```
!git clone https://bitbucket.org/Pranit570/roadsigndetectiongerman
```

```
!ls roadsigndetectiongerman
```

```
signnames.csv  test.p  train.p  valid.p
```

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


from keras.callbacks import LearningRateScheduler, ModelCheckpoint
```

```
%matplotlib inline
np.random.seed(0)
# TODO: Implement load the data here.
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)
# TODO: Load test data
with open('german-traffic-signs/test.p', 'rb') as f:
    test_data = pickle.load(f)
```

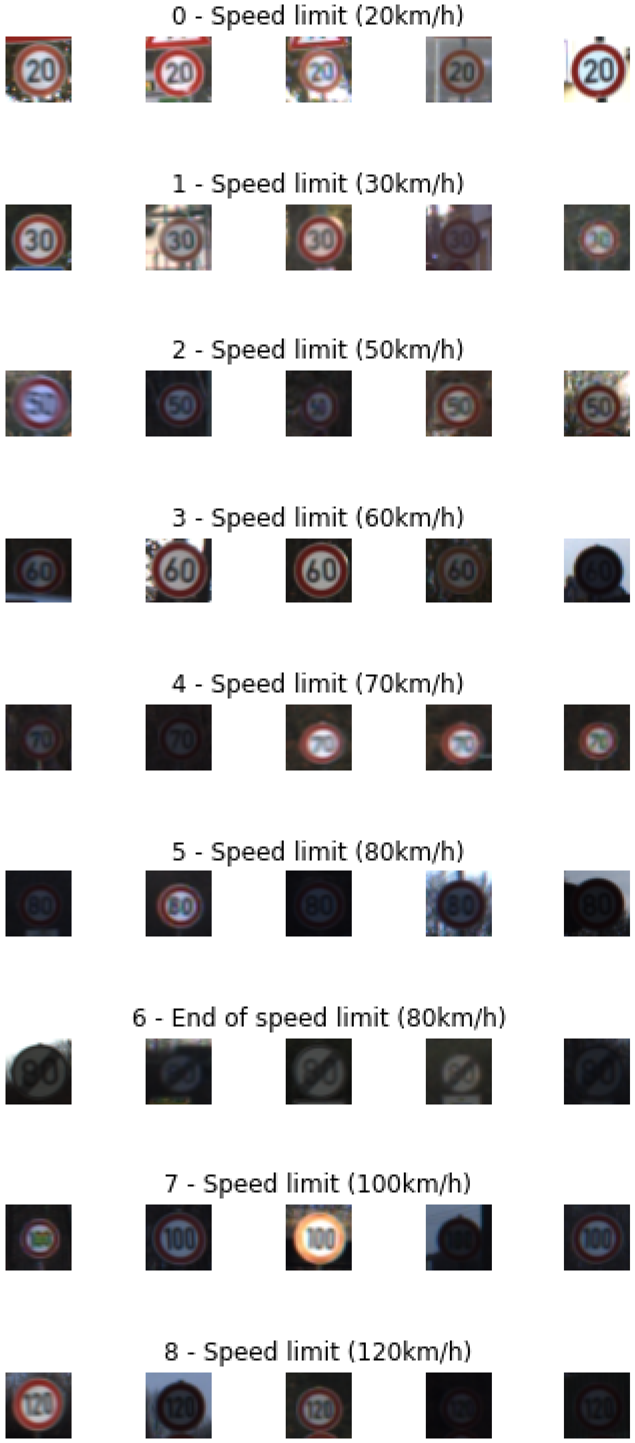
```
# Split out features and labels
X_train, y_train = train_data['features'], train_data['labels']
X_val, y_val = val_data['features'], val_data['labels']
X_test, y_test = test_data['features'], test_data['labels']

#already 4 dimensional
print(X_train.shape)
print(X_test.shape)
print(X_val.shape)
```

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

```
# STOP: Do not change the tests below. Your implementation should pass these tests.
assert(X_train.shape[0] == y_train.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_val.shape[0] == y_val.shape[0]), "The number of images is not equal to the number of labels."
assert(X_val.shape[1:] == (32,32,3)), "The dimensions of the images are not 32 x 32 x 3."
assert(X_test.shape[0] == y_test.shape[0]), "The number of images is not equal to the number of labels."
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))
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()
import cv2

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

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



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)

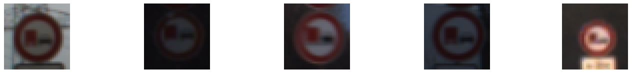
6 - End of speed limit (80km/h)

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

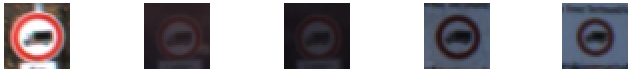
12 - Priority road

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

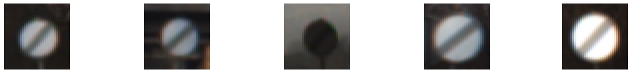
28 - Children crossing

29 - Bicycles crossing

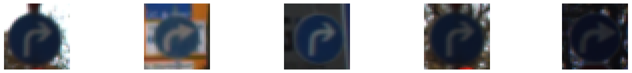
30 - Beware of ice/snow

31 - Wild animals crossing

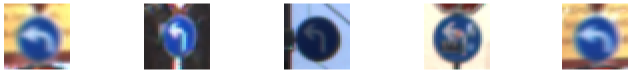
32 - End of all speed and passing limits



