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

```
Cloning into 'german-traffic-signs'...
Unpacking objects: 100% (6/6), 117.80 MiB | 8.64 MiB/s, done.
Updating files: 100% (4/4), done.
```

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

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

```
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers 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']

# 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)
```

```
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))
```

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)

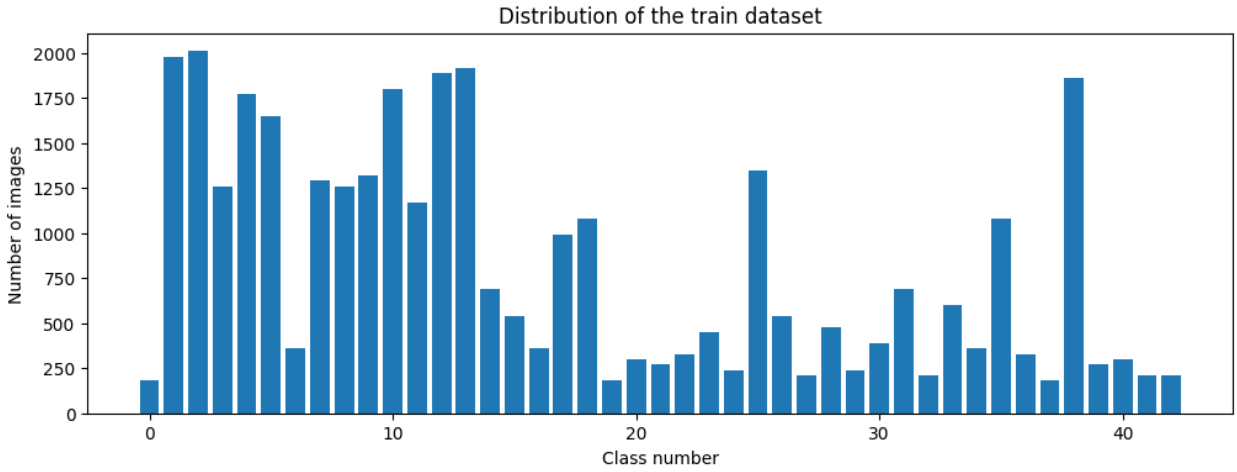
5 - Speed limit (80km/h)

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

7 - Speed limit (100km/h)

