Self-Driving Car Engineer Nanodegree

Deep Learning

Project: Build a Traffic Sign Recognition Classifier

In this notebook, a template is provided for you to implement your functionality in stages, which is required to successfully complete this project. If additional code is required that cannot be included in the notebook, be sure that the Python code is successfully imported and included in your submission if necessary.

> **Note**: Once you have completed all of the code implementations, you need to finalize your work by exporting the iPython Notebook as an HTML document. Before exporting the notebook to html, all of the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In addition to implementing code, there is a writeup to complete. The writeup should be completed in a separate file, which can be either a markdown file or a pdf document. There is a write up template (https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project/blob/master/writeup_template.md) that can be used to guide the writing process. Completing the code template and writeup template will cover all of the rubric points (https://review.udacity.com/#!/rubrics/481/view) for this project.

The rubric (https://review.udacity.com/#!/rubrics/481/view) contains "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. The stand out suggestions are optional. If you decide to pursue the "stand out suggestions", you can include the code in this Ipython notebook and also discuss the results in the writeup file.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

Step -1: Preparations

Launching on AWS EC2 and other preparations

- go to console (https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2)
- Click on "Launch instance"
- Select community and select "udacity-carnd"
- Filter gy "GPU graphics"
- Edit security group. Add rule to allow connections to port 8888. Type: "Custom TCP Rule", Protocol "TCP", Port Range "8888", Source "Anywhere" (leave the "0.0.0.0/0")
- Edit security group to enable SSH (add SSH rule)
- Click launch instance, select "Proceed without a key pair" and acknowledge that you know the password (carnd) and user (carnd)

- Wait for instance to pass. Copy IP-Adress
- ssh instance
- `source activate carnd-term1`
- Clone LeNetLab `git clone https://github.com/BernhardSchlegel/CarND-Traffic-Sign-Classifier-Project`
- cd into folder, `mkdir traffic-signs-data` and `cd traffic-signs-data`
- Download and unzip traffic sign dataset `wget https://d17h27t6h515a5.cloudfront.net/topher/2017/February/5898cd6f_traffic-signs-data/traffic-signs-data.zip`, followed by `unzip traffic-signs-data.zip` or click here (https://d17h27t6h515a5.cloudfront.net/topher/2017/February/5898cd6f_traffic-signs-data/traffic-signs-data.zip).
- Move folder up `cd ..` and start jupyter notebook `jupyter notebook`

If you're on a GPU system

- Install GPU support as described here (https://www.tensorflow.org/install/#optional_install_cuda_gpus_on_linux)
- Execute `pip install tensorflow-gpu`

Installing OpenCV

easy `conda install --channel https://conda.anaconda.org/menpo opencv3`

hard:

```
sudo apt-get -y update
sudo apt-get -y upgrade
sudo apt-get -y install build-essential cmake git pkg-config
sudo apt-get -y install libjpeg8-dev libtiff5-dev libjasper-dev libpng12-de
v libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get -y install libgtk2.0-dev
sudo apt-get -y install libatlas-base-dev gfortran
cd ~
git clone https://github.com/Itseez/opencv.git
cd opencv
git checkout 3.2.0
cd ~
git clone https://github.com/Itseez/opencv_contrib.git
cd opencv_contrib
git checkout 3.2.0
cd ~/opencv
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D INSTALL_C_EXAMPLES=ON \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
    -D BUILD_EXAMPLES=ON ..
make -j8
sudo make install
sudo ldconfig
cd ~/.virtualenvs/cv/lib/python3.5/site-packages/
ln -s /usr/local/lib/python3.5/site-packages/cv2.cpython-34m.so cv2.so
```

In [1]:

```
import time
# define Log function
def log(text):
    print(time.strftime('%Y.%m.%d, %H:%M:%S') + ': ' + text)

def ping():
    return datetime.datetime.now()

def pong(dt):
    now = datetime.datetime.now()
    diff = now - dt
    ms = round(diff.total_seconds()*1000)
    return ms
```

Step 0: Load The Data

In [2]:

```
import os

# to save time after notebook reset, save and load data
def save_data():

    directory = "processed"

    if not os.path.exists(directory):
        os.makedirs(directory)

    np.save("processed/X_train", X_train)
    np.save("processed/y_train", y_train)
    np.save("processed/X_valid", X_valid)
    np.save("processed/y_valid", y_valid)
    np.save("processed/X_test_prep", X_test_prep)
    np.save("processed/y_test", y_test)

def load_data():
    global X_train
    global y_train
    global X_valid
    global y_valid
    global X_test_prep
    global y_test

    directory = "processed"

    if os.path.exists(directory):
        X_train = np.load("processed/X_train.npy")
        y_train = np.load("processed/y_train.npy")
        X_valid = np.load("processed/X_valid.npy")
        y_valid = np.load("processed/y_valid.npy")
        X_test_prep = np.load("processed/X_test_prep.npy")
        y_test = np.load("processed/y_test.npy")
```

In [3]:

```
# Load pickled data
import pickle

training_file =  'traffic-signs-data/train.p'
validation_file= 'traffic-signs-data/valid.p'
testing_file =   'traffic-signs-data/test.p'

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

Step 1: Dataset Summary & Exploration

The pickled data is a dictionary with 4 key/value pairs:

- `'features'` is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
- `'labels'` is a 1D array containing the label/class id of the traffic sign. The file `signnames.csv` contains id -> name mappings for each id.
- `'sizes'` is a list containing tuples, (width, height) representing the original width and height the image.
- `'coords'` is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image. **THESE COORDINATES ASSUME THE ORIGINAL IMAGE. THE PICKLED DATA CONTAINS RESIZED VERSIONS (32 by 32) OF THESE IMAGES**

Complete the basic data summary below. Use python, numpy and/or pandas methods to calculate the data summary rather than hard coding the results. For example, the pandas shape method (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.shape.html) might be useful for calculating some of the summary results.

Provide a Basic Summary of the Data Set Using Python, Numpy and/or Pandas

In [4]:

```
import numpy as np
### Replace each question mark with the appropriate value.
### Use python, pandas or numpy methods rather than hard coding the results

# TODO: Number of training examples
n_train = len(X_train)

# TODO: Number of validation examples
n_validation = len(X_valid)

# TODO: Number of testing examples.
n_test = len(X_test)

# TODO: What's the shape of an traffic sign image?
image_shape = X_train[0].shape

# TODO: How many unique classes/labels there are in the dataset.
classes, class_indices, class_counts = np.unique(y_train, return_index = True, retu
n_classes = len(classes)
log("Number of training examples ={}".format(n_train))
log("Number of testing examples ={}".format(n_test))
log("Number of validation examples ={}".format(n_validation))
log("Image data shape ={}".format(image_shape))
log("Number of classes ={}".format(n_classes))

assert(len(X_train) == len(y_train))
assert(len(X_valid) == len(y_valid))
assert(len(X_test) == len(y_test))
```

```
2017.09.01, 17:54:44: Number of training examples =34799
2017.09.01, 17:54:44: Number of testing examples =12630
2017.09.01, 17:54:44: Number of validation examples =4410
2017.09.01, 17:54:44: Image data shape =(32, 32, 3)
2017.09.01, 17:54:44: Number of classes =43
```

Include an exploratory visualization of the dataset

Visualize the German Traffic Signs Dataset using the pickled file(s). This is open ended, suggestions include: plotting traffic sign images, plotting the count of each sign, etc.

The Matplotlib (http://matplotlib.org/) examples (http://matplotlib.org/examples/index.html) and gallery (http://matplotlib.org/gallery.html) pages are a great resource for doing visualizations in Python.

NOTE: It's recommended you start with something simple first. If you wish to do more, come back to it after you've completed the rest of the sections. It can be interesting to look at the distribution of classes in the training, validation and test set. Is the distribution the same? Are there more examples of some classes than others?

In [5]:

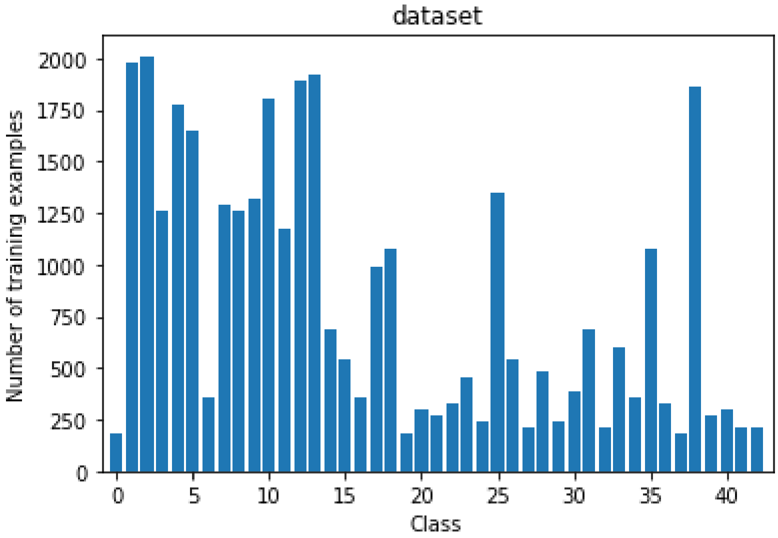
```
import pandas as pd
signnames = pd.DataFrame.from_csv('signnames.csv', index_col=None)
def get_sign_name(sign_number):
    return signnames[signnames['ClassId'] == sign_number].iloc[0,1]
```

In [6]:

```
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
# Visualizations will be shown in the notebook.
%matplotlib inline
```

In [7]:

```
# plot class distributio
def plot_class_counts(class_counts, heading = "dataset"):
    plt.bar( np.arange( 43 ), class_counts, align='center' )
    plt.xlabel('Class')
    #plt.xlabel(np.array(signnames['SignName']), rotation = 90)
    plt.ylabel('Number of training examples')
    plt.xlim([-1, 43])
    plt.title(heading)
    plt.show()

plot_class_counts(class_counts, heading = "dataset")
```



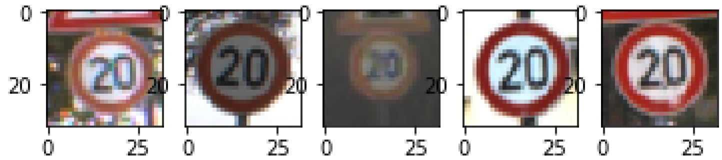
In [8]:

```
import random
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def plot_image(img):
    plt.figure(figsize=(1,1))
    plt.imshow(img.squeeze(), cmap="gray")

idx = None # ensure same images to show after each step
def plot_images_for_class(X_train, y_train, mode = 'random', max_classes = 3, n_per
    global idx
    global n_classes

    for i in classes:
        if i >= max_classes:
            break

        log('--------------------------------------------------------------')
        log('Examples for class {}, \"{}\" ({} samples)'.format(i, get_sign_name(i)

        # get 5 random indices
        X_train_class = X_train[y_train == i]

        if mode is 'random':
            if idx is None:
                idx = np.random.randint(X_train_class.shape[0], size=n_per_class)
            X_sub = X_train_class[idx]
        elif mode is 'deterministic':
            if X_train_class.shape[0] < n_per_class:
                n_classes = X_train_class.shape[0]
            X_sub = X_train_class[0:n_per_class]

        fig, axs = plt.subplots(nrows=1, ncols=n_per_class)

        for j in range(0,n_per_class):
            image = X_sub[j].squeeze()

            axs[j].imshow(image, cmap="gray")

        plt.show()
```

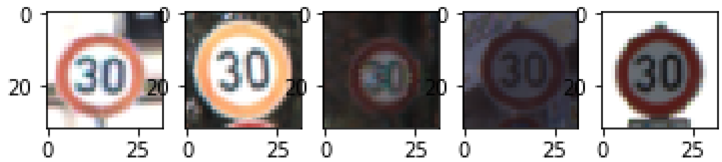
In [9]:

```
plot_images_for_class(X_train, y_train, max_classes = 43)
```

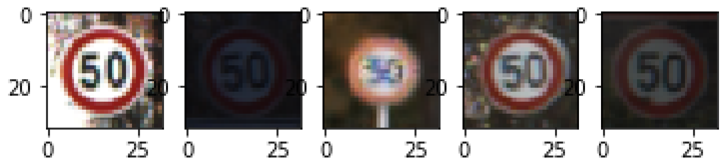
```
2017.09.01, 17:54:50: --------------------------------------------------
--------------
2017.09.01, 17:54:50: Examples for class 0, "Speed limit (20km/h)" (18
0 samples)
```



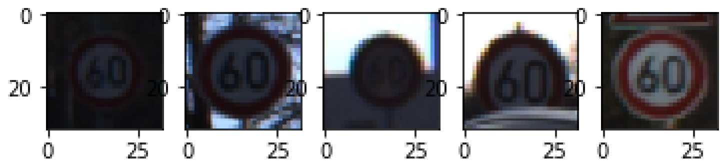
```
2017.09.01, 17:54:51: --------------------------------------------------
--------------
2017.09.01, 17:54:51: Examples for class 1, "Speed limit (30km/h)" (19
80 samples)
```



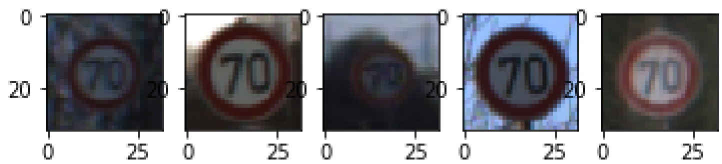
```
2017.09.01, 17:54:52: --------------------------------------------------
--------------
2017.09.01, 17:54:52: Examples for class 2, "Speed limit (50km/h)" (20
10 samples)
```



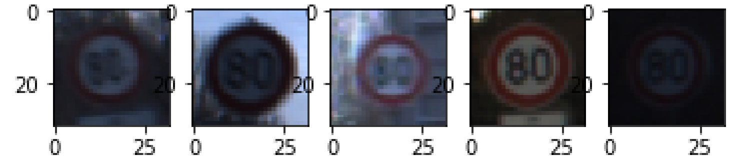
```
2017.09.01, 17:54:52: --------------------------------------------------
--------------
2017.09.01, 17:54:52: Examples for class 3, "Speed limit (60km/h)" (12
60 samples)
```



```
2017.09.01, 17:54:53: --------------------------------------------------
--------------
2017.09.01, 17:54:53: Examples for class 4, "Speed limit (70km/h)" (17
70 samples)
```

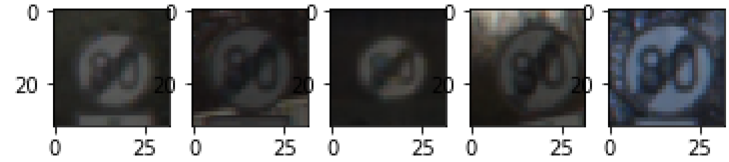


2017.09.01, 17:54:53: --



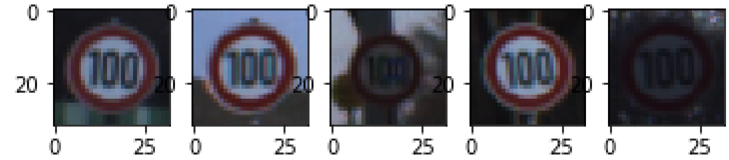
2017.09.01, 17:54:53: --

2017.09.01, 17:54:53: Examples for class 6, "End of speed limit (80km/h)" (360 samples)



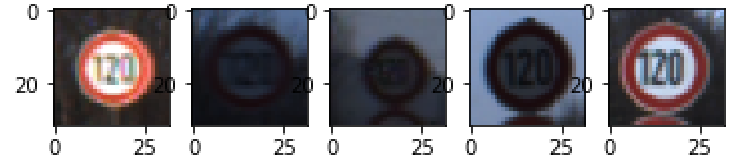
2017.09.01, 17:54:54: --

2017.09.01, 17:54:54: Examples for class 7, "Speed limit (100km/h)" (1290 samples)



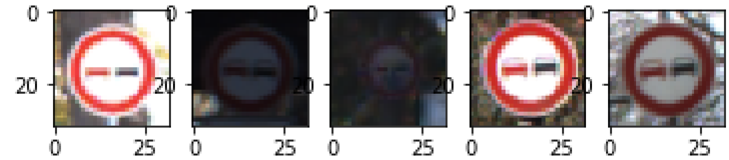
2017.09.01, 17:54:54: --

2017.09.01, 17:54:54: Examples for class 8, "Speed limit (120km/h)" (1260 samples)



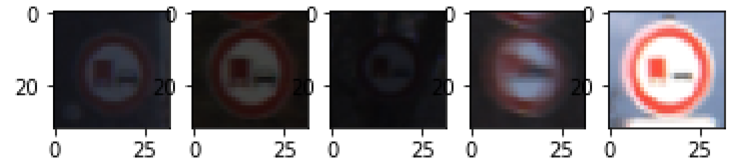
2017.09.01, 17:54:55: --

2017.09.01, 17:54:55: Examples for class 9, "No passing" (1320 samples)

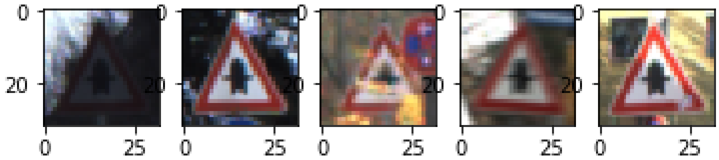


2017.09.01, 17:54:55: --

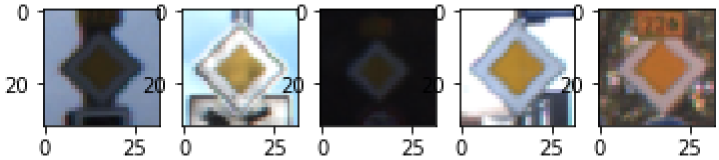
2017.09.01, 17:54:55: Examples for class 10, "No passing for vehicles over 3.5 metric tons" (1800 samples)



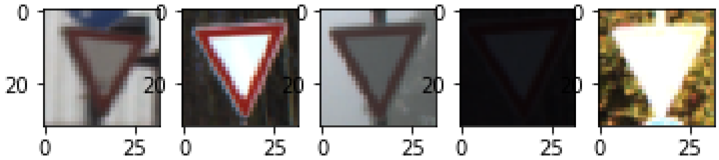
```
2017.09.01, 17:54:56: ----------------------------------------------------------------
2017.09.01, 17:54:56: Examples for class 11, "Right-of-way at the next intersection" (1170 samples)
```



