In [ ]:

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

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

In [ ]:

```
!ls german-traffic-sign
```

```
ls: cannot access 'german-traffic-sign': No such file or directory
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from tensorflow.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
```

In [ ]:

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

In [ ]:

```
# 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)
```

In [ ]:

```
# 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)
```

In [ ]:

```
# 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."
```

In [ ]:

```
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)

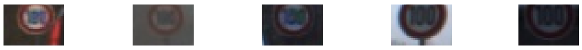
3 - Speed limit (60km/h)

4 - Speed limit (70km/h)

5 - Speed limit (80km/h)

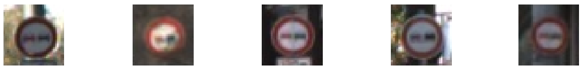
6 - End of speed limit (80km/h)

7 - Speed limit (100km/h)



8 - Speed limit (120km/h)

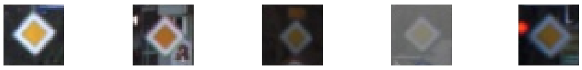
9 - No passing



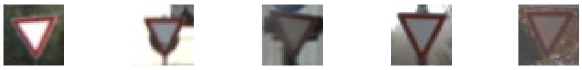
10 - No passing for vechiles over 3.5 metric tons

11 - Right-of-way at the next intersection

12 - Priority road



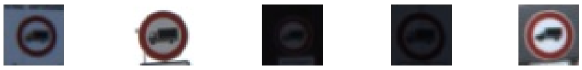
13 - Yield



14 - Stop

15 - No vechiles

16 - Vechiles over 3.5 metric tons prohibited

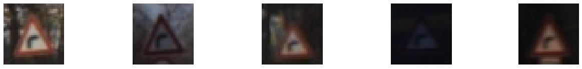


17 - No entry

18 - General caution

19 - Dangerous curve to the left

20 - Dangerous curve to the right



21 - Double curve

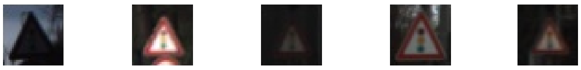
22 - Bumpy road

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

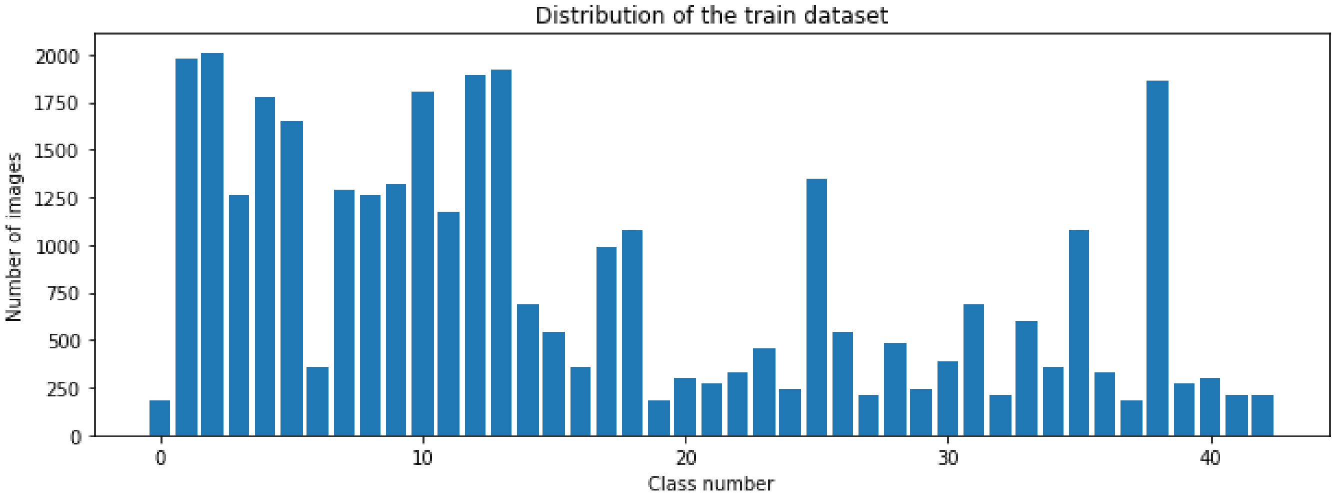
36 - Go straight or right

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

In [ ]:

```
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, 5
40, 360, 990, 1080, 180, 300, 270, 330, 450, 240, 1350, 540, 210, 480, 240, 390, 690, 210
, 599, 360, 1080, 330, 180, 1860, 270, 300, 210, 210]
```



In [ ]:

```
import cv2

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

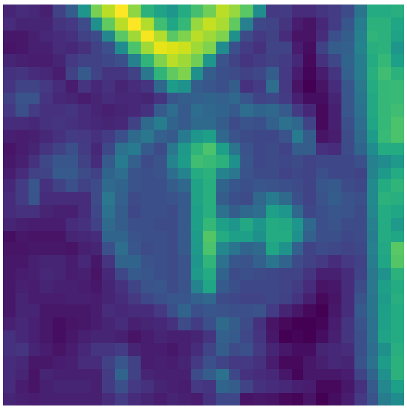
```
print(X_train[1000].shape)
print(y_train[1000])
def grayscale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img
```

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

In [ ]:

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

```
(32, 32)
```



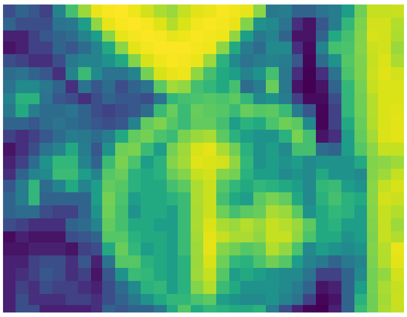
In [ ]:

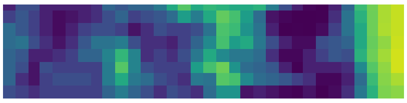
```
def equalize(img):
    img = cv2.equalizeHist(img)
    return img
```

In [ ]:

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

```
(32, 32)
```





In [ ]:

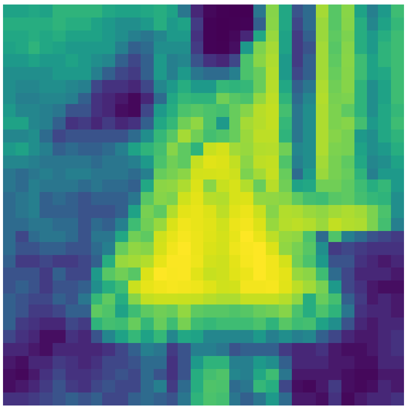
```
def preprocess(img):
    img = grayscale(img)
    img = equalize(img)
    img = img/255
    return img
```

In [ ]:

```
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)
```

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



In [ ]:

```
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)
```

In [ ]:

```
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)
```

In [ ]:

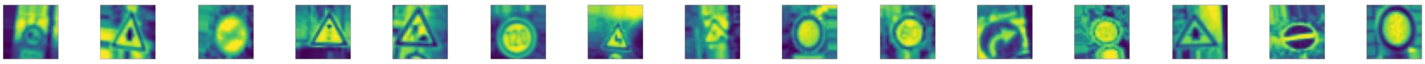
```
# for X_batch, y_batch in
batches = datagen.flow(X_train, y_train, batch_size = 20)
X_batch, y_batch = next(batches)

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")

print(X_batch.shape)
```

```
(20, 32, 32, 1)
```



In [ ]:

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

In [ ]:

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

In [ ]:

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

 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
```

In [ ]:

```
history = model.fit(datagen.flow(X_train, y_train, batch_size=15),
                              steps_per_epoch = 2000,
                              epochs=25,
                              validation_data=(X_val, y_val), shuffle = 1)
```