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

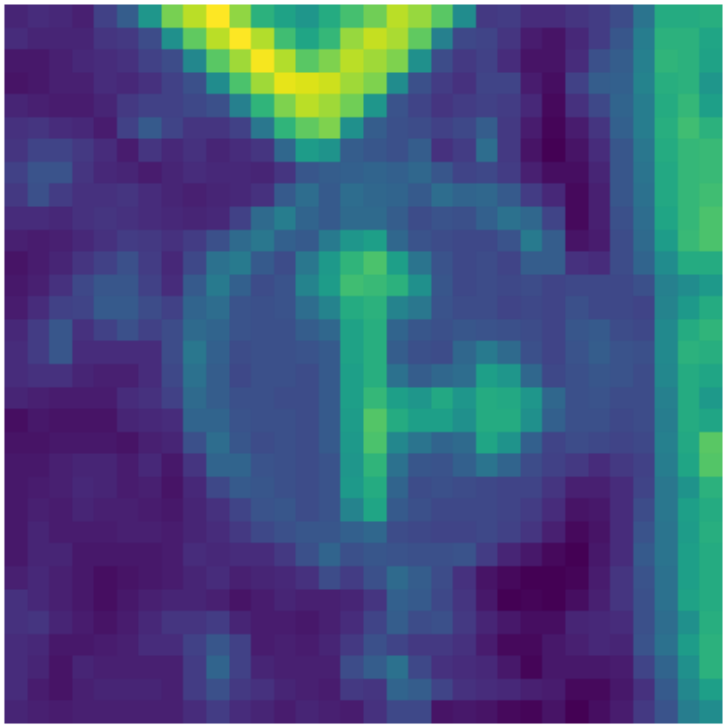



```
import cv2

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

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

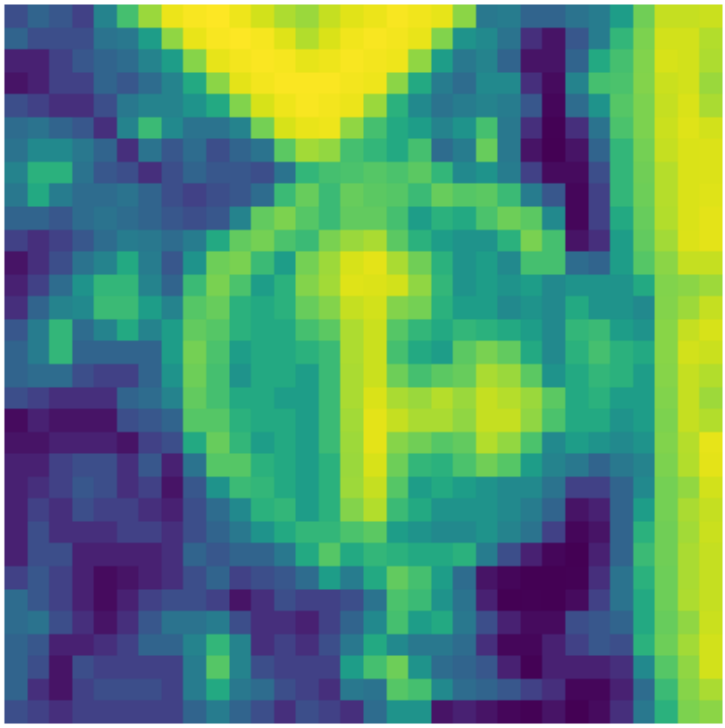
```
def grayscale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img
img = grayscale(X_train[1000])
plt.imshow(img)
plt.axis("off")
print(img.shape)
```

```
(32, 32)
```



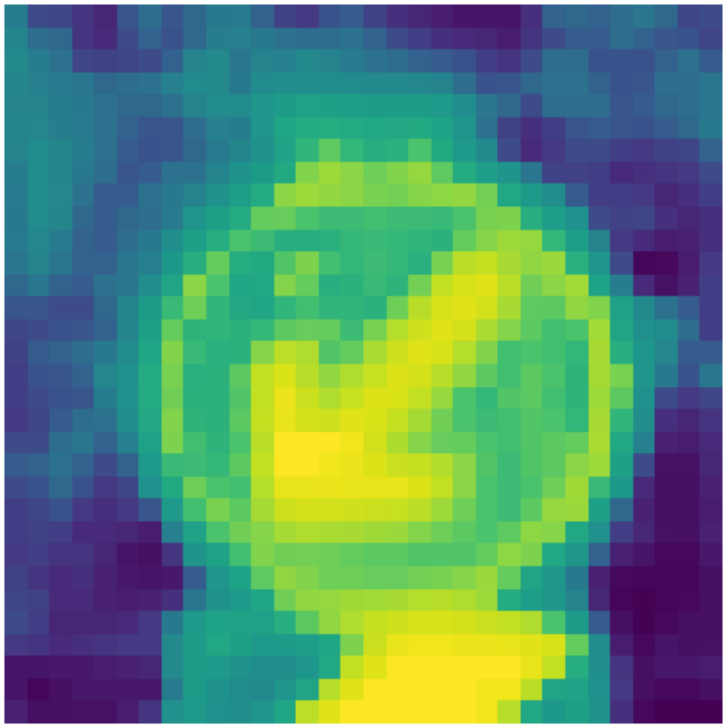
```
def equalize(img):
    img = cv2.equalizeHist(img)
    return img
img = equalize(img)
plt.imshow(img)
plt.axis("off")
print(img.shape)
```