```
2017.09.01, 17:54:56: ----------------------------------------------------------------
2017.09.01, 17:54:56: Examples for class 12, "Priority road" (1890 samples)
```

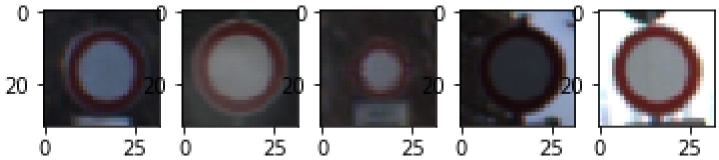


```
2017.09.01, 17:54:57: ----------------------------------------------------------------
2017.09.01, 17:54:57: Examples for class 13, "Yield" (1920 samples)
```

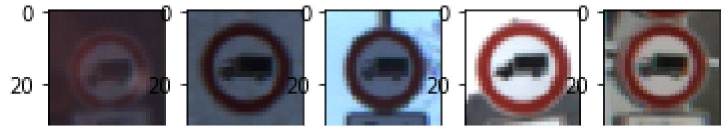


```
2017.09.01, 17:54:57: ----------------------------------------------------------------
2017.09.01, 17:54:57: Examples for class 14, "Stop" (690 samples)
```

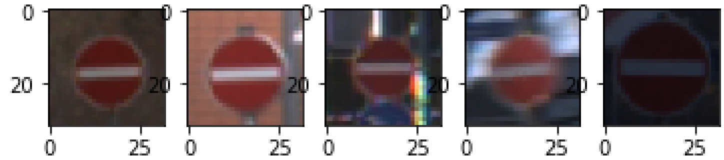
```
2017.09.01, 17:54:57: ----------------------------------------------------------------
2017.09.01, 17:54:57: Examples for class 15, "No vehicles" (540 samples)
```



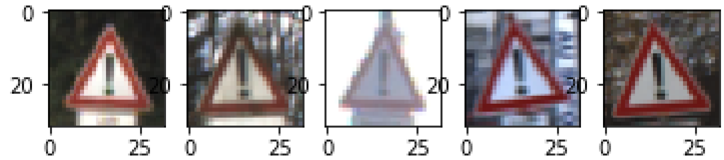
```
2017.09.01, 17:54:58: ----------------------------------------------------------------
2017.09.01, 17:54:58: Examples for class 16, "Vehicles over 3.5 metric tons prohibited" (360 samples)
```



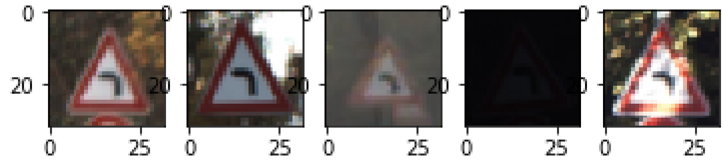
```
2017.09.01, 17:54:58: --------------------------------------------------
--------------
2017.09.01, 17:54:58: Examples for class 17, "No entry" (990 samples)
```



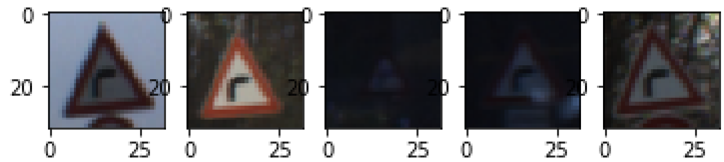
```
2017.09.01, 17:54:59: --------------------------------------------------
--------------
2017.09.01, 17:54:59: Examples for class 18, "General caution" (1080 s
amples)
```



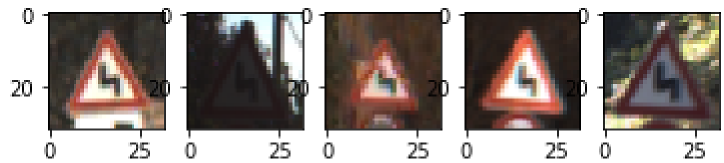
```
2017.09.01, 17:54:59: --------------------------------------------------
--------------
2017.09.01, 17:54:59: Examples for class 19, "Dangerous curve to the l
eft" (180 samples)
```



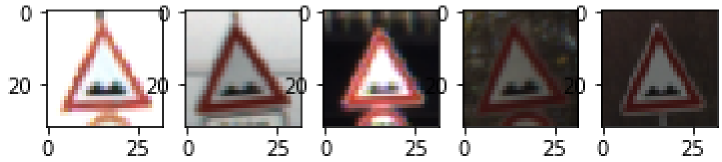
```
2017.09.01, 17:55:00: --------------------------------------------------
--------------
2017.09.01, 17:55:00: Examples for class 20, "Dangerous curve to the r
ight" (300 samples)
```



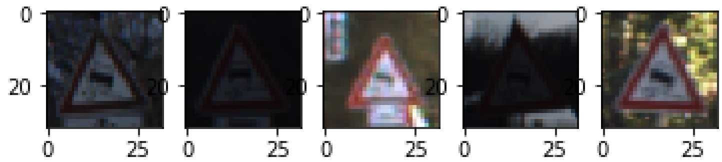
```
2017.09.01, 17:55:00: --------------------------------------------------
--------------
2017.09.01, 17:55:00: Examples for class 21, "Double curve" (270 sampl
es)
```



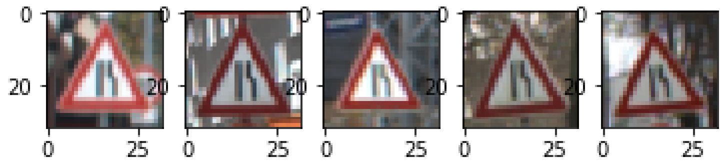
```
2017.09.01, 17:55:00: --------------------------------------------------
--------------
2017.09.01, 17:55:00: Examples for class 22, "Bumpy road" (330 sample
s)
```



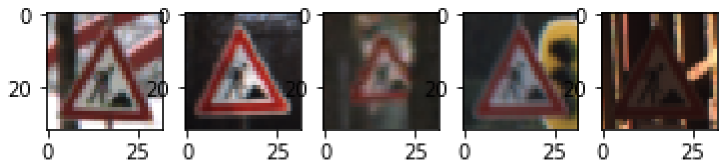
```
2017.09.01, 17:55:01: --------------------------------------------------------------
2017.09.01, 17:55:01: Examples for class 23, "Slippery road" (450 samples)
```



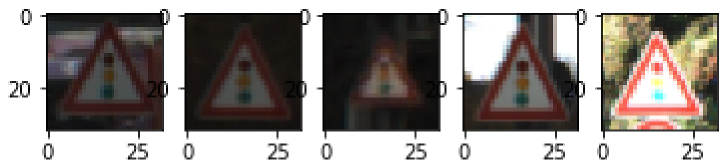
```
2017.09.01, 17:55:01: --------------------------------------------------------------
2017.09.01, 17:55:01: Examples for class 24, "Road narrows on the right" (240 samples)
```



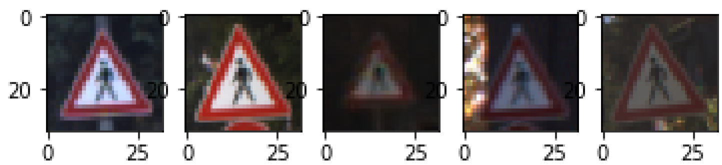
```
2017.09.01, 17:55:02: --------------------------------------------------------------
2017.09.01, 17:55:02: Examples for class 25, "Road work" (1350 samples)
```



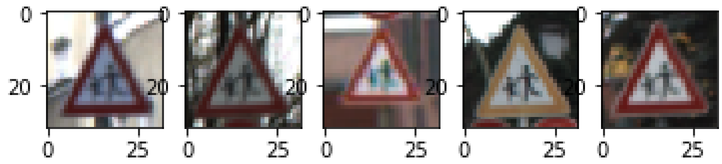
```
2017.09.01, 17:55:02: --------------------------------------------------------------
2017.09.01, 17:55:02: Examples for class 26, "Traffic signals" (540 samples)
```



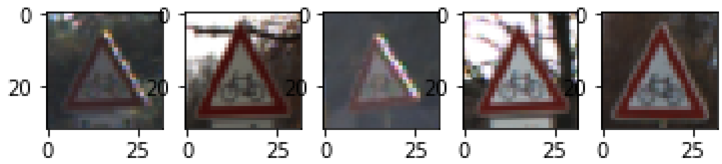
```
2017.09.01, 17:55:03: --------------------------------------------------------------
2017.09.01, 17:55:03: Examples for class 27, "Pedestrians" (210 samples)
```



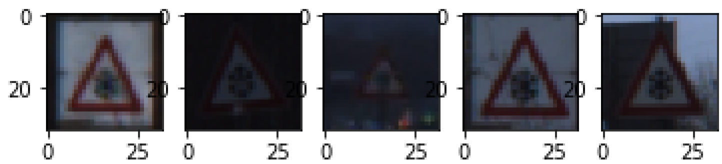
```
2017.09.01, 17:55:03: -------------------------------------------------
--------------
2017.09.01, 17:55:03: Examples for class 28, "Children crossing" (480
 samples)
```



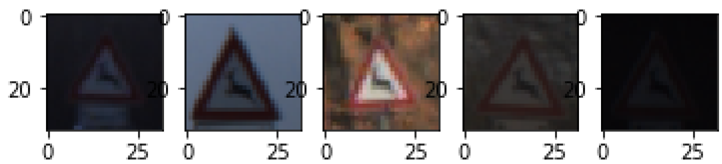
```
2017.09.01, 17:55:03: -------------------------------------------------
--------------
2017.09.01, 17:55:03: Examples for class 29, "Bicycles crossing" (240
 samples)
```



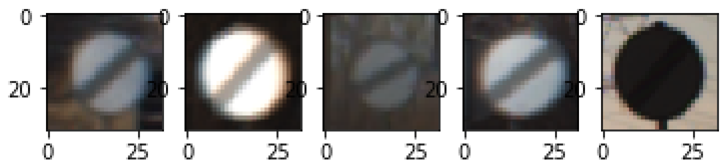
```
2017.09.01, 17:55:04: -------------------------------------------------
--------------
2017.09.01, 17:55:04: Examples for class 30, "Beware of ice/snow" (390
 samples)
```



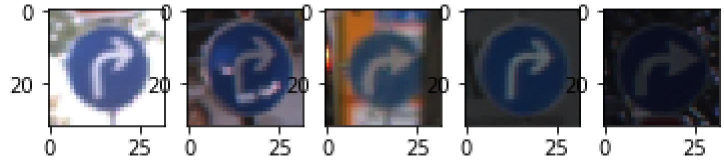
```
2017.09.01, 17:55:04: -------------------------------------------------
--------------
2017.09.01, 17:55:04: Examples for class 31, "Wild animals crossing"
 (690 samples)
```



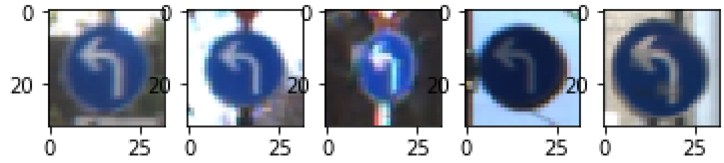
```
2017.09.01, 17:55:05: -------------------------------------------------
--------------
2017.09.01, 17:55:05: Examples for class 32, "End of all speed and pas
sing limits" (210 samples)
```



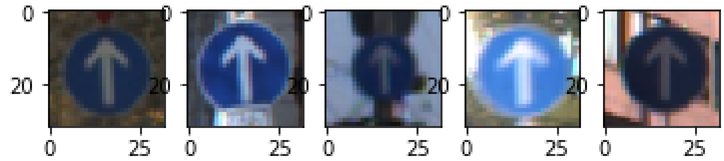
```
2017.09.01, 17:55:05: ----------------------------------------------------------------
2017.09.01, 17:55:05: Examples for class 33, "Turn right ahead" (599
```



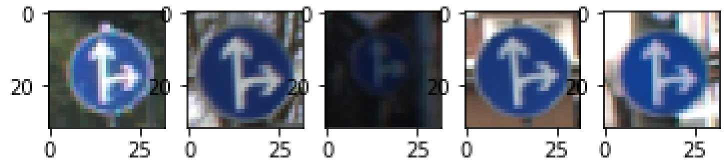
```
2017.09.01, 17:55:06: -----------------------------------------------------------------
2017.09.01, 17:55:06: Examples for class 34, "Turn left ahead" (360 sa
mples)
```



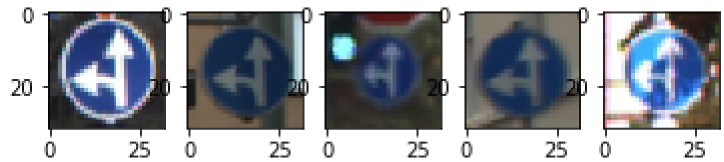
```
2017.09.01, 17:55:06: -----------------------------------------------------------------
2017.09.01, 17:55:06: Examples for class 35, "Ahead only" (1080 sample
s)
```



```
2017.09.01, 17:55:07: -----------------------------------------------------------------
2017.09.01, 17:55:07: Examples for class 36, "Go straight or right" (3
30 samples)
```



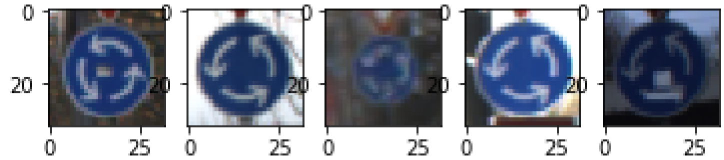
```
2017.09.01, 17:55:07: -----------------------------------------------------------------
2017.09.01, 17:55:07: Examples for class 37, "Go straight or left" (18
0 samples)
```



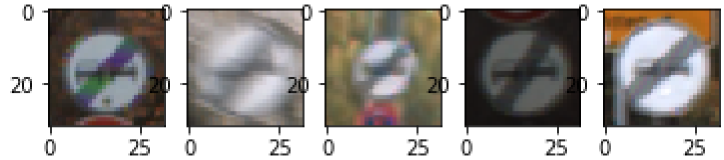
```
2017.09.01, 17:55:07: -----------------------------------------------------------------
2017.09.01, 17:55:07: Examples for class 38, "Keep right" (1860 sample
s)
```

```
2017.09.01, 17:55:08: -------------------------------------------------
--------------
2017.09.01, 17:55:08: Examples for class 39, "Keep left" (270 samples)
```

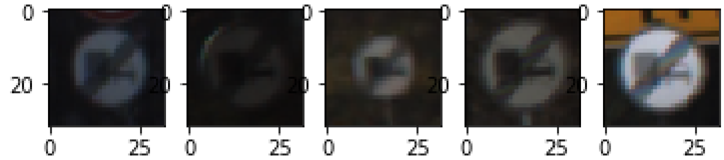
```
2017.09.01, 17:55:08: -------------------------------------------------
--------------
2017.09.01, 17:55:08: Examples for class 40, "Roundabout mandatory" (3
00 samples)
```