```
Epoch 1/25
2000/2000 [==============================] - 45s 16ms/step - loss: 1.6555 - accuracy: 0.5
275 - val_loss: 0.2404 - val_accuracy: 0.9329
Epoch 2/25
2000/2000 [==============================] - 31s 16ms/step - loss: 0.4900 - accuracy: 0.8
473 - val_loss: 0.1311 - val_accuracy: 0.9558
Epoch 3/25
2000/2000 [==============================] - 31s 15ms/step - loss: 0.3243 - accuracy: 0.8
985 - val_loss: 0.1496 - val_accuracy: 0.9506
Epoch 4/25
2000/2000 [==============================] - 31s 16ms/step - loss: 0.2616 - accuracy: 0.9
185 - val_loss: 0.0898 - val_accuracy: 0.9687
Epoch 5/25
2000/2000 [==============================] - 32s 16ms/step - loss: 0.2293 - accuracy: 0.9
309 - val_loss: 0.0618 - val_accuracy: 0.9816
Epoch 6/25
2000/2000 [==============================] - 31s 16ms/step - loss: 0.2065 - accuracy: 0.9
343 - val_loss: 0.0679 - val_accuracy: 0.9825
Epoch 7/25
2000/2000 [==============================] - 32s 16ms/step - loss: 0.1807 - accuracy: 0.9
442 - val_loss: 0.0635 - val_accuracy: 0.9800
Epoch 8/25
2000/2000 [==============================] - 32s 16ms/step - loss: 0.1637 - accuracy: 0.9
492 - val_loss: 0.0433 - val_accuracy: 0.9875
Epoch 9/25
2000/2000 [==============================] - 32s 16ms/step - loss: 0.1553 - accuracy: 0.9
532 - val_loss: 0.0620 - val_accuracy: 0.9841
Epoch 10/25
2000/2000 [==============================] - 32s 16ms/step - loss: 0.1479 - accuracy: 0.9
561 - val_loss: 0.0437 - val_accuracy: 0.9884
Epoch 11/25
2000/2000 [==============================] - 31s 16ms/step - loss: 0.1445 - accuracy: 0.9
564 - val_loss: 0.0445 - val_accuracy: 0.9878
Epoch 12/25
2000/2000 [==============================] - 31s 16ms/step - loss: 0.1353 - accuracy: 0.9
602 - val_loss: 0.0443 - val_accuracy: 0.9891
Epoch 13/25
2000/2000 [==============================] - 32s 16ms/step - loss: 0.1326 - accuracy: 0.9
617 - val_loss: 0.0682 - val_accuracy: 0.9823
Epoch 14/25
2000/2000 [==============================] - 32s 16ms/step - loss: 0.1197 - accuracy: 0.9
645 - val_loss: 0.0554 - val_accuracy: 0.9857
Epoch 15/25
2000/2000 [==============================] - 37s 19ms/step - loss: 0.1240 - accuracy: 0.9
632 - val_loss: 0.0386 - val_accuracy: 0.9887
Epoch 16/25
2000/2000 [==============================] - 31s 16ms/step - loss: 0.1201 - accuracy: 0.9
645 - val_loss: 0.0320 - val_accuracy: 0.9902
Epoch 17/25
2000/2000 [==============================] - 30s 15ms/step - loss: 0.1235 - accuracy: 0.9
637 - val_loss: 0.0486 - val_accuracy: 0.9857
Epoch 18/25
2000/2000 [==============================] - 32s 16ms/step - loss: 0.1133 - accuracy: 0.9
674 - val_loss: 0.0424 - val_accuracy: 0.9887
Epoch 19/25
2000/2000 [==============================] - 31s 16ms/step - loss: 0.1156 - accuracy: 0.9
671 - val_loss: 0.0265 - val_accuracy: 0.9934
Epoch 20/25
2000/2000 [==============================] - 31s 15ms/step - loss: 0.1135 - accuracy: 0.9
669 - val_loss: 0.0258 - val_accuracy: 0.9927
Epoch 21/25
2000/2000 [==============================] - 31s 15ms/step - loss: 0.1157 - accuracy: 0.9
659 - val_loss: 0.0557 - val_accuracy: 0.9871
Epoch 22/25
```

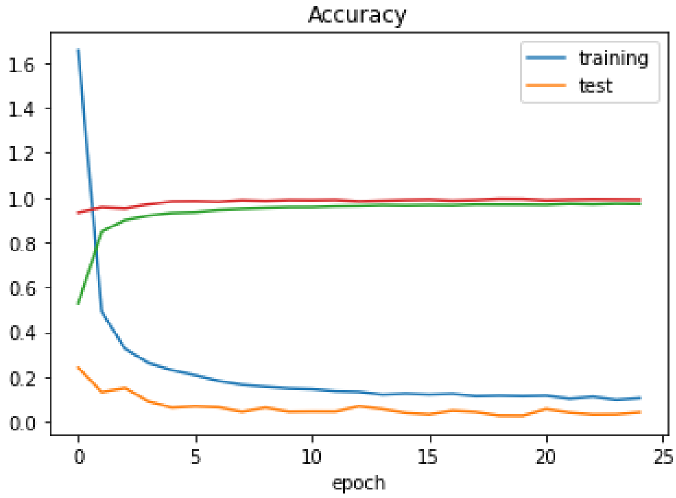
```
2000/2000 [==============================] - 30s 15ms/step - loss: 0.1015 - accuracy: 0.9
712 - val_loss: 0.0401 - val_accuracy: 0.9900
Epoch 23/25
2000/2000 [==============================] - 31s 15ms/step - loss: 0.1105 - accuracy: 0.9
686 - val_loss: 0.0314 - val_accuracy: 0.9914
Epoch 24/25
2000/2000 [==============================] - 30s 15ms/step - loss: 0.0966 - accuracy: 0.9
721 - val_loss: 0.0322 - val_accuracy: 0.9907
Epoch 25/25
2000/2000 [==============================] - 30s 15ms/step - loss: 0.1043 - accuracy: 0.9
706 - val_loss: 0.0415 - val_accuracy: 0.9900
```

In [ ]:

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

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

# TODO: Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=0)
```



In [ ]:

```
print('Test score:', score[0])
print('Test accuracy:', score[1])

#predict internet number
import requests
from PIL import Image
url = 'https://c8.alamy.com/comp/J2MRAJ/german-road-sign-bicycles-crossing-J2MRAJ.jpg'
r = requests.get(url, 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 = preprocess(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
img = img.reshape(1, 32, 32, 1)
img = model.predict(img)
img = np.argmax(img,axis=1)

print("predicted sign: "+ str(img))
```

```
Test score: 0.12952885031700134
Test accuracy: 0.9692794680595398
(32, 32)
```

```
predicted sign: [29]
```