```
(32, 32)
```



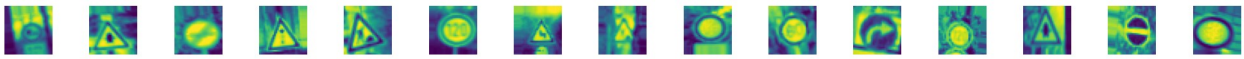
```
def preprocess(img):
    img = grayscale(img)
    img = equalize(img)
    img = img/255
    return img

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)
```

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



```
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")

print(X_batch.shape)
```

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



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

# 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())

history = model.fit_generator(datagen.flow(X_train, y_train,
batch_size=50),
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=(X_val, y_val), shuffle =
1)
```

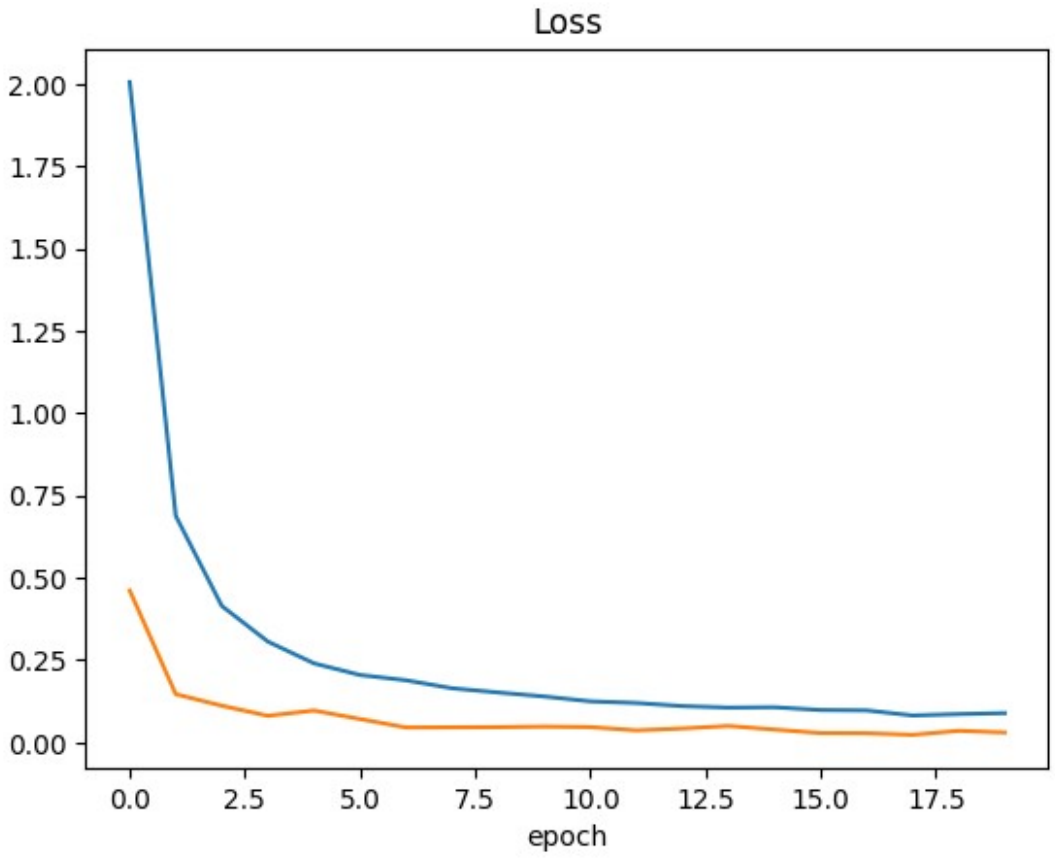
```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.Adam.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 60)        1560
```

```
 conv2d_1 (Conv2D)           (None, 24, 24, 60)        90060

 max_pooling2d (MaxPooling2  (None, 12, 12, 60)        0
 D)

 conv2d_2 (Conv2D)           (None, 10, 10, 30)        16230

 conv2d_3 (Conv2D)           (None, 8, 8, 30)          8130

 max_pooling2d_1 (MaxPoolin  (None, 4, 4, 30)          0
 g2D)

 flatten (Flatten)           (None, 480)               0

 dense (Dense)               (None, 500)               240500

 dropout (Dropout)           (None, 500)               0

 dense_1 (Dense)             (None, 43)                21543

=================================================================
Total params: 378023 (1.44 MB)
Trainable params: 378023 (1.44 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

<ipython-input-23-f90eee302bcd>:4: UserWarning: `Model.fit_generator`
is deprecated and will be removed in a future version. Please use
`Model.fit`, which supports generators.
  history = model.fit_generator(datagen.flow(X_train, y_train,
batch_size=50),

None
Epoch 1/20
500/500 [==============================] - 24s 25ms/step - loss:
2.0057 - accuracy: 0.4365 - val_loss: 0.4613 - val_accuracy: 0.8533
Epoch 2/20
500/500 [==============================] - 12s 24ms/step - loss:
0.6879 - accuracy: 0.7870 - val_loss: 0.1464 - val_accuracy: 0.9587
Epoch 3/20
500/500 [==============================] - 12s 25ms/step - loss:
0.4144 - accuracy: 0.8722 - val_loss: 0.1113 - val_accuracy: 0.9694
Epoch 4/20
500/500 [==============================] - 11s 23ms/step - loss:
0.3062 - accuracy: 0.9069 - val_loss: 0.0807 - val_accuracy: 0.9780
Epoch 5/20
500/500 [==============================] - 11s 22ms/step - loss:
0.2400 - accuracy: 0.9270 - val_loss: 0.0967 - val_accuracy: 0.9721
Epoch 6/20
500/500 [==============================] - 12s 24ms/step - loss:
```