```
2017.09.01, 17:55:09: -------------------------------------------------
--------------
2017.09.01, 17:55:09: Examples for class 41, "End of no passing" (210
 samples)
```



```
2017.09.01, 17:55:09: -------------------------------------------------
--------------
2017.09.01, 17:55:09: Examples for class 42, "End of no passing by veh
icles over 3.5 metric tons" (210 samples)
```



In [10]:

```
# image_num, flip_x, flip_y, flip_xy, new class after transformation (-1 new = old)
augmentables = [
    [0, 0, 0, 0, -1],   # class Speed limit (20km/h)
    [1, 0, 0, 0, -1],   # class Speed limit (30km/h)
    [2, 0, 0, 0, -1],   # class Speed limit (50km/h)
    [3, 0, 0, 0, -1],   # class Speed limit (60km/h)
    [4, 0, 0, 0, -1],   # class Speed limit (70km/h)
    [5, 0, 0, 0, -1],   # class Speed limit (80km/h)
    [6, 0, 0, 0, -1],   # class End of speed limit (80km/h)
    [7, 0, 0, 0, -1],   # class Speed limit (100km/h)
    [8, 0, 0, 0, -1],   # class Speed limit (120km/h)
    [9, 1, 0, 0, -1],   # class No passing
    [10, 0, 0, 0, -1],  # class No passing for vehicles over 3.5 metric tons
    [11, 1, 0, 0, -1],  # class Right-of-way at the next intersection
    [12, 1, 1, 1, -1],  # class Priority road
    [13, 1, 0, 0, -1],  # class Yield
    [14, 0, 0, 0, -1],  # class Stop
    [15, 1, 1, 1, -1],  # class No vehicles
    [16, 0, 0, 0, -1],  # class Vehicles over 3.5 metric tons prohibited
    [17, 1, 1, 1, -1],  # class No entry
    [18, 1, 0, 0, -1],  # class General caution
    [19, 1, 0, 0, 20],  # class Dangerous curve to the left
    [20, 1, 0, 0, 19],  # class Dangerous curve to the right
    [21, 0, 0, 0, -1],  # class Double curve
    [22, 1, 0, 0, -1],  # class Bumpy road
    [23, 0, 0, 0, -1],  # class Slippery road
    [24, 0, 0, 0, -1],  # class Road narrows on the right
    [25, 0, 0, 0, -1],  # class Road work
    [26, 1, 0, 0, -1],  # class Traffic signals
    [27, 0, 0, 0, -1],  # class Pedestrians
    [28, 0, 0, 0, -1],  # class Children crossing
    [29, 0, 0, 0, -1],  # class Bicycles crossing
    [30, 1, 0, 0, -1],  # class Beware of ice/snow
    [31, 0, 0, 0, -1],  # class Wild animals crossing
    [32, 0, 0, 1, -1],  # class End of all speed and passing limits
    [33, 1, 0, 0, 34],  # class Turn right ahead
    [34, 1, 0, 0, 33],  # class Turn left ahead
    [35, 1, 0, 0, -1],  # class Ahead only
    [36, 1, 0, 0, 37],  # class Go straight or right
    [37, 1, 0, 0, 36],  # class Go straight or left
    [38, 1, 0, 0, 39],  # class Keep right
    [39, 1, 0, 0, 38],  # class Keep left
    [40, 0, 0, 0, -1],  # class Roundabout mandatory
    [41, 0, 0, 0, -1],  # class End of no passing
    [42, 0, 0, 0, -1]   # class End of no passing by vehicles over 3.5 metric tons
]
```

In [11]:

```
from skimage.transform import warp
from skimage.transform import rotate
from skimage.transform import rescale
from skimage.transform import ProjectiveTransform
import cv2

def flip_X(img):
    return img.copy()[:, ::-1]

def flip_Y(img):
    return img.copy()[::-1, :]

def augment_by_flipping(X, y):
    # augment as much as possible
    X_augmented = []
    y_augmented = []

    for image_num, flip_x, flip_y, flip_xy, new_class in augmentables:
        X_sub_image = X[y == image_num]
        n_images_of_class = X_sub_image.shape[0]
        if (flip_x + flip_y + flip_xy) > 0:
            log("found {} images of class {}. Applying: flip_x={}, flip_y={}, flip_
                n_images_of_class, image_num, flip_x, flip_y, flip_xy))

            counter = 0
            for image in X_sub_image:
                if new_class is -1:
                    new_class = image_num

                if flip_x > 0:
                    X_augmented.append(flip_X(image))
                    y_augmented.append(new_class)
                    counter += 1

                if flip_y > 0:
                    X_augmented.append(flip_Y(image))
                    y_augmented.append(new_class)
                    counter += 1

                if flip_xy > 0:
                    image = flip_Y(image)
                    X_augmented.append(flip_X(image))
                    y_augmented.append(new_class)
                    counter += 1

            log("created {} artificial samples.".format(counter))
        else:
            log("class {} not transformable".format(image_num))

    return np.array(X_augmented), y_augmented



def modify_warp(img, intensity = 0.25, depth_layer = 0):
    rows,cols,ch = img.shape

    assert len(img.shape) is 3, "this method only works for images with a depth cha
    assert ch is 1, "this method only works for images with one channel"
```

```
    img_shape_x, img_shape_y, depth = img.shape
    p1_left_x = int(img_shape_x * intensity)
    p1_right_x = int(img_shape_x * ( 1 - intensity))
    p1_top_y = int(img_shape_y * intensity)
    p1_bot_y = int(img_shape_y * ( 1 - intensity))
    p2_left_x = int(img_shape_x * intensity)
    p2_right_x = int(img_shape_x * ( 1 - intensity))
    p2_top_y = int(img_shape_y * intensity)
    p2_bot_y = int(img_shape_y * ( 1 - intensity))

    # points in array are numbered like follows
    #  1. 3.
    #  2. 4.
    pts1 = np.float32([[p1_left_x, p1_top_y], [p1_left_x, p1_bot_y], [p1_right_x, p
    pts2 = np.float32([[p1_left_x - random.uniform(0, p1_left_x) , p1_top_y - rando
                       [p1_left_x - random.uniform(0, p1_left_x) , p1_bot_y + rando
                       [p1_right_x + random.uniform(0, img_shape_x - p1_right_x) ,
                       [p1_right_x + random.uniform(0, img_shape_x - p1_right_x), p

    M = cv2.getPerspectiveTransform(pts1,pts2)
    dst = np.ones(shape = [32,32,1])
    dst[:,:,depth_layer] = np.array(cv2.warpPerspective(img,M,(img_shape_y,img_shap

    return dst


def augment_by_transforming(X, y, class_counts, intensity = 0.2, min_multiplier = 1
    """
    This function will modify each class count so that every class is present
    number of times of the most present class times the min_multiplier
    """

    X_augmented = []
    y_augmented = []

    n_idx_max = np.argmax(class_counts)
    n_max = class_counts[n_idx_max]
    target_samples = n_max * min_multiplier
    log("class {} ({}) is the maximally represented class (samples={}). Target samp
        n_idx_max, get_sign_name(n_idx_max), n_max, target_samples))

    for class_num in range(len(class_counts)):
        X_sub_image = X[y == class_num]
        n_class = class_counts[class_num]
        oversampling_ratio = target_samples / n_class
        samples_to_create = int(oversampling_ratio * n_class) - n_class
        log("class {0} is present {1} times. Oversampling ratio is {2:.3}. Samples
                class_num, n_class, oversampling_ratio, samples_to_create))

        n_generated = 0
        while n_generated < samples_to_create:
            # iterate through class instances
            image = X_sub_image[n_generated%class_counts[class_num]]

            # add to arrray (X and y)
            X_augmented.append(modify_warp(image, intensity = intensity))
            y_augmented.append(class_num)

            n_generated += 1
            if int(n_generated % (samples_to_create / 10)) is 0 and debug is True:
                log('generating {} / {} sample. '.format(n_generated, samples_to_cr
```

```
        #else:
          #  log('## {} / {}'.format(n_generated, samples_to_create))

    return np.array(X_augmented), y_augmented
```

In [12]:

```
def augment_dataset(X, y):
    # before
    classes, class_indices, class_counts = np.unique(y, return_index = True, return
    plot_class_counts(class_counts, "dataset original")

    # augment by flipping
    X_flipped, y_flipped = augment_by_flipping(X, y)
    log("created {} new samples by flipping.".format(X_flipped.shape[0]))
    X_train_augmented = np.append(X, X_flipped, axis = 0)
    y_train_augmented = np.append(y, y_flipped, axis = 0)
    classes, class_indices, class_counts = np.unique(y_train_augmented, return_inde
    plot_class_counts(class_counts)

    # augment by transforming
    X_transformed, y_transformed = augment_by_transforming(X_train_augmented, y_tra
    log("created {} new samples by transforming.".format(X_transformed.shape[0] - X
    X_train_augmented = np.append(X_train_augmented, X_transformed, axis = 0)
    y_train_augmented = np.append(y_train_augmented, y_transformed, axis = 0)
    classes, class_indices, class_counts = np.unique(y_train_augmented, return_inde
    plot_class_counts(class_counts, "dataset original")

    return X_train_augmented, y_train_augmented
```

Step 2: Design and Test a Model Architecture

Design and implement a deep learning model that learns to recognize traffic signs. Train and test your model on the German Traffic Sign Dataset (http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset).

The LeNet-5 implementation shown in the classroom (https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/601ae704-1035-4287-8b11-e2c2716217ad/concepts/d4aca031-508f-4e0b-b493-e7b706120f81) at the end of the CNN lesson is a solid starting point. You'll have to change the number of classes and possibly the preprocessing, but aside from that it's plug and play!

With the LeNet-5 solution from the lecture, you should expect a validation set accuracy of about 0.89. To meet specifications, the validation set accuracy will need to be at least 0.93. It is possible to get an even higher accuracy, but 0.93 is the minimum for a successful project submission.

There are various aspects to consider when thinking about this problem:

- Neural network architecture (is the network over or underfitting?)
- Play around preprocessing techniques (normalization, rgb to grayscale, etc)
- Number of examples per label (some have more than others).
- Generate fake data.

Here is an example of a published baseline model on this problem (http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf). It's not required to be familiar with the approach used in the paper but, it's good practice to try to read papers like these.

Pre-process the Data Set (normalization, grayscale, etc.)

Minimally, the image data should be normalized so that the data has mean zero and equal variance. For image data, `(pixel - 128)/ 128` is a quick way to approximately normalize the data and can be used in this project.

Other pre-processing steps are optional. You can try different techniques to see if it improves performance.

Use the code cell (or multiple code cells, if necessary) to implement the first step of your project.

In [13]:

```
from skimage import exposure
import sklearn.preprocessing as pp
from sklearn.utils import shuffle

def rgb2gray(rgb):
    new = np.empty([rgb.shape[0], rgb.shape[1], rgb.shape[2], 1])
    new[:,:,:,0] = np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
    return new
```

In [14]:

```
def preprocess_images(X, do_hist = True):
    # grayscale
    X = rgb2gray(X)

    # normalize
    X =  X = (X / 255.).astype(np.float32)

    # histogram localication
    if do_hist:
        for i in range(X.shape[0]):
            X[i,:,:,0] = exposure.equalize_adapthist(X[i,:,:,0])

    return X
```

In [ ]:

```
load_data()
```

In [15]:

```
log('processing training')
X_train = preprocess_images(X_train, do_hist=True)
```

```
2017.09.01, 17:55:11: processing training
```

```
/home/q372283/anaconda3/lib/python3.5/site-packages/skimage/util/dtyp
e.py:122: UserWarning: Possible precision loss when converting from fl
oat32 to uint16
  .format(dtypeobj_in, dtypeobj_out))
```

In [16]:

```
log('processing validation')
X_valid = preprocess_images(X_valid, do_hist=True)
```

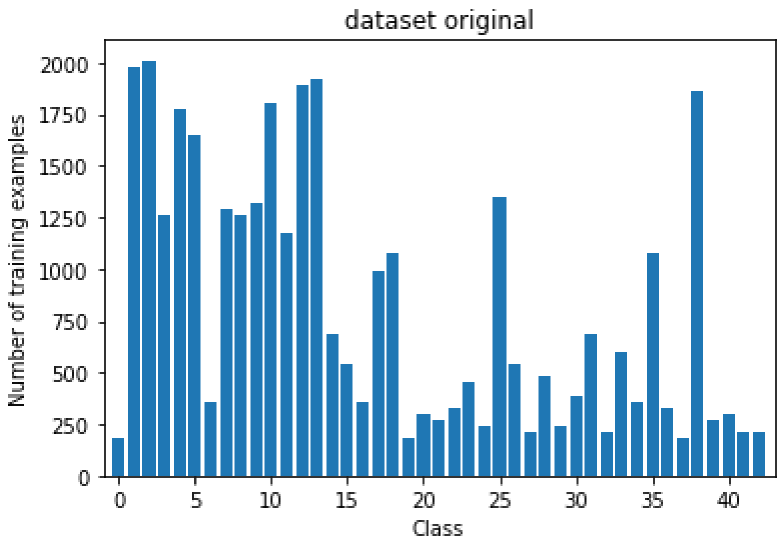
```
2017.09.01, 17:59:42: processing validation
```

```
/home/q372283/anaconda3/lib/python3.5/site-packages/skimage/util/dtyp
e.py:122: UserWarning: Possible precision loss when converting from fl
oat32 to uint16
  .format(dtypeobj_in, dtypeobj_out))
```

In [17]:

```
log('augmenting train')
X_train_aug, y_train_aug = augment_dataset(X_train, y_train)
```

```
2017.09.01, 18:00:15: augmenting train
```



In [18]:

```
X_test_prep = preprocess_images(X_test)
```

```
/home/q372283/anaconda3/lib/python3.5/site-packages/skimage/util/dtyp
e.py:122: UserWarning: Possible precision loss when converting from fl
oat32 to uint16
  .format(dtypeobj_in, dtypeobj_out))
```

In [19]:

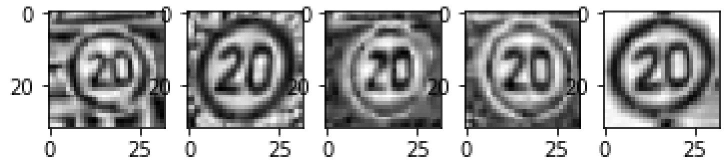
```
log("training with a total number of {} samples".format(y_train.shape[0]))
```

```
2017.09.01, 18:02:11: training with a total number of 34799 samples
```

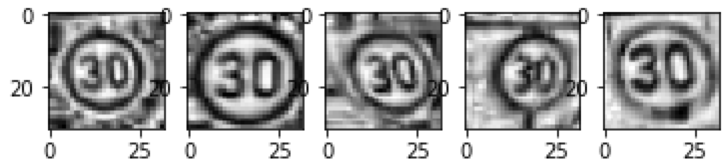
In [20]:

```
idx = None
plot_images_for_class(X_train_aug, y_train_aug, max_classes=3)
```

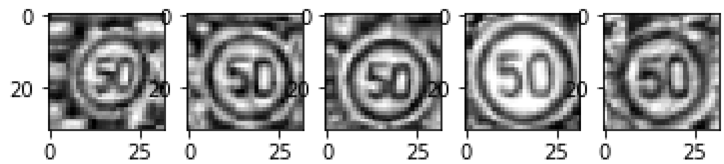
```
2017.09.01, 18:02:11: --------------------------------------------------
--------------
2017.09.01, 18:02:11: Examples for class 0, "Speed limit (20km/h)" (18
0 samples)
```



```
2017.09.01, 18:02:11: --------------------------------------------------
--------------
2017.09.01, 18:02:11: Examples for class 1, "Speed limit (30km/h)" (19
80 samples)
```



```
2017.09.01, 18:02:12: --------------------------------------------------
--------------
2017.09.01, 18:02:12: Examples for class 2, "Speed limit (50km/h)" (20
10 samples)
```

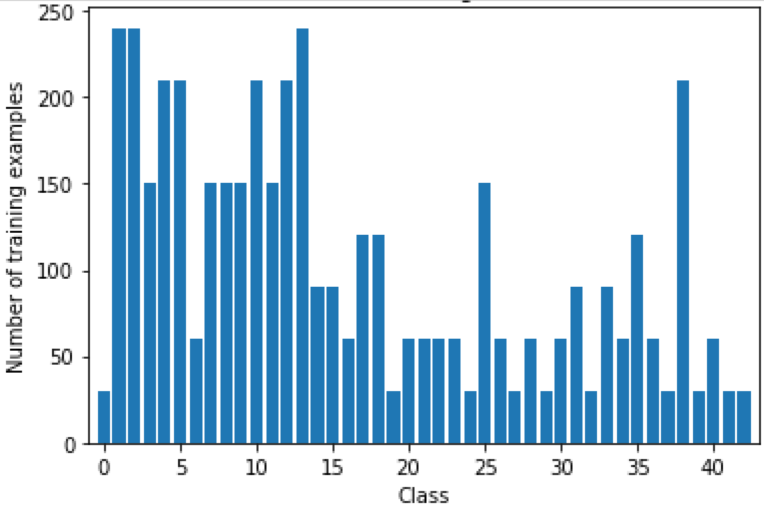


In [21]:

```
X_train, y_train = shuffle(X_train_aug, y_train_aug)
```

In [22]:

```
log('augmenting valid')
X_valid_aug, y_valid_aug = augment_dataset(X_valid, y_valid)
```



In [23]:

```
X_valid, y_valid = X_valid_aug, y_valid_aug
```

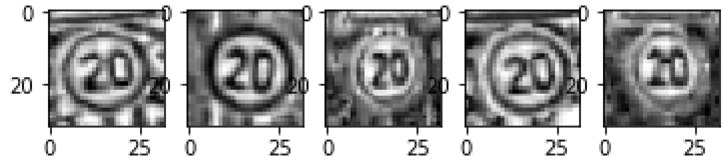
In [24]:

```
save_data()
```

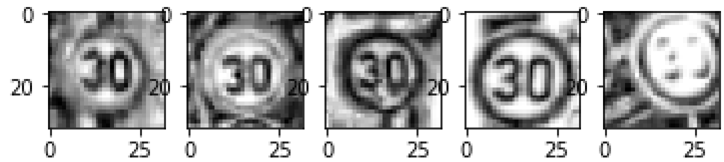
In [25]:

```
plot_images_for_class(X_train, y_train, max_classes = 43)
```

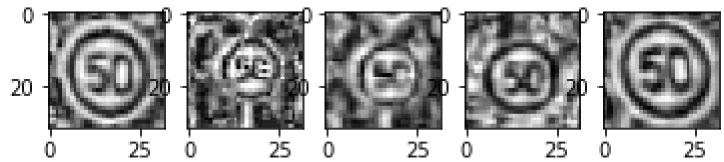
```
2017.09.01, 18:02:24: -------------------------------------------------
-------------
2017.09.01, 18:02:24: Examples for class 0, "Speed limit (20km/h)" (18
0 samples)
```



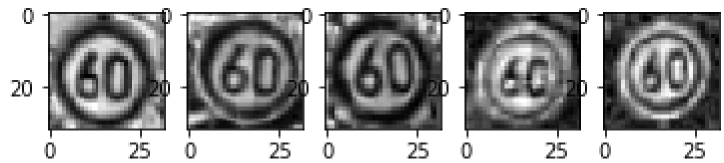
```
2017.09.01, 18:02:24: -------------------------------------------------
-------------
2017.09.01, 18:02:24: Examples for class 1, "Speed limit (30km/h)" (19
80 samples)
```



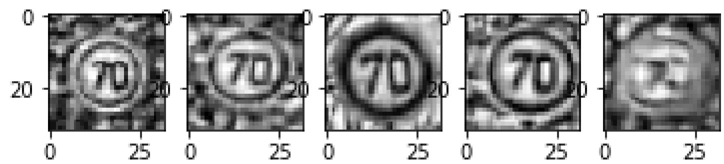
```
2017.09.01, 18:02:25: -------------------------------------------------
-------------
2017.09.01, 18:02:25: Examples for class 2, "Speed limit (50km/h)" (20
10 samples)
```



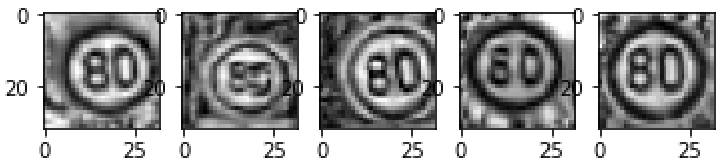
```
2017.09.01, 18:02:25: -------------------------------------------------
-------------
2017.09.01, 18:02:25: Examples for class 3, "Speed limit (60km/h)" (12
60 samples)
```



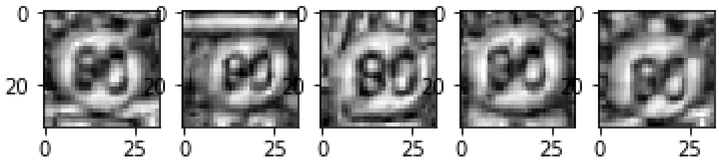
```
2017.09.01, 18:02:26: -------------------------------------------------
-------------
2017.09.01, 18:02:26: Examples for class 4, "Speed limit (70km/h)" (17
70 samples)
```



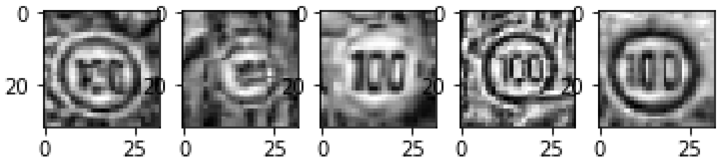
```
2017.09.01, 18:02:26: -------------------------------------------------
```



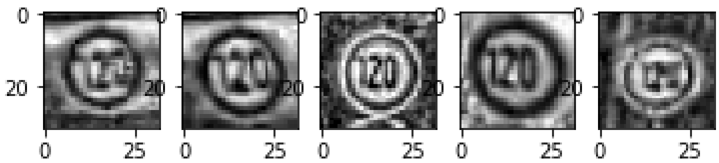
```
2017.09.01, 18:02:27: -------------------------------------------------
--------------
2017.09.01, 18:02:27: Examples for class 6, "End of speed limit (80km/
h)" (360 samples)
```



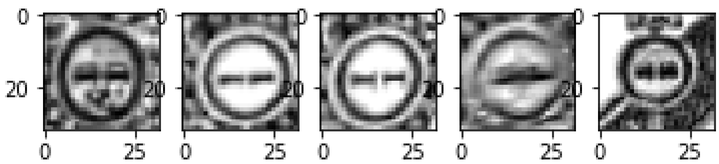
```
2017.09.01, 18:02:27: -------------------------------------------------
--------------
2017.09.01, 18:02:27: Examples for class 7, "Speed limit (100km/h)" (1
290 samples)
```



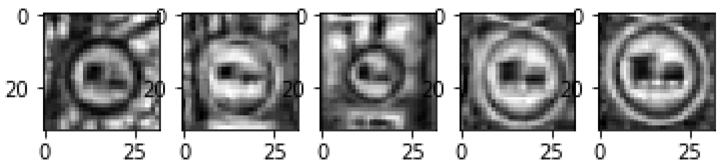
```
2017.09.01, 18:02:28: -------------------------------------------------
--------------
2017.09.01, 18:02:28: Examples for class 8, "Speed limit (120km/h)" (1
260 samples)
```



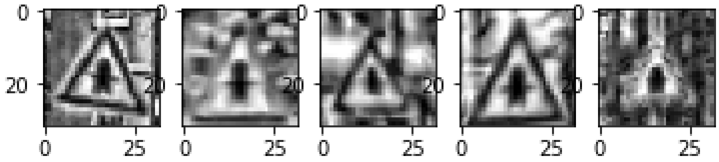
```
2017.09.01, 18:02:28: -------------------------------------------------
--------------
2017.09.01, 18:02:28: Examples for class 9, "No passing" (1320 sample
s)
```



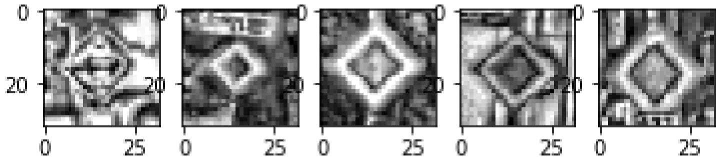
```
2017.09.01, 18:02:28: -------------------------------------------------
--------------
2017.09.01, 18:02:28: Examples for class 10, "No passing for vehicles
 over 3.5 metric tons" (1800 samples)
```



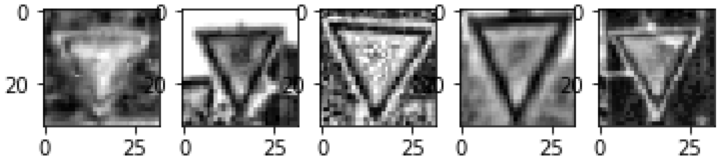
```
2017.09.01, 18:02:29: ----------------------------------------------------------------
2017.09.01, 18:02:29: Examples for class 11, "Right-of-way at the next intersection" (1170 samples)
```



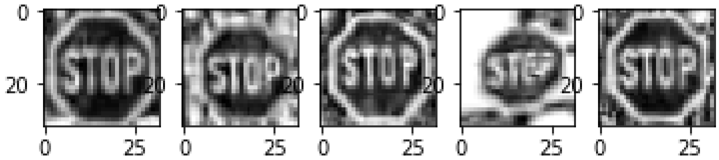
```
2017.09.01, 18:02:29: ----------------------------------------------------------------
2017.09.01, 18:02:29: Examples for class 12, "Priority road" (1890 samples)
```



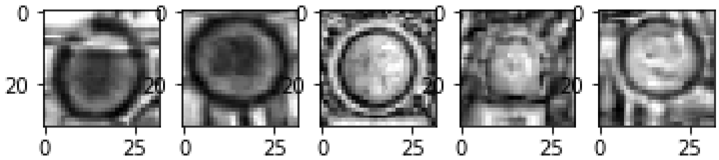
```
2017.09.01, 18:02:30: ----------------------------------------------------------------
2017.09.01, 18:02:30: Examples for class 13, "Yield" (1920 samples)
```



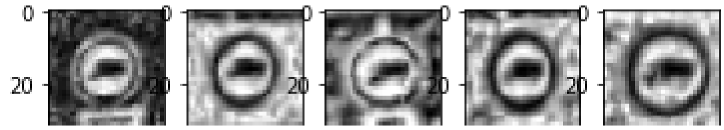
```
2017.09.01, 18:02:30: ----------------------------------------------------------------
2017.09.01, 18:02:30: Examples for class 14, "Stop" (690 samples)
```



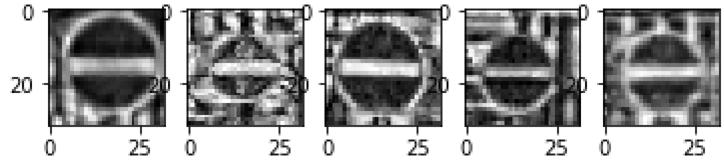
```
2017.09.01, 18:02:31: ----------------------------------------------------------------
2017.09.01, 18:02:31: Examples for class 15, "No vehicles" (540 samples)
```



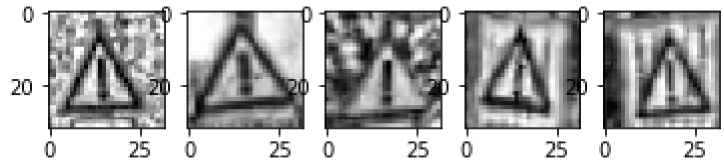
```
2017.09.01, 18:02:31: ----------------------------------------------------------------
2017.09.01, 18:02:31: Examples for class 16, "Vehicles over 3.5 metric tons prohibited" (360 samples)
```



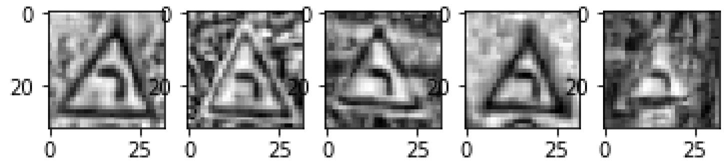
```
2017.09.01, 18:02:32: ------------------------------------------------------------
--------------
2017.09.01, 18:02:32: Examples for class 17, "No entry" (990 samples)
```



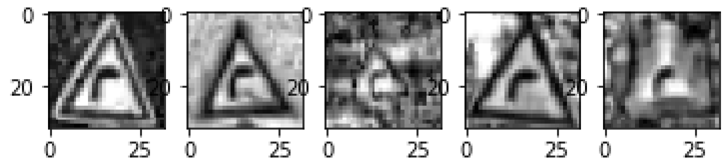
```
2017.09.01, 18:02:32: ------------------------------------------------------------
--------------
2017.09.01, 18:02:32: Examples for class 18, "General caution" (1080 s
amples)
```



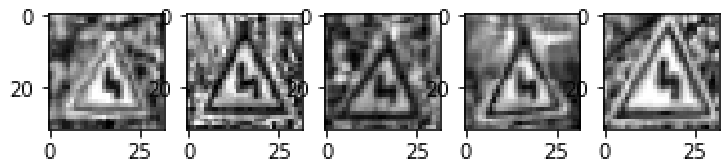
```
2017.09.01, 18:02:33: ------------------------------------------------------------
--------------
2017.09.01, 18:02:33: Examples for class 19, "Dangerous curve to the l
eft" (180 samples)
```



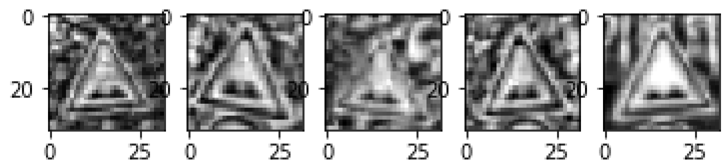
```
2017.09.01, 18:02:33: ------------------------------------------------------------
--------------
2017.09.01, 18:02:33: Examples for class 20, "Dangerous curve to the r
ight" (300 samples)
```



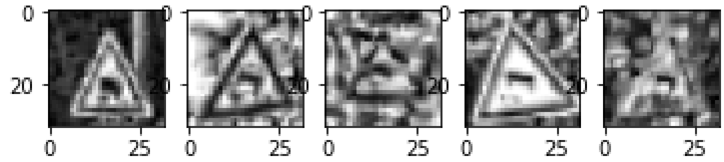
```
2017.09.01, 18:02:33: ------------------------------------------------------------
--------------
2017.09.01, 18:02:33: Examples for class 21, "Double curve" (270 sampl
es)
```



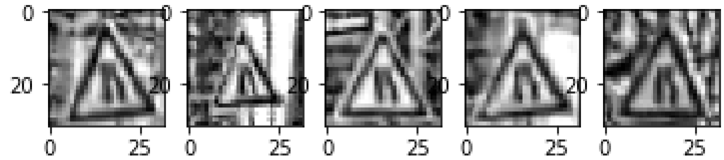
```
2017.09.01, 18:02:34: ------------------------------------------------------------
--------------
2017.09.01, 18:02:34: Examples for class 22, "Bumpy road" (330 sample
s)
```



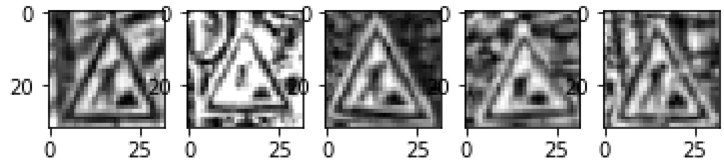
```
2017.09.01, 18:02:34: -------------------------------------------------------
--------------
2017.09.01, 18:02:34: Examples for class 23, "Slippery road" (450 samp
les)
```



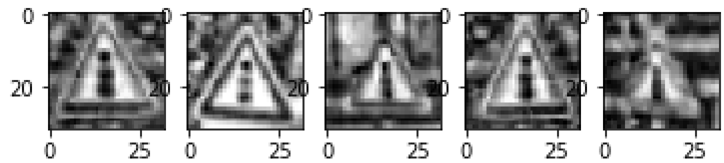
```
2017.09.01, 18:02:35: -------------------------------------------------------
--------------
2017.09.01, 18:02:35: Examples for class 24, "Road narrows on the righ
t" (240 samples)
```



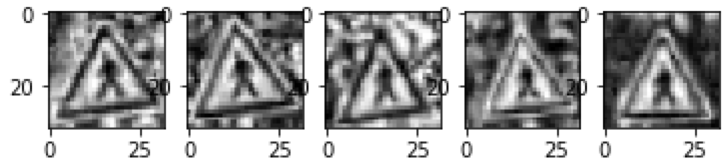
```
2017.09.01, 18:02:35: -------------------------------------------------------
--------------
2017.09.01, 18:02:35: Examples for class 25, "Road work" (1350 sample
s)
```



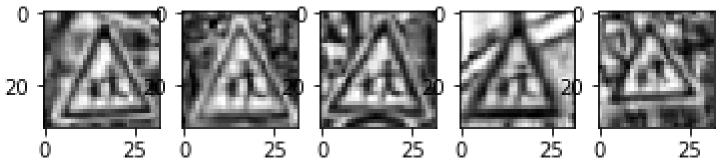
```
2017.09.01, 18:02:35: -------------------------------------------------------
--------------
2017.09.01, 18:02:35: Examples for class 26, "Traffic signals" (540 sa
mples)
```



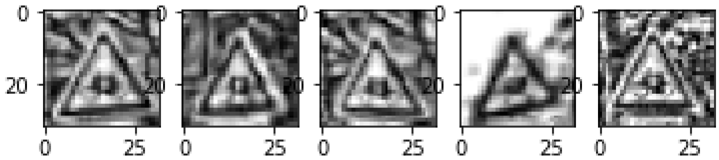
```
2017.09.01, 18:02:36: -------------------------------------------------------
--------------
2017.09.01, 18:02:36: Examples for class 27, "Pedestrians" (210 sample
s)
```



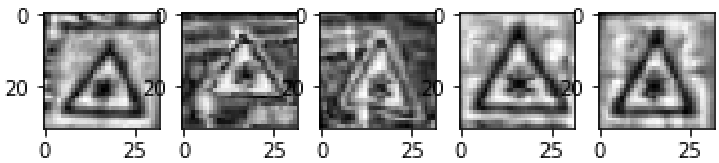
```
2017.09.01, 18:02:36: ------------------------------------------------
--------------
2017.09.01, 18:02:36: Examples for class 28, "Children crossing" (480
 samples)
```



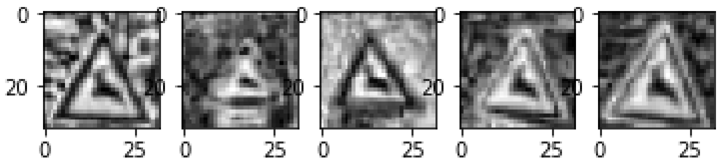
```
2017.09.01, 18:02:37: ------------------------------------------------
--------------
2017.09.01, 18:02:37: Examples for class 29, "Bicycles crossing" (240
 samples)
```



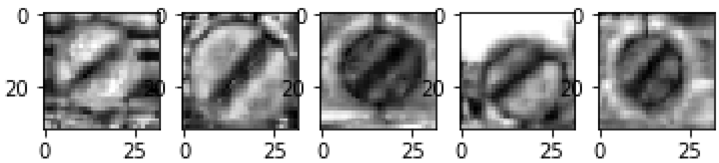
```
2017.09.01, 18:02:37: ------------------------------------------------
--------------
2017.09.01, 18:02:37: Examples for class 30, "Beware of ice/snow" (390
 samples)
```



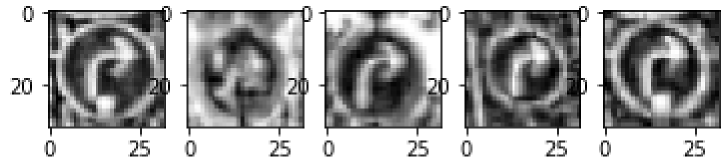
```
2017.09.01, 18:02:37: ------------------------------------------------
--------------
2017.09.01, 18:02:37: Examples for class 31, "Wild animals crossing"
 (690 samples)
```



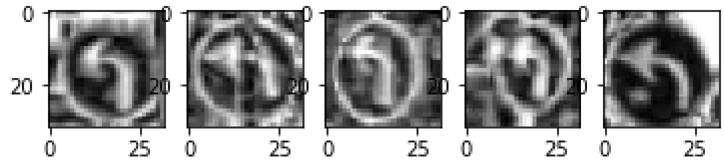
```
2017.09.01, 18:02:38: ------------------------------------------------
--------------
2017.09.01, 18:02:38: Examples for class 32, "End of all speed and pas
sing limits" (210 samples)
```



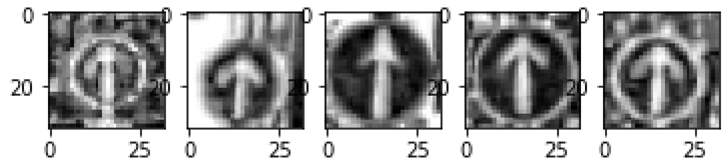
```
2017.09.01, 18:02:38: -----------------------------------------------------
----------------
2017.09.01, 18:02:38: Examples for class 33, "Turn right ahead" (599
```



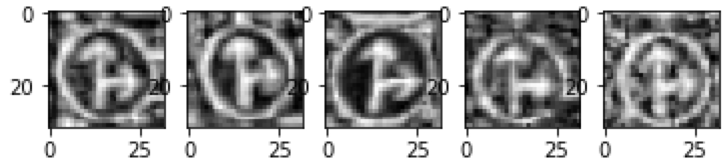
```
2017.09.01, 18:02:39: ------------------------------------------------------
--------------
2017.09.01, 18:02:39: Examples for class 34, "Turn left ahead" (360 sa
mples)
```



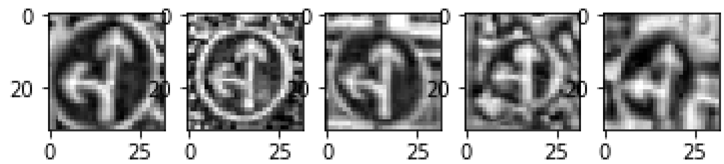
```
2017.09.01, 18:02:39: ------------------------------------------------------
--------------
2017.09.01, 18:02:39: Examples for class 35, "Ahead only" (1080 sample
s)
```



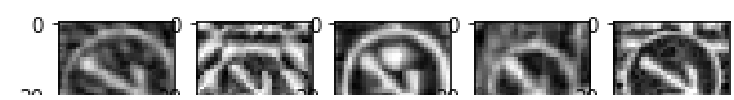
```
2017.09.01, 18:02:39: ------------------------------------------------------
--------------
2017.09.01, 18:02:39: Examples for class 36, "Go straight or right" (3
30 samples)
```



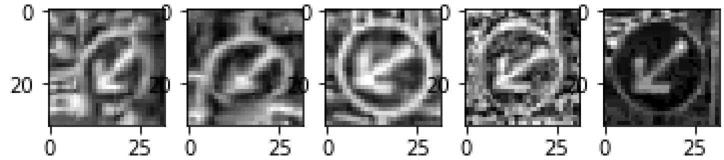
```
2017.09.01, 18:02:40: ------------------------------------------------------
--------------
2017.09.01, 18:02:40: Examples for class 37, "Go straight or left" (18
0 samples)
```



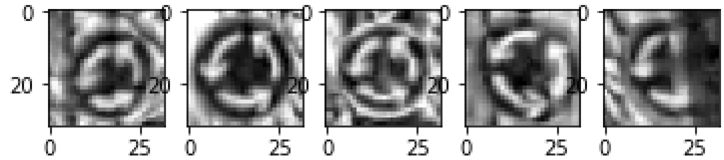
```
2017.09.01, 18:02:40: ------------------------------------------------------
--------------
2017.09.01, 18:02:40: Examples for class 38, "Keep right" (1860 sample
s)
```



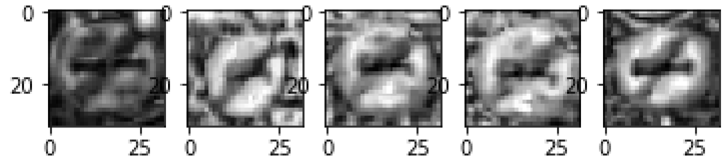
```
2017.09.01, 18:02:41: -----------------------------------------------------
--------------
2017.09.01, 18:02:41: Examples for class 39, "Keep left" (270 samples)
```



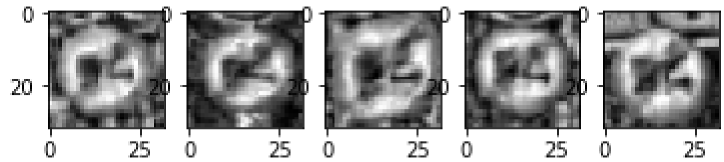
```
2017.09.01, 18:02:41: -----------------------------------------------------
--------------
2017.09.01, 18:02:41: Examples for class 40, "Roundabout mandatory" (3
00 samples)
```