33 - Turn right ahead



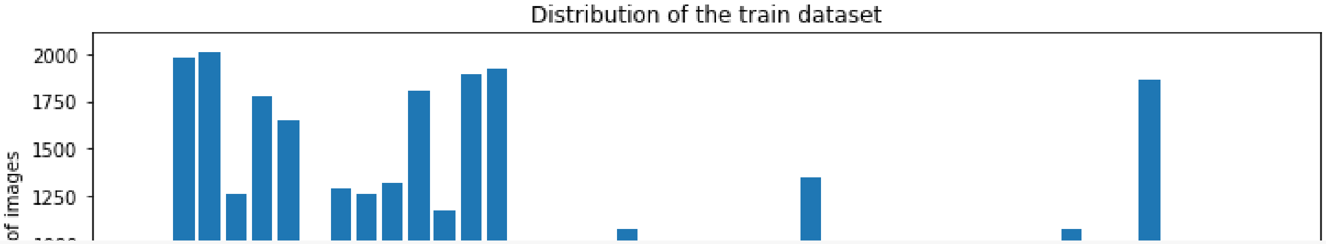
34 - Turn left ahead



35 - Ahead only

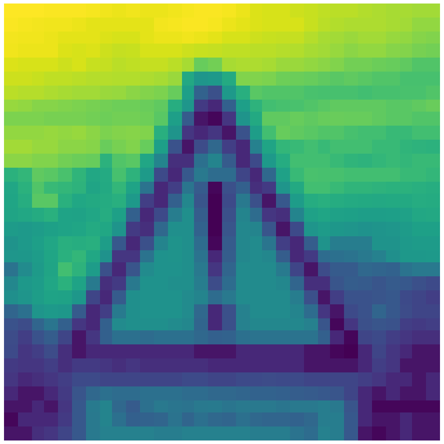
36 - Go straight or right

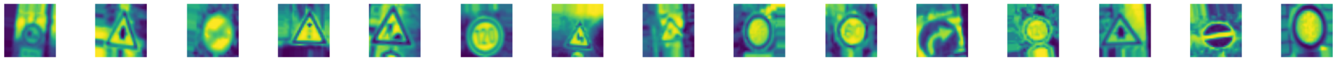
37 - Go straight or left
38 - Keep right
39 - Keep left
40 - Roundabout mandatory
41 - End of no passing
42 - End of no passing by vechiles over 3.5 metric tons



```
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)
def equalize(img):
    img = cv2.equalizeHist(img)
    img = cv2.equalizeHist(img)
    return img
img = equalize(img)
plt.imshow(img)
plt.axis("off")
print(img.shape)
def preprocess(img):
    img = grayscale(img)
    img = equalize(img)
    img = img/255
    return img

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

plt.imshow(X_train[random.randint(0, len(X_train) - 1)])
plt.axis('off')
print(X_train.shape)
X_train = X_train.reshape(34799, 32, 32, 1)
X_test = X_test.reshape(12630, 32, 32, 1)
X_val = X_val.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)
# for X_batch, y_batch in

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")
    axs[1].axis('off')

print(X_batch.shape)
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
y_val = to_categorical(y_val, 43)
```

```
(32, 32)
(32, 32)
(34799, 32, 32)
(15, 32, 32, 1)
```





```
# create model

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(43, activation='softmax'))

  model.compile(Adam(lr = 0.001), loss='categorical_crossentropy', metrics=['accuracy'])
  return model
model = modified_model()
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 60)        1560
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 60)        90060
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 60)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 30)        16230
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 30)          8130
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 30)          0
_________________________________________________________________
flatten (Flatten)            (None, 480)               0
_________________________________________________________________
dense (Dense)                (None, 500)               240500
_________________________________________________________________
dropout (Dropout)            (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 43)                21543
=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
_________________________________________________________________
None
```

