 - val_accuracy: 0.9685
Epoch 3/10
696/696 [==============================] - 13s 19ms/step - loss: 0.2150 - accuracy: 0.9338 - val_loss: 0.0922 - val_accuracy: 0.9732
Epoch 4/10
696/696 [==============================] - 14s 20ms/step - loss: 0.1582 - accuracy: 0.9511 - val_loss: 0.0734 - val_accuracy: 0.9780
Epoch 5/10
696/696 [==============================] - 14s 20ms/step - loss: 0.1273 - accuracy: 0.9599 - val_loss: 0.0849 - val_accuracy: 0.9748
Epoch 6/10
696/696 [==============================] - 13s 19ms/step - loss: 0.1129 - accuracy: 0.9646 - val_loss: 0.0755 - val_accuracy: 0.9771
Epoch 7/10
696/696 [==============================] - 14s 20ms/step - loss: 0.0980 - accuracy: 0.9694 - val_loss: 0.0576 - val_accuracy: 0.9832
Epoch 8/10
696/696 [==============================] - 13s 19ms/step - loss: 0.0877 - accuracy: 0.9725 - val_loss: 0.0511 - val_accuracy: 0.9844
Epoch 9/10
696/696 [==============================] - 13s 19ms/step - loss: 0.0794 - accuracy: 0.9751 - val_loss: 0.0451 - val_accuracy: 0.9880
Epoch 10/10
696/696 [==============================] - 14s 20ms/step - loss: 0.0778 - accuracy: 0.9757 - val_loss: 0.0469 - val_accuracy: 0.9889
```

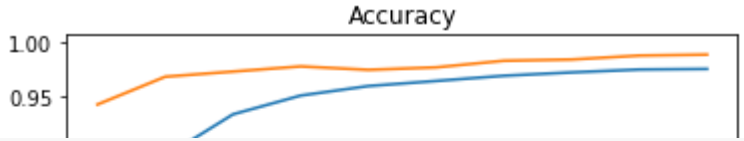
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training','validation'])
plt.title('Loss')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title('Accuracy')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```




```
score = model.evaluate(X_test,y_test,verbose=0)
print('Test Score',score[0])
print('Test Accuracy',score[1])
```

```
Test Score 0.15319083631038666
Test Accuracy 0.9634996056556702
```


```
import requests
from PIL import Image
url1 = 'https://c8.alamy.com/comp/G667W0/road-sign-speed-limit-30-kmh-zone-passau-bavaria-germany-G667W0.jpg' #1
url2 = 'https://c8.alamy.com/comp/A0RX23/cars-and-automobiles-must-turn-left-ahead-sign-A0RX23.jpg' #34
url3 = 'https://previews.123rf.com/images/bwylezich/bwylezich1608/bwylezich160800375/64914157-german-road-sign-slippery-road.jpg' #23
url4 = 'https://previews.123rf.com/images/pejo/pejo0907/pejo090700003/5155701-german-traffic-sign-no-205-give-way.jpg' #13
url5 = 'https://c8.alamy.com/comp/J2MRAJ/german-road-sign-bicycles-crossing-J2MRAJ.jpg' #29
```

```
r = requests.get(url1, stream=True)
img = Image.open(r.raw)
plt.imshow(img, cmap=plt.get_cmap('gray'))
img = np.asarray(img)
img = cv2.resize(img, (32, 32))
img = preprocessing(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
```

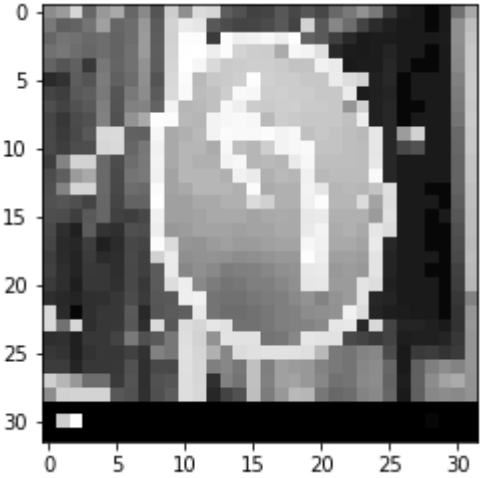
```
(32, 32)
```