```
2017.09.01, 18:02:41: -----------------------------------------------------
--------------
2017.09.01, 18:02:41: Examples for class 41, "End of no passing" (210
 samples)
```



```
2017.09.01, 18:02:42: -----------------------------------------------------
--------------
2017.09.01, 18:02:42: Examples for class 42, "End of no passing by veh
icles over 3.5 metric tons" (210 samples)
```



Model Architecture

In [26]:

```
import tensorflow as tf

n_classes = 43
image_depth = 1
```

In [27]:

```
from tensorflow.contrib.layers import flatten

def conv2d(input, filter_width, filter_height, stage_name = "default", input_depth
                 stride = 2):
    # This solution uses the tf.truncated_normal() function to initialize the weigh
    # Using the default mean and standard deviation from tf.truncated_normal() is f
    # these hyperparameters can result in better performance.
    mu = 0
    sigma = 0.1

    # Filter (weights and bias)
    # The shape of the filter weight is (height, width, input_depth, output_depth)
    # The shape of the filter bias is (output_depth,)
    # TODO: Define the filter weights `F_W` and filter bias `F_b`.
    # NOTE: Remember to wrap them in ` tf.Variable`, they are trainable parameters
    F_W = tf.Variable(tf.truncated_normal(shape = (filter_width, filter_height, inp
                                          mean = mu,
                                          stddev = sigma), name = "weight_" + stage_
    tf.add_to_collection(tf.GraphKeys.REGULARIZATION_LOSSES, F_W)
    F_b = tf.Variable(tf.zeros(output_depth), name = "bias_" + stage_name)
    # TODO: Set the stride for each dimension (batch_size, height, width, depth)
    strides = [1, stride, stride, 1]
    # TODO: set the padding, either 'VALID' or 'SAME'.
    padding = 'VALID'
    # https://www.tensorflow.org/versions/r0.11/api_docs/python/nn.html#conv2d
    # `tf.nn.conv2d` does not include the bias computation so we have to add it our
    return tf.nn.conv2d(input, F_W, strides, padding) + F_b

def maxpool(input, filter_height, filter_width, stride):
    # TODO: Set the ksize (filter size) for each dimension (batch_size, height, wid
    ksize = [1, filter_height, filter_width, 1]
    # TODO: Set the stride for each dimension (batch_size, height, width, depth)
    strides = [1, stride, stride, 1]
    # TODO: set the padding, either 'VALID' or 'SAME'.
    padding = 'VALID'
    # https://www.tensorflow.org/versions/r0.11/api_docs/python/nn.html#max_pool
    return tf.nn.max_pool(input, ksize, strides, padding)


def LeNet3(x, keep_prob = tf.placeholder(tf.float32), n_classes = 43, original_dept

    # Arguments used for tf.truncated_normal, randomly defines variables for the we
    mu = 0
    sigma = 0.1

    # TODO: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    layer1_conv = conv2d(x, 5, 5, stage_name = "layer1_conv", input_depth = origina

    # TODO: Activation.
    # Use predefined relu function as activation
    layer1_relu = tf.nn.relu(layer1_conv)

    # TODO: Pooling. Input = 28x28x6. Output = 14x14x6.
    layer1_pooled = maxpool(layer1_relu, filter_height = 2, filter_width = 2, strid

    # TODO: Layer 2: Convolutional. Output = 10x10x16.
    layer2_conv = conv2d(layer1_pooled, filter_height = 5, stage_name = "layer2_con

    # TODO: Activation.
```

```
    layer2_relu = tf.nn.relu(layer2_conv)

    # UPGRADE: Dropout
    layer2_relu = tf.nn.dropout(layer2_relu, keep_prob)

    # TODO: Pooling. Input = 10x10x16. Output = 5x5x16.
    layer2_pooled = maxpool(layer2_relu, filter_height = 2, filter_width = 2, strid

    # TODO: Flatten. Input = 5x5x16. Output = 400.
    flattened = tf.contrib.layers.flatten(layer2_pooled)

    # TODO: Layer 3: Fully Connected. Input = 400. Output = 120.
    layer3 = tf.contrib.layers.fully_connected(flattened, 120)

    # TODO: Activation.
    layer3_relu = tf.nn.relu(layer3)

    # UPGRADE: Dropout
    layer3_relu = tf.nn.dropout(layer3_relu, keep_prob)

    # SOLUTION: Layer 4: Fully Connected. Input = 120. Output = 84.
    fc2_W  = tf.Variable(tf.truncated_normal(shape=(120, 84), mean = mu, stddev = s
    fc2_b  = tf.Variable(tf.zeros(84), name= 'bias_fc2_b')
    layer4    = tf.matmul(layer3_relu, fc2_W) + fc2_b

    # TODO: Activation.
    layer4_relu = tf.nn.relu(layer4)

    # SOLUTION: Layer 5: Fully Connected. Input = 84. Output = 10.
    fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, n_classes), mean = mu, stdd
    fc3_b  = tf.Variable(tf.zeros(n_classes), name= 'bias_fc3_b')
    logits = tf.matmul(layer4_relu, fc3_W) + fc3_b

    return logits
```

In [28]:

```
x = tf.placeholder(tf.float32, (None, 32, 32, image_depth))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, n_classes)
```

Train, Validate and Test the Model

A validation set can be used to assess how well the model is performing. A low accuracy on the training and validation sets imply underfitting. A high accuracy on the training set but low accuracy on the validation set implies overfitting.

Train

In [29]:

```
rate_coarse = 0.0005
rate_fine = 0.00005
EPOCHS = 30
BATCH_SIZE = 256
BETA = 0.001 # This is a good beta value to start with for regularization

keep_prob = tf.placeholder(tf.float32)
logits = LeNet3(x, keep_prob, n_classes= n_classes, original_depth= image_depth)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, logits=lo
loss_operation = tf.reduce_mean(cross_entropy)

# EXPERIMENTAL
# Loss function using L2 Regularization
trainable_vars   = tf.trainable_variables()
lossL2 = tf.add_n([ tf.nn.l2_loss(v) for v in trainable_vars
                    if 'bias' not in v.name ])
#weights = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
#regularizer = tf.nn.l2_loss(weights)
loss_operation = tf.reduce_mean(loss_operation + BETA * lossL2)
# EXPERIMENTAL

optimizer = tf.train.AdamOptimizer(learning_rate = rate_coarse)
training_operation = optimizer.minimize(loss_operation)
```

In [30]:

```
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_y, 1))
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
saver = tf.train.Saver()

def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0
    sess = tf.get_default_session()
    for offset in range(0, num_examples, BATCH_SIZE):
        batch_x, batch_y = X_data[offset:offset+BATCH_SIZE], y_data[offset:offset+B
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: batch_y,
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

In [47]:

```
def train_net(sess, X_train, y_train):
    accuracies = np.zeros(n_auto_stop).tolist()

    sess.run(tf.global_variables_initializer())
    num_examples = len(X_train)

    log("Training...")
    for i in range(EPOCHS):
        X_train, y_train = shuffle(X_train, y_train)
        for offset in range(0, num_examples, BATCH_SIZE):
            end = offset + BATCH_SIZE
            batch_x, batch_y = X_train[offset:end], y_train[offset:end]
            sess.run(training_operation, feed_dict={x: batch_x, y: batch_y, keep_pr

        validation_accuracy = evaluate(X_valid, y_valid)
        log("EPOCH {} ...".format(i+1))
        log("Validation Accuracy = {:.3f}".format(validation_accuracy))
        if len(accuracies) > n_auto_stop:
            accuracies = accuracies[-n_auto_stop:] # take last n_auto_stop elements
        #if round(min(accuracies)*100) >= round(validation_accuracy * 100):
        #    log("accuracy stopped increasing more than 0.1. stopping...")
        #    break
        #else:
        #    log("minimum accuracy during last {0} epochs was {1:.3}, tends to incr
        #        n_auto_stop, min(accuracies)))

        accuracies.append(validation_accuracy)

    saver.save(sess, './lenet')
    log("Model saved")
```