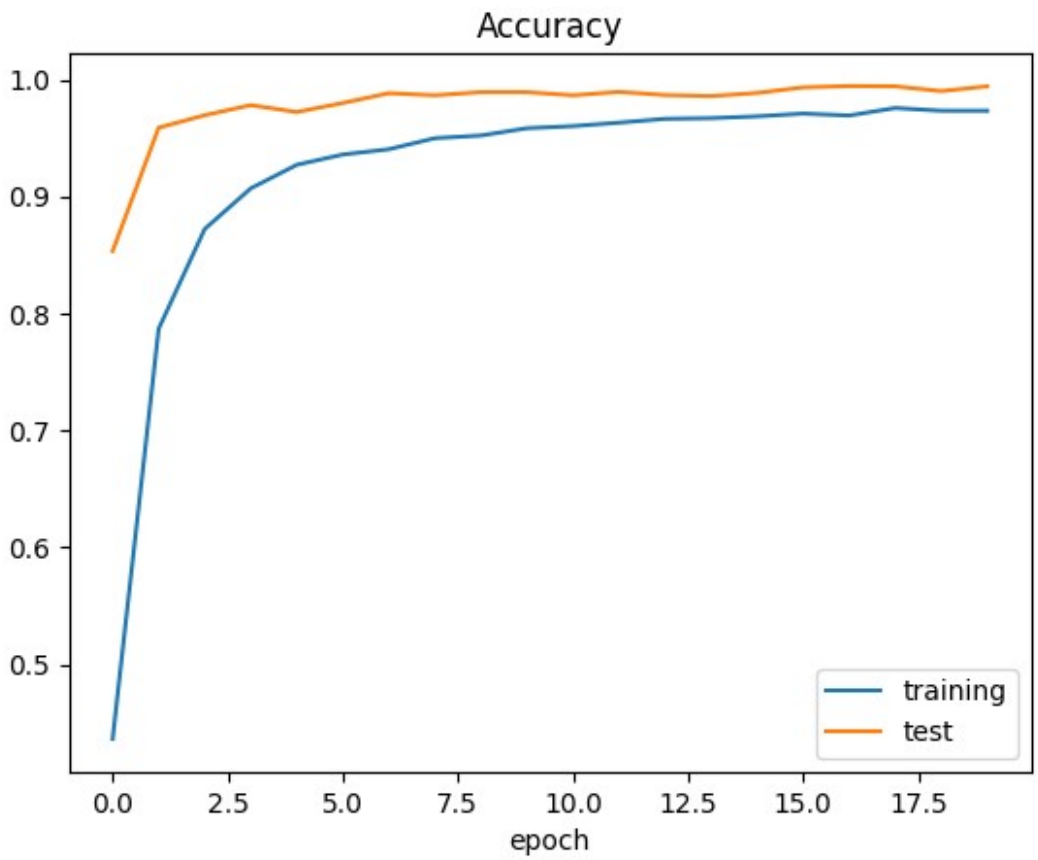
```
0.2047 - accuracy: 0.9358 - val_loss: 0.0709 - val_accuracy: 0.9798
Epoch 7/20
500/500 [==============================] - 12s 24ms/step - loss:
0.1886 - accuracy: 0.9403 - val_loss: 0.0453 - val_accuracy: 0.9882
Epoch 8/20
500/500 [==============================] - 12s 24ms/step - loss:
0.1643 - accuracy: 0.9498 - val_loss: 0.0457 - val_accuracy: 0.9864
Epoch 9/20
500/500 [==============================] - 11s 23ms/step - loss:
0.1516 - accuracy: 0.9520 - val_loss: 0.0462 - val_accuracy: 0.9891
Epoch 10/20
500/500 [==============================] - 12s 23ms/step - loss:
0.1394 - accuracy: 0.9583 - val_loss: 0.0480 - val_accuracy: 0.9891
Epoch 11/20
500/500 [==============================] - 12s 23ms/step - loss:
0.1247 - accuracy: 0.9601 - val_loss: 0.0464 - val_accuracy: 0.9864
Epoch 12/20
500/500 [==============================] - 12s 23ms/step - loss:
0.1197 - accuracy: 0.9630 - val_loss: 0.0365 - val_accuracy: 0.9893
Epoch 13/20
500/500 [==============================] - 12s 23ms/step - loss:
0.1102 - accuracy: 0.9663 - val_loss: 0.0423 - val_accuracy: 0.9866
Epoch 14/20
500/500 [==============================] - 11s 23ms/step - loss:
0.1056 - accuracy: 0.9669 - val_loss: 0.0499 - val_accuracy: 0.9857
Epoch 15/20
500/500 [==============================] - 11s 23ms/step - loss:
0.1066 - accuracy: 0.9684 - val_loss: 0.0389 - val_accuracy: 0.9884
Epoch 16/20
500/500 [==============================] - 11s 22ms/step - loss:
0.0983 - accuracy: 0.9708 - val_loss: 0.0283 - val_accuracy: 0.9932
Epoch 17/20
500/500 [==============================] - 11s 22ms/step - loss:
0.0971 - accuracy: 0.9692 - val_loss: 0.0278 - val_accuracy: 0.9943
Epoch 18/20
500/500 [==============================] - 11s 23ms/step - loss:
0.0817 - accuracy: 0.9756 - val_loss: 0.0229 - val_accuracy: 0.9941
Epoch 19/20
500/500 [==============================] - 12s 23ms/step - loss:
0.0859 - accuracy: 0.9732 - val_loss: 0.0352 - val_accuracy: 0.9900
Epoch 20/20
500/500 [==============================] - 12s 24ms/step - loss:
0.0889 - accuracy: 0.9732 - val_loss: 0.0300 - val_accuracy: 0.9941
```

```
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')

# 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: 0.13481435179710388
Test accuracy: 0.9684877395629883

#fetch image

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'))


#Preprocess image

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

```
#Reshape reshape

img = img.reshape(1, 32, 32, 1)

#Test image
print("predicted sign: "+ str(np.argmax(model.predict(img), axis=-1)))
```

```
(32, 32)
1/1 [==============================] - 0s 18ms/step
predicted sign: [29]
```