```
img = img.reshape(1, 32, 32, 1)
print("Predicted sign: "+ str(np.argmax(model.predict(img), axis=-1)))
```

```
Predicted sign: [1]
```

```
r = requests.get(url2, stream=True)
img = Image.open(r.raw)
plt.imshow(img, cmap=plt.get_cmap('gray'))
img = np.asarray(img)
img = cv2.resize(img, (32, 32))
img = preprocessing(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
```

```
(32, 32)
```



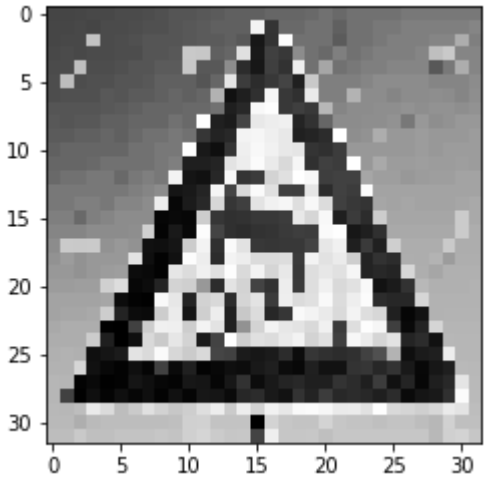

```
img = img.reshape(1, 32, 32, 1)
print("Predicted sign: "+ str(np.argmax(model.predict(img), axis=-1)))
```

```
Predicted sign: [34]
```

```
r = requests.get(url3, stream=True)
```

```
img = Image.open(r.raw)
plt.imshow(img, cmap=plt.get_cmap('gray'))
img = np.asarray(img)
img = cv2.resize(img, (32, 32))
img = preprocessing(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
```

```
(32, 32)
```

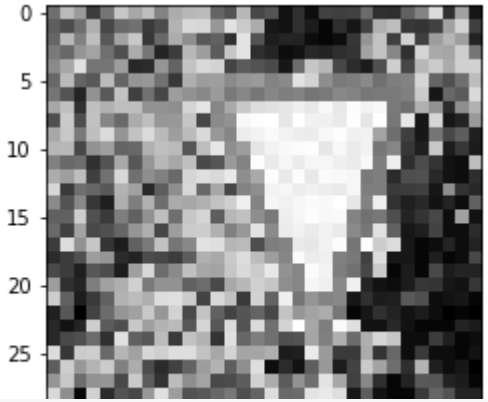



```
img = img.reshape(1, 32, 32, 1)
print("Predicted sign: "+ str(np.argmax(model.predict(img), axis=-1)))
```

```
Predicted sign: [23]
```

```
r = requests.get(url4, stream=True)
img = Image.open(r.raw)
plt.imshow(img, cmap=plt.get_cmap('gray'))
img = np.asarray(img)
img = cv2.resize(img, (32, 32))
img = preprocessing(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
```

```
(32, 32)
```

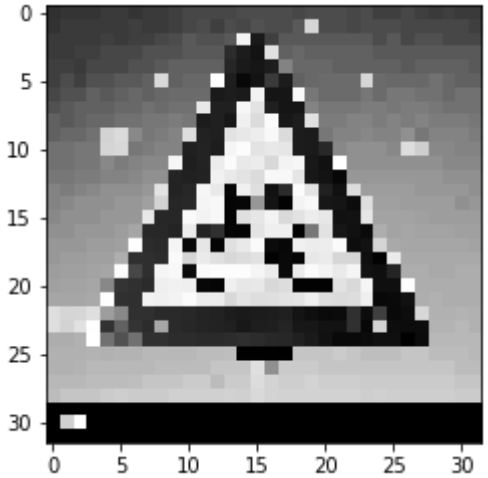



```
img = img.reshape(1, 32, 32, 1)
print("Predicted sign: "+ str(np.argmax(model.predict(img), axis=-1)))
```

```
Predicted sign: [13]
```

```
r = requests.get(url5, stream=True)
img = Image.open(r.raw)
plt.imshow(img, cmap=plt.get_cmap('gray'))
img = np.asarray(img)
img = cv2.resize(img, (32, 32))
img = preprocessing(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
```

```
(32, 32)
```



```
img = img.reshape(1, 32, 32, 1)
print("Predicted sign: "+ str(np.argmax(model.predict(img), axis=-1)))
```

```
Predicted sign: [29]
```