In [48]:

```
# train again with finer settings
optimizer = tf.train.AdamOptimizer(learning_rate = rate_fine)
training_operation = optimizer.minimize(loss_operation)
n_auto_stop = 10
EPOCHS = 500

with tf.Session() as sess:
    train_net(sess, X_train, y_train)
```

```
2017.09.02, 10:00:03: Training...
2017.09.02, 10:00:25: EPOCH 1 ...
2017.09.02, 10:00:25: Validation Accuracy = 0.551
2017.09.02, 10:00:48: EPOCH 2 ...
2017.09.02, 10:00:48: Validation Accuracy = 0.686
2017.09.02, 10:01:12: EPOCH 3 ...
2017.09.02, 10:01:12: Validation Accuracy = 0.746
2017.09.02, 10:01:36: EPOCH 4 ...
2017.09.02, 10:01:36: Validation Accuracy = 0.790
2017.09.02, 10:02:00: EPOCH 5 ...
2017.09.02, 10:02:00: Validation Accuracy = 0.819
2017.09.02, 10:02:23: EPOCH 6 ...
2017.09.02, 10:02:23: Validation Accuracy = 0.836
2017.09.02, 10:02:47: EPOCH 7 ...
2017.09.02, 10:02:47: Validation Accuracy = 0.853
2017.09.02, 10:03:10: EPOCH 8 ...
2017.09.02, 10:03:10: Validation Accuracy = 0.869
2017.09.02, 10:03:34: EPOCH 9 ...
2017.09.02, 10:03:34: Validation Accuracy = 0.874
2017.09.02, 10:03:57: EPOCH 10 ...
2017.09.02, 10:03:57: Validation Accuracy = 0.885
2017.09.02, 10:04:21: EPOCH 11 ...
2017.09.02, 10:04:21: Validation Accuracy = 0.888
2017.09.02, 10:04:44: EPOCH 12 ...
2017.09.02, 10:04:44: Validation Accuracy = 0.899
2017.09.02, 10:05:08: EPOCH 13 ...
2017.09.02, 10:05:08: Validation Accuracy = 0.900
2017.09.02, 10:05:32: EPOCH 14 ...
2017.09.02, 10:05:32: Validation Accuracy = 0.906
2017.09.02, 10:05:55: EPOCH 15 ...
2017.09.02, 10:05:55: Validation Accuracy = 0.907
2017.09.02, 10:06:19: EPOCH 16 ...
2017.09.02, 10:06:19: Validation Accuracy = 0.913
2017.09.02, 10:06:42: EPOCH 17 ...
2017.09.02, 10:06:42: Validation Accuracy = 0.915
2017.09.02, 10:07:06: EPOCH 18 ...
2017.09.02, 10:07:06: Validation Accuracy = 0.915
2017.09.02, 10:07:29: EPOCH 19 ...
2017.09.02, 10:07:29: Validation Accuracy = 0.920
2017.09.02, 10:07:53: EPOCH 20 ...
2017.09.02, 10:07:53: Validation Accuracy = 0.924
2017.09.02, 10:08:16: EPOCH 21 ...
2017.09.02, 10:08:16: Validation Accuracy = 0.923
2017.09.02, 10:08:40: EPOCH 22 ...
2017.09.02, 10:08:40: Validation Accuracy = 0.924
2017.09.02, 10:09:03: EPOCH 23 ...
2017.09.02, 10:09:03: Validation Accuracy = 0.927
2017.09.02, 10:09:27: EPOCH 24 ...
2017.09.02, 10:09:27: Validation Accuracy = 0.929
2017.09.02, 10:09:50: EPOCH 25 ...
2017.09.02, 10:09:50: Validation Accuracy = 0.930
2017.09.02, 10:10:14: EPOCH 26 ...
2017.09.02, 10:10:14: Validation Accuracy = 0.933
2017.09.02, 10:10:38: EPOCH 27 ...
2017.09.02, 10:10:38: Validation Accuracy = 0.933
2017.09.02, 10:11:01: EPOCH 28 ...
2017.09.02, 10:11:01: Validation Accuracy = 0.933
2017.09.02, 10:11:26: EPOCH 29 ...
2017.09.02, 10:11:26: Validation Accuracy = 0.937
2017.09.02, 10:11:50: EPOCH 30 ...
2017.09.02, 10:11:50: Validation Accuracy = 0.935
```

```
2017.09.02, 10:12:14: EPOCH 31 ...
2017.09.02, 10:12:14: Validation Accuracy = 0.937
2017.09.02, 10:12:39: EPOCH 32 ...
2017.09.02, 10:12:39: Validation Accuracy = 0.937
2017.09.02, 10:13:03: EPOCH 33 ...
2017.09.02, 10:13:03: Validation Accuracy = 0.937
2017.09.02, 10:13:27: EPOCH 34 ...
2017.09.02, 10:13:27: Validation Accuracy = 0.939
2017.09.02, 10:13:51: EPOCH 35 ...
2017.09.02, 10:13:51: Validation Accuracy = 0.939
2017.09.02, 10:14:16: EPOCH 36 ...
2017.09.02, 10:14:16: Validation Accuracy = 0.939
2017.09.02, 10:14:40: EPOCH 37 ...
2017.09.02, 10:14:40: Validation Accuracy = 0.938
2017.09.02, 10:15:04: EPOCH 38 ...
2017.09.02, 10:15:04: Validation Accuracy = 0.941
2017.09.02, 10:15:28: EPOCH 39 ...
2017.09.02, 10:15:28: Validation Accuracy = 0.941
2017.09.02, 10:15:53: EPOCH 40 ...
2017.09.02, 10:15:53: Validation Accuracy = 0.942
2017.09.02, 10:16:17: EPOCH 41 ...
2017.09.02, 10:16:17: Validation Accuracy = 0.942
2017.09.02, 10:16:41: EPOCH 42 ...
2017.09.02, 10:16:41: Validation Accuracy = 0.942
2017.09.02, 10:17:05: EPOCH 43 ...
2017.09.02, 10:17:05: Validation Accuracy = 0.942
2017.09.02, 10:17:30: EPOCH 44 ...
2017.09.02, 10:17:30: Validation Accuracy = 0.945
2017.09.02, 10:17:54: EPOCH 45 ...
2017.09.02, 10:17:54: Validation Accuracy = 0.945
2017.09.02, 10:18:18: EPOCH 46 ...
2017.09.02, 10:18:18: Validation Accuracy = 0.944
2017.09.02, 10:18:43: EPOCH 47 ...
2017.09.02, 10:18:43: Validation Accuracy = 0.944
2017.09.02, 10:19:07: EPOCH 48 ...
2017.09.02, 10:19:07: Validation Accuracy = 0.946
2017.09.02, 10:19:31: EPOCH 49 ...
2017.09.02, 10:19:31: Validation Accuracy = 0.947
2017.09.02, 10:19:55: EPOCH 50 ...
2017.09.02, 10:19:55: Validation Accuracy = 0.947
2017.09.02, 10:20:20: EPOCH 51 ...
2017.09.02, 10:20:20: Validation Accuracy = 0.947
2017.09.02, 10:20:44: EPOCH 52 ...
2017.09.02, 10:20:44: Validation Accuracy = 0.948
2017.09.02, 10:21:08: EPOCH 53 ...
2017.09.02, 10:21:08: Validation Accuracy = 0.948
2017.09.02, 10:21:33: EPOCH 54 ...
2017.09.02, 10:21:33: Validation Accuracy = 0.946
2017.09.02, 10:21:57: EPOCH 55 ...
2017.09.02, 10:21:57: Validation Accuracy = 0.949
2017.09.02, 10:22:21: EPOCH 56 ...
2017.09.02, 10:22:21: Validation Accuracy = 0.949
2017.09.02, 10:22:45: EPOCH 57 ...
2017.09.02, 10:22:45: Validation Accuracy = 0.947
2017.09.02, 10:23:09: EPOCH 58 ...
2017.09.02, 10:23:09: Validation Accuracy = 0.950
2017.09.02, 10:23:34: EPOCH 59 ...
2017.09.02, 10:23:34: Validation Accuracy = 0.951
2017.09.02, 10:23:58: EPOCH 60 ...
2017.09.02, 10:23:58: Validation Accuracy = 0.951
2017.09.02, 10:24:22: EPOCH 61 ...
```

```
2017.09.02, 10:24:22: Validation Accuracy = 0.949
2017.09.02, 10:24:46: EPOCH 62 ...
2017.09.02, 10:24:46: Validation Accuracy = 0.950
2017.09.02, 10:25:11: EPOCH 63 ...
2017.09.02, 10:25:11: Validation Accuracy = 0.952
2017.09.02, 10:25:35: EPOCH 64 ...
2017.09.02, 10:25:35: Validation Accuracy = 0.952
2017.09.02, 10:25:59: EPOCH 65 ...
2017.09.02, 10:25:59: Validation Accuracy = 0.952
2017.09.02, 10:26:24: EPOCH 66 ...
2017.09.02, 10:26:24: Validation Accuracy = 0.951
2017.09.02, 10:26:48: EPOCH 67 ...
2017.09.02, 10:26:48: Validation Accuracy = 0.952
2017.09.02, 10:27:12: EPOCH 68 ...
2017.09.02, 10:27:12: Validation Accuracy = 0.951
2017.09.02, 10:27:36: EPOCH 69 ...
2017.09.02, 10:27:36: Validation Accuracy = 0.953
2017.09.02, 10:28:01: EPOCH 70 ...
2017.09.02, 10:28:01: Validation Accuracy = 0.953
2017.09.02, 10:28:25: EPOCH 71 ...
2017.09.02, 10:28:25: Validation Accuracy = 0.951
2017.09.02, 10:28:49: EPOCH 72 ...
2017.09.02, 10:28:49: Validation Accuracy = 0.953
2017.09.02, 10:29:13: EPOCH 73 ...
2017.09.02, 10:29:13: Validation Accuracy = 0.953
2017.09.02, 10:29:38: EPOCH 74 ...
2017.09.02, 10:29:38: Validation Accuracy = 0.953
2017.09.02, 10:30:02: EPOCH 75 ...
2017.09.02, 10:30:02: Validation Accuracy = 0.952
2017.09.02, 10:30:26: EPOCH 76 ...
2017.09.02, 10:30:26: Validation Accuracy = 0.953
2017.09.02, 10:30:50: EPOCH 77 ...
2017.09.02, 10:30:50: Validation Accuracy = 0.953
2017.09.02, 10:31:15: EPOCH 78 ...
2017.09.02, 10:31:15: Validation Accuracy = 0.955
2017.09.02, 10:31:39: EPOCH 79 ...
2017.09.02, 10:31:39: Validation Accuracy = 0.955
2017.09.02, 10:32:03: EPOCH 80 ...
2017.09.02, 10:32:03: Validation Accuracy = 0.954
2017.09.02, 10:32:27: EPOCH 81 ...
2017.09.02, 10:32:27: Validation Accuracy = 0.955
2017.09.02, 10:32:52: EPOCH 82 ...
2017.09.02, 10:32:52: Validation Accuracy = 0.954
2017.09.02, 10:33:16: EPOCH 83 ...
2017.09.02, 10:33:16: Validation Accuracy = 0.952
2017.09.02, 10:33:40: EPOCH 84 ...
2017.09.02, 10:33:40: Validation Accuracy = 0.956
2017.09.02, 10:34:04: EPOCH 85 ...
2017.09.02, 10:34:04: Validation Accuracy = 0.955
2017.09.02, 10:34:28: EPOCH 86 ...
2017.09.02, 10:34:28: Validation Accuracy = 0.956
2017.09.02, 10:34:53: EPOCH 87 ...
2017.09.02, 10:34:53: Validation Accuracy = 0.956
2017.09.02, 10:35:17: EPOCH 88 ...
2017.09.02, 10:35:17: Validation Accuracy = 0.956
2017.09.02, 10:35:41: EPOCH 89 ...
2017.09.02, 10:35:41: Validation Accuracy = 0.957
2017.09.02, 10:36:05: EPOCH 90 ...
2017.09.02, 10:36:05: Validation Accuracy = 0.957
2017.09.02, 10:36:30: EPOCH 91 ...
2017.09.02, 10:36:30: Validation Accuracy = 0.956
```

```
2017.09.02, 10:36:54: EPOCH 92 ...
2017.09.02, 10:36:54: Validation Accuracy = 0.956
2017.09.02, 10:37:18: EPOCH 93 ...
2017.09.02, 10:37:18: Validation Accuracy = 0.956
2017.09.02, 10:37:42: EPOCH 94 ...
2017.09.02, 10:37:42: Validation Accuracy = 0.955
2017.09.02, 10:38:07: EPOCH 95 ...
2017.09.02, 10:38:07: Validation Accuracy = 0.957
2017.09.02, 10:38:31: EPOCH 96 ...
2017.09.02, 10:38:31: Validation Accuracy = 0.957
2017.09.02, 10:38:55: EPOCH 97 ...
2017.09.02, 10:38:55: Validation Accuracy = 0.957
```

```
2017.09.02, 10:39:20: EPOCH 98 ...
2017.09.02, 10:39:20: Validation Accuracy = 0.957
2017.09.02, 10:39:44: EPOCH 99 ...
2017.09.02, 10:39:44: Validation Accuracy = 0.955
2017.09.02, 10:40:08: EPOCH 100 ...
2017.09.02, 10:40:08: Validation Accuracy = 0.956
2017.09.02, 10:40:32: EPOCH 101 ...
2017.09.02, 10:40:32: Validation Accuracy = 0.958
2017.09.02, 10:40:57: EPOCH 102 ...
2017.09.02, 10:40:57: Validation Accuracy = 0.957
2017.09.02, 10:41:21: EPOCH 103 ...
2017.09.02, 10:41:21: Validation Accuracy = 0.958
2017.09.02, 10:41:45: EPOCH 104 ...
2017.09.02, 10:41:45: Validation Accuracy = 0.957
2017.09.02, 10:42:09: EPOCH 105 ...
2017.09.02, 10:42:09: Validation Accuracy = 0.957
2017.09.02, 10:42:33: EPOCH 106 ...
2017.09.02, 10:42:33: Validation Accuracy = 0.958
2017.09.02, 10:42:58: EPOCH 107 ...
2017.09.02, 10:42:58: Validation Accuracy = 0.958
2017.09.02, 10:43:22: EPOCH 108 ...
2017.09.02, 10:43:22: Validation Accuracy = 0.958
2017.09.02, 10:43:46: EPOCH 109 ...
2017.09.02, 10:43:46: Validation Accuracy = 0.957
2017.09.02, 10:44:10: EPOCH 110 ...
2017.09.02, 10:44:10: Validation Accuracy = 0.957
2017.09.02, 10:44:35: EPOCH 111 ...
2017.09.02, 10:44:35: Validation Accuracy = 0.958
2017.09.02, 10:44:59: EPOCH 112 ...
2017.09.02, 10:44:59: Validation Accuracy = 0.958
2017.09.02, 10:45:23: EPOCH 113 ...
2017.09.02, 10:45:23: Validation Accuracy = 0.958
2017.09.02, 10:45:48: EPOCH 114 ...
2017.09.02, 10:45:48: Validation Accuracy = 0.958
2017.09.02, 10:46:12: EPOCH 115 ...
2017.09.02, 10:46:12: Validation Accuracy = 0.958
2017.09.02, 10:46:36: EPOCH 116 ...
2017.09.02, 10:46:36: Validation Accuracy = 0.958
2017.09.02, 10:47:00: EPOCH 117 ...
2017.09.02, 10:47:00: Validation Accuracy = 0.959
2017.09.02, 10:47:25: EPOCH 118 ...
2017.09.02, 10:47:25: Validation Accuracy = 0.957
2017.09.02, 10:47:49: EPOCH 119 ...
2017.09.02, 10:47:49: Validation Accuracy = 0.959
2017.09.02, 10:48:13: EPOCH 120 ...
2017.09.02, 10:48:13: Validation Accuracy = 0.958
2017.09.02, 10:48:37: EPOCH 121 ...
2017.09.02, 10:48:37: Validation Accuracy = 0.958
2017.09.02, 10:49:02: EPOCH 122 ...
2017.09.02, 10:49:02: Validation Accuracy = 0.959
2017.09.02, 10:49:26: EPOCH 123 ...
2017.09.02, 10:49:26: Validation Accuracy = 0.958
2017.09.02, 10:49:50: EPOCH 124 ...
2017.09.02, 10:49:50: Validation Accuracy = 0.958
2017.09.02, 10:50:15: EPOCH 125 ...
2017.09.02, 10:50:15: Validation Accuracy = 0.960
2017.09.02, 10:50:39: EPOCH 126 ...
2017.09.02, 10:50:39: Validation Accuracy = 0.958
2017.09.02, 10:51:03: EPOCH 127 ...
2017.09.02, 10:51:03: Validation Accuracy = 0.959
2017.09.02, 10:51:27: EPOCH 128 ...
```

```