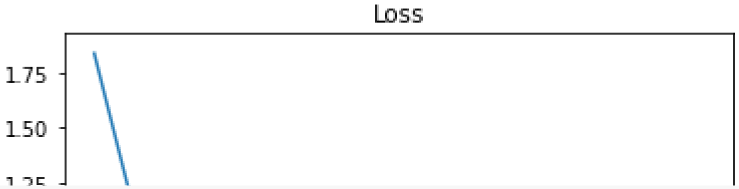
```
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=50),
                            steps_per_epoch=600,
                            epochs=10,
                            validation_data=(X_val, y_val), shuffle = 1)
```

```
                        validation_data=(x_val, y_val), shuffle = 1)
```

```
WARNING:tensorflow:From <ipython-input-11-29a595a3013d>:4: Model.fit_generator (from tensorflow.python.keras.engine.training) is
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/10
600/600 [==============================] - 313s 522ms/step - loss: 1.8384 - accuracy: 0.4754 - val_loss: 0.3839 - val_accuracy: 0
Epoch 2/10
600/600 [==============================] - 321s 535ms/step - loss: 0.6219 - accuracy: 0.8071 - val_loss: 0.1617 - val_accuracy: 0
Epoch 3/10
600/600 [==============================] - 354s 590ms/step - loss: 0.3907 - accuracy: 0.8779 - val_loss: 0.1038 - val_accuracy: 0
Epoch 4/10
600/600 [==============================] - 348s 579ms/step - loss: 0.2891 - accuracy: 0.9089 - val_loss: 0.0855 - val_accuracy: 0
Epoch 5/10
600/600 [==============================] - 348s 580ms/step - loss: 0.2421 - accuracy: 0.9245 - val_loss: 0.0730 - val_accuracy: 0
Epoch 6/10
600/600 [==============================] - 349s 582ms/step - loss: 0.2095 - accuracy: 0.9349 - val_loss: 0.0761 - val_accuracy: 0
Epoch 7/10
600/600 [==============================] - 354s 590ms/step - loss: 0.1782 - accuracy: 0.9450 - val_loss: 0.0581 - val_accuracy: 0
Epoch 8/10
600/600 [==============================] - 348s 579ms/step - loss: 0.1572 - accuracy: 0.9520 - val_loss: 0.0486 - val_accuracy: 0
Epoch 9/10
600/600 [==============================] - 349s 582ms/step - loss: 0.1505 - accuracy: 0.9535 - val_loss: 0.0423 - val_accuracy: 0
Epoch 10/10
600/600 [==============================] - 351s 585ms/step - loss: 0.1414 - accuracy: 0.9563 - val_loss: 0.0436 - val_accuracy: 0
```

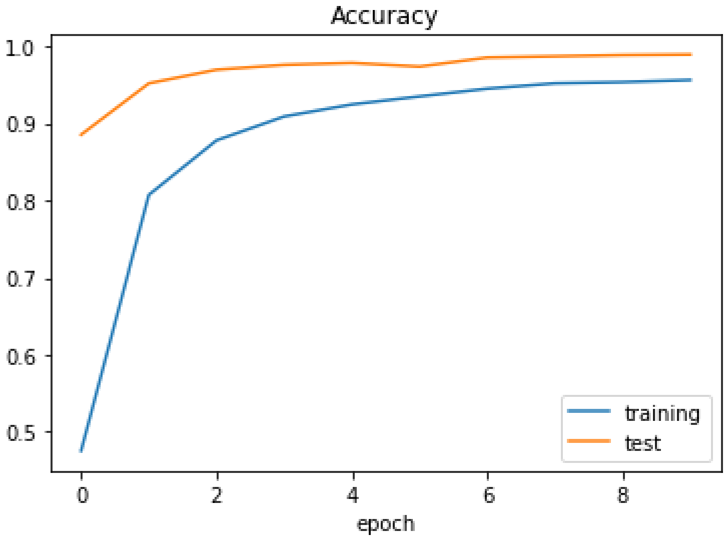
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss')
plt.xlabel('epoch')
```

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



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

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



```
# TODO: Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

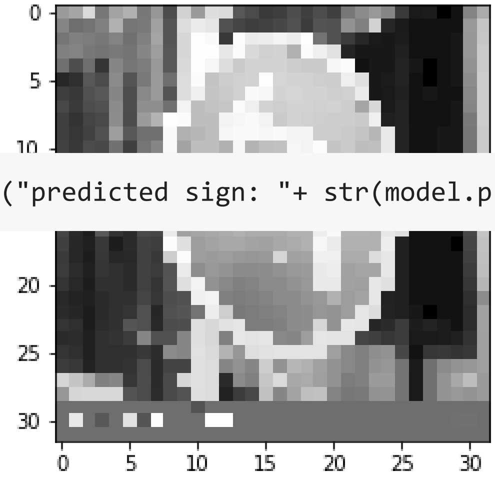
```
Test score: 0.11179137974977493
Test accuracy: 0.9713380932807922
```

```
#predict internet number
import requests
import requests
from PIL import Image
url = 'https://c8.alamy.com/comp/A0RX23/cars-and-automobiles-must-turn-left-ahead-sign-A0RX23.jpg'
r = requests.get(url, stream=True)
img = Image.open(r.raw)
plt.imshow(img, cmap=plt.get_cmap('gray'))
```

```
<matplotlib.image.AxesImage at 0x7f11924c0400>
```

```
img = np.asarray(img)
img = cv2.resize(img, (32, 32))
img = preprocess(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
img = img.reshape(1, 32, 32, 1)
```

```
(32, 32)
```



```
print("predicted sign: "+ str(model.predict_classes(img)))
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.