2017.09.02, 10:51:27: Validation Accuracy = 0.959
2017.09.02, 10:51:51: EPOCH 129 ...
2017.09.02, 10:51:51: Validation Accuracy = 0.959
2017.09.02, 10:52:16: EPOCH 130 ...
2017.09.02, 10:52:16: Validation Accuracy = 0.959
2017.09.02, 10:52:40: EPOCH 131 ...
2017.09.02, 10:52:40: Validation Accuracy = 0.959
2017.09.02, 10:53:04: EPOCH 132 ...
2017.09.02, 10:53:04: Validation Accuracy = 0.959
2017.09.02, 10:53:28: EPOCH 133 ...
2017.09.02, 10:53:28: Validation Accuracy = 0.959
2017.09.02, 10:53:53: EPOCH 134 ...
2017.09.02, 10:53:53: Validation Accuracy = 0.959
2017.09.02, 10:54:17: EPOCH 135 ...
2017.09.02, 10:54:17: Validation Accuracy = 0.959
2017.09.02, 10:54:41: EPOCH 136 ...
2017.09.02, 10:54:41: Validation Accuracy = 0.959
2017.09.02, 10:55:05: EPOCH 137 ...
2017.09.02, 10:55:05: Validation Accuracy = 0.959
2017.09.02, 10:55:30: EPOCH 138 ...
2017.09.02, 10:55:30: Validation Accuracy = 0.959
2017.09.02, 10:55:54: EPOCH 139 ...
2017.09.02, 10:55:54: Validation Accuracy = 0.959
2017.09.02, 10:56:18: EPOCH 140 ...
2017.09.02, 10:56:18: Validation Accuracy = 0.959
2017.09.02, 10:56:42: EPOCH 141 ...
2017.09.02, 10:56:42: Validation Accuracy = 0.959
2017.09.02, 10:57:07: EPOCH 142 ...
2017.09.02, 10:57:07: Validation Accuracy = 0.960
2017.09.02, 10:57:31: EPOCH 143 ...
2017.09.02, 10:57:31: Validation Accuracy = 0.960
2017.09.02, 10:57:55: EPOCH 144 ...
2017.09.02, 10:57:55: Validation Accuracy = 0.959
2017.09.02, 10:58:20: EPOCH 145 ...
2017.09.02, 10:58:20: Validation Accuracy = 0.960
2017.09.02, 10:58:44: EPOCH 146 ...
2017.09.02, 10:58:44: Validation Accuracy = 0.960
2017.09.02, 10:59:08: EPOCH 147 ...
2017.09.02, 10:59:08: Validation Accuracy = 0.960
2017.09.02, 10:59:32: EPOCH 148 ...
2017.09.02, 10:59:32: Validation Accuracy = 0.960
2017.09.02, 10:59:56: EPOCH 149 ...
2017.09.02, 10:59:56: Validation Accuracy = 0.960
2017.09.02, 11:00:21: EPOCH 150 ...
2017.09.02, 11:00:21: Validation Accuracy = 0.960
2017.09.02, 11:00:45: EPOCH 151 ...
2017.09.02, 11:00:45: Validation Accuracy = 0.960
2017.09.02, 11:01:09: EPOCH 152 ...
2017.09.02, 11:01:09: Validation Accuracy = 0.960
2017.09.02, 11:01:34: EPOCH 153 ...
2017.09.02, 11:01:34: Validation Accuracy = 0.959
2017.09.02, 11:01:58: EPOCH 154 ...
2017.09.02, 11:01:58: Validation Accuracy = 0.960
2017.09.02, 11:02:22: EPOCH 155 ...
2017.09.02, 11:02:22: Validation Accuracy = 0.960
2017.09.02, 11:02:46: EPOCH 156 ...
2017.09.02, 11:02:46: Validation Accuracy = 0.960
2017.09.02, 11:03:11: EPOCH 157 ...
2017.09.02, 11:03:11: Validation Accuracy = 0.960
2017.09.02, 11:03:35: EPOCH 158 ...
2017.09.02, 11:03:35: Validation Accuracy = 0.960
```

```
2017.09.02, 11:03:59: EPOCH 159 ...
2017.09.02, 11:03:59: Validation Accuracy = 0.960
2017.09.02, 11:04:24: EPOCH 160 ...
2017.09.02, 11:04:24: Validation Accuracy = 0.960
2017.09.02, 11:04:48: EPOCH 161 ...
2017.09.02, 11:04:48: Validation Accuracy = 0.960
2017.09.02, 11:05:12: EPOCH 162 ...
2017.09.02, 11:05:12: Validation Accuracy = 0.960
2017.09.02, 11:05:36: EPOCH 163 ...
2017.09.02, 11:05:36: Validation Accuracy = 0.960
2017.09.02, 11:06:01: EPOCH 164 ...
2017.09.02, 11:06:01: Validation Accuracy = 0.959
2017.09.02, 11:06:25: EPOCH 165 ...
2017.09.02, 11:06:25: Validation Accuracy = 0.960
2017.09.02, 11:06:49: EPOCH 166 ...
2017.09.02, 11:06:49: Validation Accuracy = 0.960
2017.09.02, 11:07:14: EPOCH 167 ...
2017.09.02, 11:07:14: Validation Accuracy = 0.961
2017.09.02, 11:07:38: EPOCH 168 ...
2017.09.02, 11:07:38: Validation Accuracy = 0.960
2017.09.02, 11:08:02: EPOCH 169 ...
2017.09.02, 11:08:02: Validation Accuracy = 0.961
2017.09.02, 11:08:26: EPOCH 170 ...
2017.09.02, 11:08:26: Validation Accuracy = 0.961
2017.09.02, 11:08:50: EPOCH 171 ...
2017.09.02, 11:08:50: Validation Accuracy = 0.960
2017.09.02, 11:09:15: EPOCH 172 ...
2017.09.02, 11:09:15: Validation Accuracy = 0.960
2017.09.02, 11:09:39: EPOCH 173 ...
2017.09.02, 11:09:39: Validation Accuracy = 0.961
2017.09.02, 11:10:03: EPOCH 174 ...
2017.09.02, 11:10:03: Validation Accuracy = 0.962
2017.09.02, 11:10:27: EPOCH 175 ...
2017.09.02, 11:10:27: Validation Accuracy = 0.961
2017.09.02, 11:10:52: EPOCH 176 ...
2017.09.02, 11:10:52: Validation Accuracy = 0.961
2017.09.02, 11:11:16: EPOCH 177 ...
2017.09.02, 11:11:16: Validation Accuracy = 0.961
2017.09.02, 11:11:40: EPOCH 178 ...
2017.09.02, 11:11:40: Validation Accuracy = 0.961
2017.09.02, 11:12:05: EPOCH 179 ...
2017.09.02, 11:12:05: Validation Accuracy = 0.960
2017.09.02, 11:12:29: EPOCH 180 ...
2017.09.02, 11:12:29: Validation Accuracy = 0.961
2017.09.02, 11:12:53: EPOCH 181 ...
2017.09.02, 11:12:53: Validation Accuracy = 0.961
2017.09.02, 11:13:17: EPOCH 182 ...
2017.09.02, 11:13:17: Validation Accuracy = 0.960
2017.09.02, 11:13:41: EPOCH 183 ...
2017.09.02, 11:13:41: Validation Accuracy = 0.961
2017.09.02, 11:14:06: EPOCH 184 ...
2017.09.02, 11:14:06: Validation Accuracy = 0.963
2017.09.02, 11:14:30: EPOCH 185 ...
2017.09.02, 11:14:30: Validation Accuracy = 0.962
2017.09.02, 11:14:54: EPOCH 186 ...
2017.09.02, 11:14:54: Validation Accuracy = 0.961
2017.09.02, 11:15:19: EPOCH 187 ...
2017.09.02, 11:15:19: Validation Accuracy = 0.962
2017.09.02, 11:15:43: EPOCH 188 ...
2017.09.02, 11:15:43: Validation Accuracy = 0.961
2017.09.02, 11:16:07: EPOCH 189 ...
```

```
2017.09.02, 11:16:07: Validation Accuracy = 0.960
2017.09.02, 11:16:31: EPOCH 190 ...
2017.09.02, 11:16:31: Validation Accuracy = 0.962
2017.09.02, 11:16:56: EPOCH 191 ...
2017.09.02, 11:16:56: Validation Accuracy = 0.961
2017.09.02, 11:17:20: EPOCH 192 ...
2017.09.02, 11:17:20: Validation Accuracy = 0.962
2017.09.02, 11:17:44: EPOCH 193 ...
2017.09.02, 11:17:44: Validation Accuracy = 0.961
```

```
2017.09.02, 11:18:08: EPOCH 194 ...
2017.09.02, 11:18:08: Validation Accuracy = 0.962
2017.09.02, 11:18:33: EPOCH 195 ...
2017.09.02, 11:18:33: Validation Accuracy = 0.962
2017.09.02, 11:18:57: EPOCH 196 ...
2017.09.02, 11:18:57: Validation Accuracy = 0.960
2017.09.02, 11:19:21: EPOCH 197 ...
2017.09.02, 11:19:21: Validation Accuracy = 0.961
2017.09.02, 11:19:46: EPOCH 198 ...
2017.09.02, 11:19:46: Validation Accuracy = 0.961
2017.09.02, 11:20:10: EPOCH 199 ...
2017.09.02, 11:20:10: Validation Accuracy = 0.961
2017.09.02, 11:20:34: EPOCH 200 ...
2017.09.02, 11:20:34: Validation Accuracy = 0.961
2017.09.02, 11:20:58: EPOCH 201 ...
2017.09.02, 11:20:58: Validation Accuracy = 0.961
2017.09.02, 11:21:23: EPOCH 202 ...
2017.09.02, 11:21:23: Validation Accuracy = 0.961
2017.09.02, 11:21:47: EPOCH 203 ...
2017.09.02, 11:21:47: Validation Accuracy = 0.961
2017.09.02, 11:22:11: EPOCH 204 ...
2017.09.02, 11:22:11: Validation Accuracy = 0.961
2017.09.02, 11:22:35: EPOCH 205 ...
2017.09.02, 11:22:35: Validation Accuracy = 0.961
2017.09.02, 11:23:00: EPOCH 206 ...
2017.09.02, 11:23:00: Validation Accuracy = 0.961
2017.09.02, 11:23:24: EPOCH 207 ...
2017.09.02, 11:23:24: Validation Accuracy = 0.962
2017.09.02, 11:23:48: EPOCH 208 ...
2017.09.02, 11:23:48: Validation Accuracy = 0.960
2017.09.02, 11:24:12: EPOCH 209 ...
2017.09.02, 11:24:12: Validation Accuracy = 0.960
2017.09.02, 11:24:37: EPOCH 210 ...
2017.09.02, 11:24:37: Validation Accuracy = 0.961
2017.09.02, 11:25:01: EPOCH 211 ...
2017.09.02, 11:25:01: Validation Accuracy = 0.961
2017.09.02, 11:25:25: EPOCH 212 ...
2017.09.02, 11:25:25: Validation Accuracy = 0.961
2017.09.02, 11:25:49: EPOCH 213 ...
2017.09.02, 11:25:49: Validation Accuracy = 0.961
2017.09.02, 11:26:14: EPOCH 214 ...
2017.09.02, 11:26:14: Validation Accuracy = 0.961
2017.09.02, 11:26:38: EPOCH 215 ...
2017.09.02, 11:26:38: Validation Accuracy = 0.962
2017.09.02, 11:27:02: EPOCH 216 ...
2017.09.02, 11:27:02: Validation Accuracy = 0.963
2017.09.02, 11:27:26: EPOCH 217 ...
2017.09.02, 11:27:26: Validation Accuracy = 0.962
2017.09.02, 11:27:51: EPOCH 218 ...
2017.09.02, 11:27:51: Validation Accuracy = 0.963
2017.09.02, 11:28:15: EPOCH 219 ...
2017.09.02, 11:28:15: Validation Accuracy = 0.962
2017.09.02, 11:28:39: EPOCH 220 ...
2017.09.02, 11:28:39: Validation Accuracy = 0.962
2017.09.02, 11:29:03: EPOCH 221 ...
2017.09.02, 11:29:03: Validation Accuracy = 0.962
2017.09.02, 11:29:28: EPOCH 222 ...
2017.09.02, 11:29:28: Validation Accuracy = 0.962
2017.09.02, 11:29:52: EPOCH 223 ...
2017.09.02, 11:29:52: Validation Accuracy = 0.962
2017.09.02, 11:30:16: EPOCH 224 ...
```

```
2017.09.02, 11:30:16: Validation Accuracy = 0.962
2017.09.02, 11:30:41: EPOCH 225 ...
2017.09.02, 11:30:41: Validation Accuracy = 0.963
2017.09.02, 11:31:05: EPOCH 226 ...
2017.09.02, 11:31:05: Validation Accuracy = 0.962
2017.09.02, 11:31:29: EPOCH 227 ...
2017.09.02, 11:31:29: Validation Accuracy = 0.961
2017.09.02, 11:31:53: EPOCH 228 ...
2017.09.02, 11:31:53: Validation Accuracy = 0.962
2017.09.02, 11:32:18: EPOCH 229 ...
2017.09.02, 11:32:18: Validation Accuracy = 0.962
2017.09.02, 11:32:42: EPOCH 230 ...
2017.09.02, 11:32:42: Validation Accuracy = 0.961
2017.09.02, 11:33:06: EPOCH 231 ...
2017.09.02, 11:33:06: Validation Accuracy = 0.963
2017.09.02, 11:33:31: EPOCH 232 ...
2017.09.02, 11:33:31: Validation Accuracy = 0.963
2017.09.02, 11:33:55: EPOCH 233 ...
2017.09.02, 11:33:55: Validation Accuracy = 0.961
2017.09.02, 11:34:19: EPOCH 234 ...
2017.09.02, 11:34:19: Validation Accuracy = 0.962
2017.09.02, 11:34:43: EPOCH 235 ...
2017.09.02, 11:34:43: Validation Accuracy = 0.962
2017.09.02, 11:35:08: EPOCH 236 ...
2017.09.02, 11:35:08: Validation Accuracy = 0.960
2017.09.02, 11:35:32: EPOCH 237 ...
2017.09.02, 11:35:32: Validation Accuracy = 0.962
2017.09.02, 11:35:56: EPOCH 238 ...
2017.09.02, 11:35:56: Validation Accuracy = 0.963
2017.09.02, 11:36:20: EPOCH 239 ...
2017.09.02, 11:36:20: Validation Accuracy = 0.960
2017.09.02, 11:36:45: EPOCH 240 ...
2017.09.02, 11:36:45: Validation Accuracy = 0.961
2017.09.02, 11:37:09: EPOCH 241 ...
2017.09.02, 11:37:09: Validation Accuracy = 0.962
2017.09.02, 11:37:33: EPOCH 242 ...
2017.09.02, 11:37:33: Validation Accuracy = 0.963
2017.09.02, 11:37:57: EPOCH 243 ...
2017.09.02, 11:37:57: Validation Accuracy = 0.962
2017.09.02, 11:38:22: EPOCH 244 ...
2017.09.02, 11:38:22: Validation Accuracy = 0.962
2017.09.02, 11:38:46: EPOCH 245 ...
2017.09.02, 11:38:46: Validation Accuracy = 0.962
2017.09.02, 11:39:10: EPOCH 246 ...
2017.09.02, 11:39:10: Validation Accuracy = 0.962
2017.09.02, 11:39:34: EPOCH 247 ...
2017.09.02, 11:39:34: Validation Accuracy = 0.962
2017.09.02, 11:39:59: EPOCH 248 ...
2017.09.02, 11:39:59: Validation Accuracy = 0.961
2017.09.02, 11:40:23: EPOCH 249 ...
2017.09.02, 11:40:23: Validation Accuracy = 0.963
2017.09.02, 11:40:47: EPOCH 250 ...
2017.09.02, 11:40:47: Validation Accuracy = 0.962
2017.09.02, 11:41:12: EPOCH 251 ...
2017.09.02, 11:41:12: Validation Accuracy = 0.961
2017.09.02, 11:41:36: EPOCH 252 ...
2017.09.02, 11:41:36: Validation Accuracy = 0.962
2017.09.02, 11:42:00: EPOCH 253 ...
2017.09.02, 11:42:00: Validation Accuracy = 0.961
2017.09.02, 11:42:24: EPOCH 254 ...
2017.09.02, 11:42:24: Validation Accuracy = 0.963
```

```
2017.09.02, 11:42:49: EPOCH 255 ...
2017.09.02, 11:42:49: Validation Accuracy = 0.962
2017.09.02, 11:43:13: EPOCH 256 ...
2017.09.02, 11:43:13: Validation Accuracy = 0.962
2017.09.02, 11:43:37: EPOCH 257 ...
2017.09.02, 11:43:37: Validation Accuracy = 0.962
2017.09.02, 11:44:01: EPOCH 258 ...
2017.09.02, 11:44:01: Validation Accuracy = 0.962
2017.09.02, 11:44:26: EPOCH 259 ...
2017.09.02, 11:44:26: Validation Accuracy = 0.963
2017.09.02, 11:44:50: EPOCH 260 ...
2017.09.02, 11:44:50: Validation Accuracy = 0.962
2017.09.02, 11:45:14: EPOCH 261 ...
2017.09.02, 11:45:14: Validation Accuracy = 0.962
2017.09.02, 11:45:39: EPOCH 262 ...
2017.09.02, 11:45:39: Validation Accuracy = 0.962
2017.09.02, 11:46:03: EPOCH 263 ...
2017.09.02, 11:46:03: Validation Accuracy = 0.962
2017.09.02, 11:46:27: EPOCH 264 ...
2017.09.02, 11:46:27: Validation Accuracy = 0.962
2017.09.02, 11:46:51: EPOCH 265 ...
2017.09.02, 11:46:51: Validation Accuracy = 0.961
2017.09.02, 11:47:16: EPOCH 266 ...
2017.09.02, 11:47:16: Validation Accuracy = 0.963
2017.09.02, 11:47:40: EPOCH 267 ...
2017.09.02, 11:47:40: Validation Accuracy = 0.962
2017.09.02, 11:48:04: EPOCH 268 ...
2017.09.02, 11:48:04: Validation Accuracy = 0.963
2017.09.02, 11:48:28: EPOCH 269 ...
2017.09.02, 11:48:28: Validation Accuracy = 0.962
2017.09.02, 11:48:53: EPOCH 270 ...
2017.09.02, 11:48:53: Validation Accuracy = 0.961
2017.09.02, 11:49:17: EPOCH 271 ...
2017.09.02, 11:49:17: Validation Accuracy = 0.962
2017.09.02, 11:49:41: EPOCH 272 ...
2017.09.02, 11:49:41: Validation Accuracy = 0.963
2017.09.02, 11:50:06: EPOCH 273 ...
2017.09.02, 11:50:06: Validation Accuracy = 0.963
2017.09.02, 11:50:30: EPOCH 274 ...
2017.09.02, 11:50:30: Validation Accuracy = 0.962
2017.09.02, 11:50:54: EPOCH 275 ...
2017.09.02, 11:50:54: Validation Accuracy = 0.962
2017.09.02, 11:51:18: EPOCH 276 ...
2017.09.02, 11:51:18: Validation Accuracy = 0.962
2017.09.02, 11:51:43: EPOCH 277 ...
2017.09.02, 11:51:43: Validation Accuracy = 0.962
2017.09.02, 11:52:07: EPOCH 278 ...
2017.09.02, 11:52:07: Validation Accuracy = 0.962
2017.09.02, 11:52:31: EPOCH 279 ...
2017.09.02, 11:52:31: Validation Accuracy = 0.962
2017.09.02, 11:52:56: EPOCH 280 ...
2017.09.02, 11:52:56: Validation Accuracy = 0.963
2017.09.02, 11:53:20: EPOCH 281 ...
2017.09.02, 11:53:20: Validation Accuracy = 0.963
2017.09.02, 11:53:44: EPOCH 282 ...
2017.09.02, 11:53:44: Validation Accuracy = 0.963
2017.09.02, 11:54:08: EPOCH 283 ...
2017.09.02, 11:54:08: Validation Accuracy = 0.963
2017.09.02, 11:54:33: EPOCH 284 ...
2017.09.02, 11:54:33: Validation Accuracy = 0.962
2017.09.02, 11:54:57: EPOCH 285 ...
```

```
2017.09.02, 11:54:57: Validation Accuracy = 0.963
2017.09.02, 11:55:21: EPOCH 286 ...
2017.09.02, 11:55:21: Validation Accuracy = 0.963
2017.09.02, 11:55:45: EPOCH 287 ...
2017.09.02, 11:55:45: Validation Accuracy = 0.962
2017.09.02, 11:56:10: EPOCH 288 ...
2017.09.02, 11:56:10: Validation Accuracy = 0.962
2017.09.02, 11:56:34: EPOCH 289 ...
2017.09.02, 11:56:34: Validation Accuracy = 0.963
```

```
2017.09.02, 11:56:58: EPOCH 290 ...
2017.09.02, 11:56:58: Validation Accuracy = 0.962
2017.09.02, 11:57:23: EPOCH 291 ...
2017.09.02, 11:57:23: Validation Accuracy = 0.962
2017.09.02, 11:57:47: EPOCH 292 ...
2017.09.02, 11:57:47: Validation Accuracy = 0.962
2017.09.02, 11:58:11: EPOCH 293 ...
2017.09.02, 11:58:11: Validation Accuracy = 0.963
2017.09.02, 11:58:35: EPOCH 294 ...
2017.09.02, 11:58:35: Validation Accuracy = 0.963
2017.09.02, 11:59:00: EPOCH 295 ...
2017.09.02, 11:59:00: Validation Accuracy = 0.962
2017.09.02, 11:59:24: EPOCH 296 ...
2017.09.02, 11:59:24: Validation Accuracy = 0.962
2017.09.02, 11:59:48: EPOCH 297 ...
2017.09.02, 11:59:48: Validation Accuracy = 0.962
2017.09.02, 12:00:12: EPOCH 298 ...
2017.09.02, 12:00:12: Validation Accuracy = 0.962
2017.09.02, 12:00:37: EPOCH 299 ...
2017.09.02, 12:00:37: Validation Accuracy = 0.963
2017.09.02, 12:01:01: EPOCH 300 ...
2017.09.02, 12:01:01: Validation Accuracy = 0.962
2017.09.02, 12:01:25: EPOCH 301 ...
2017.09.02, 12:01:25: Validation Accuracy = 0.962
2017.09.02, 12:01:50: EPOCH 302 ...
2017.09.02, 12:01:50: Validation Accuracy = 0.962
2017.09.02, 12:02:14: EPOCH 303 ...
2017.09.02, 12:02:14: Validation Accuracy = 0.963
2017.09.02, 12:02:38: EPOCH 304 ...
2017.09.02, 12:02:38: Validation Accuracy = 0.964
2017.09.02, 12:03:03: EPOCH 305 ...
2017.09.02, 12:03:03: Validation Accuracy = 0.963
2017.09.02, 12:03:27: EPOCH 306 ...
2017.09.02, 12:03:27: Validation Accuracy = 0.962
2017.09.02, 12:03:51: EPOCH 307 ...
2017.09.02, 12:03:51: Validation Accuracy = 0.963
2017.09.02, 12:04:15: EPOCH 308 ...
2017.09.02, 12:04:15: Validation Accuracy = 0.963
2017.09.02, 12:04:39: EPOCH 309 ...
2017.09.02, 12:04:39: Validation Accuracy = 0.962
2017.09.02, 12:05:04: EPOCH 310 ...
2017.09.02, 12:05:04: Validation Accuracy = 0.963
2017.09.02, 12:05:28: EPOCH 311 ...
2017.09.02, 12:05:28: Validation Accuracy = 0.961
2017.09.02, 12:05:52: EPOCH 312 ...
2017.09.02, 12:05:52: Validation Accuracy = 0.962
2017.09.02, 12:06:16: EPOCH 313 ...
2017.09.02, 12:06:16: Validation Accuracy = 0.963
2017.09.02, 12:06:41: EPOCH 314 ...
2017.09.02, 12:06:41: Validation Accuracy = 0.963
2017.09.02, 12:07:05: EPOCH 315 ...
2017.09.02, 12:07:05: Validation Accuracy = 0.962
2017.09.02, 12:07:29: EPOCH 316 ...
2017.09.02, 12:07:29: Validation Accuracy = 0.963
2017.09.02, 12:07:53: EPOCH 317 ...
2017.09.02, 12:07:53: Validation Accuracy = 0.963
2017.09.02, 12:08:18: EPOCH 318 ...
2017.09.02, 12:08:18: Validation Accuracy = 0.963
2017.09.02, 12:08:42: EPOCH 319 ...
2017.09.02, 12:08:42: Validation Accuracy = 0.963
2017.09.02, 12:09:06: EPOCH 320 ...
```

```
2017.09.02, 12:09:06: Validation Accuracy = 0.964
2017.09.02, 12:09:31: EPOCH 321 ...
2017.09.02, 12:09:31: Validation Accuracy = 0.962
2017.09.02, 12:09:55: EPOCH 322 ...
2017.09.02, 12:09:55: Validation Accuracy = 0.962
2017.09.02, 12:10:19: EPOCH 323 ...
2017.09.02, 12:10:19: Validation Accuracy = 0.963
2017.09.02, 12:10:43: EPOCH 324 ...
2017.09.02, 12:10:43: Validation Accuracy = 0.963
2017.09.02, 12:11:08: EPOCH 325 ...
2017.09.02, 12:11:08: Validation Accuracy = 0.964
2017.09.02, 12:11:32: EPOCH 326 ...
2017.09.02, 12:11:32: Validation Accuracy = 0.963
2017.09.02, 12:11:56: EPOCH 327 ...
2017.09.02, 12:11:56: Validation Accuracy = 0.963
2017.09.02, 12:12:20: EPOCH 328 ...
2017.09.02, 12:12:20: Validation Accuracy = 0.962
2017.09.02, 12:12:45: EPOCH 329 ...
2017.09.02, 12:12:45: Validation Accuracy = 0.963
2017.09.02, 12:13:09: EPOCH 330 ...
2017.09.02, 12:13:09: Validation Accuracy = 0.964
2017.09.02, 12:13:34: EPOCH 331 ...
2017.09.02, 12:13:34: Validation Accuracy = 0.963
2017.09.02, 12:13:58: EPOCH 332 ...
2017.09.02, 12:13:58: Validation Accuracy = 0.964
2017.09.02, 12:14:22: EPOCH 333 ...
2017.09.02, 12:14:22: Validation Accuracy = 0.963
2017.09.02, 12:14:46: EPOCH 334 ...
2017.09.02, 12:14:46: Validation Accuracy = 0.963
2017.09.02, 12:15:11: EPOCH 335 ...
2017.09.02, 12:15:11: Validation Accuracy = 0.963
2017.09.02, 12:15:35: EPOCH 336 ...
2017.09.02, 12:15:35: Validation Accuracy = 0.963
2017.09.02, 12:15:59: EPOCH 337 ...
2017.09.02, 12:15:59: Validation Accuracy = 0.963
2017.09.02, 12:16:23: EPOCH 338 ...
2017.09.02, 12:16:23: Validation Accuracy = 0.964
2017.09.02, 12:16:48: EPOCH 339 ...
2017.09.02, 12:16:48: Validation Accuracy = 0.963
2017.09.02, 12:17:12: EPOCH 340 ...
2017.09.02, 12:17:12: Validation Accuracy = 0.963
2017.09.02, 12:17:36: EPOCH 341 ...
2017.09.02, 12:17:36: Validation Accuracy = 0.962
2017.09.02, 12:18:00: EPOCH 342 ...
2017.09.02, 12:18:00: Validation Accuracy = 0.964
2017.09.02, 12:18:25: EPOCH 343 ...
2017.09.02, 12:18:25: Validation Accuracy = 0.963
2017.09.02, 12:18:49: EPOCH 344 ...
2017.09.02, 12:18:49: Validation Accuracy = 0.964
2017.09.02, 12:19:13: EPOCH 345 ...
2017.09.02, 12:19:13: Validation Accuracy = 0.963
2017.09.02, 12:19:37: EPOCH 346 ...
2017.09.02, 12:19:37: Validation Accuracy = 0.963
2017.09.02, 12:20:02: EPOCH 347 ...
2017.09.02, 12:20:02: Validation Accuracy = 0.963
2017.09.02, 12:20:26: EPOCH 348 ...
2017.09.02, 12:20:26: Validation Accuracy = 0.963
2017.09.02, 12:20:50: EPOCH 349 ...
2017.09.02, 12:20:50: Validation Accuracy = 0.963
2017.09.02, 12:21:15: EPOCH 350 ...
2017.09.02, 12:21:15: Validation Accuracy = 0.963
```

```
2017.09.02, 12:21:39: EPOCH 351 ...
2017.09.02, 12:21:39: Validation Accuracy = 0.963
2017.09.02, 12:22:03: EPOCH 352 ...
2017.09.02, 12:22:03: Validation Accuracy = 0.964
2017.09.02, 12:22:28: EPOCH 353 ...
2017.09.02, 12:22:28: Validation Accuracy = 0.962
2017.09.02, 12:22:52: EPOCH 354 ...
2017.09.02, 12:22:52: Validation Accuracy = 0.964
2017.09.02, 12:23:16: EPOCH 355 ...
2017.09.02, 12:23:16: Validation Accuracy = 0.963
2017.09.02, 12:23:40: EPOCH 356 ...
2017.09.02, 12:23:40: Validation Accuracy = 0.963
2017.09.02, 12:24:05: EPOCH 357 ...
2017.09.02, 12:24:05: Validation Accuracy = 0.962
2017.09.02, 12:24:29: EPOCH 358 ...
2017.09.02, 12:24:29: Validation Accuracy = 0.963
2017.09.02, 12:24:53: EPOCH 359 ...
2017.09.02, 12:24:53: Validation Accuracy = 0.963
2017.09.02, 12:25:17: EPOCH 360 ...
2017.09.02, 12:25:17: Validation Accuracy = 0.963
2017.09.02, 12:25:42: EPOCH 361 ...
2017.09.02, 12:25:42: Validation Accuracy = 0.965
2017.09.02, 12:26:06: EPOCH 362 ...
2017.09.02, 12:26:06: Validation Accuracy = 0.963
2017.09.02, 12:26:30: EPOCH 363 ...
2017.09.02, 12:26:30: Validation Accuracy = 0.964
2017.09.02, 12:26:55: EPOCH 364 ...
2017.09.02, 12:26:55: Validation Accuracy = 0.964
2017.09.02, 12:27:19: EPOCH 365 ...
2017.09.02, 12:27:19: Validation Accuracy = 0.963
2017.09.02, 12:27:43: EPOCH 366 ...
2017.09.02, 12:27:43: Validation Accuracy = 0.963
2017.09.02, 12:28:07: EPOCH 367 ...
2017.09.02, 12:28:07: Validation Accuracy = 0.963
2017.09.02, 12:28:32: EPOCH 368 ...
2017.09.02, 12:28:32: Validation Accuracy = 0.964
2017.09.02, 12:28:56: EPOCH 369 ...
2017.09.02, 12:28:56: Validation Accuracy = 0.964
2017.09.02, 12:29:20: EPOCH 370 ...
2017.09.02, 12:29:20: Validation Accuracy = 0.963
2017.09.02, 12:29:45: EPOCH 371 ...
2017.09.02, 12:29:45: Validation Accuracy = 0.963
2017.09.02, 12:30:09: EPOCH 372 ...
2017.09.02, 12:30:09: Validation Accuracy = 0.963
2017.09.02, 12:30:33: EPOCH 373 ...
2017.09.02, 12:30:33: Validation Accuracy = 0.963
2017.09.02, 12:30:57: EPOCH 374 ...
2017.09.02, 12:30:58: Validation Accuracy = 0.963
2017.09.02, 12:31:22: EPOCH 375 ...
2017.09.02, 12:31:22: Validation Accuracy = 0.963
2017.09.02, 12:31:46: EPOCH 376 ...
2017.09.02, 12:31:46: Validation Accuracy = 0.964
2017.09.02, 12:32:10: EPOCH 377 ...
2017.09.02, 12:32:10: Validation Accuracy = 0.963
2017.09.02, 12:32:35: EPOCH 378 ...
2017.09.02, 12:32:35: Validation Accuracy = 0.963
2017.09.02, 12:32:59: EPOCH 379 ...
2017.09.02, 12:32:59: Validation Accuracy = 0.963
2017.09.02, 12:33:23: EPOCH 380 ...
2017.09.02, 12:33:23: Validation Accuracy = 0.963
2017.09.02, 12:33:48: EPOCH 381 ...
```

```
2017.09.02, 12:33:48: Validation Accuracy = 0.963
2017.09.02, 12:34:12: EPOCH 382 ...
2017.09.02, 12:34:12: Validation Accuracy = 0.964
2017.09.02, 12:34:36: EPOCH 383 ...
2017.09.02, 12:34:36: Validation Accuracy = 0.963
2017.09.02, 12:35:01: EPOCH 384 ...
2017.09.02, 12:35:01: Validation Accuracy = 0.964
2017.09.02, 12:35:25: EPOCH 385 ...
2017.09.02, 12:35:25: Validation Accuracy = 0.963
```

```
2017.09.02, 12:35:49: EPOCH 386 ...
2017.09.02, 12:35:49: Validation Accuracy = 0.964
2017.09.02, 12:36:13: EPOCH 387 ...
2017.09.02, 12:36:13: Validation Accuracy = 0.964
2017.09.02, 12:36:38: EPOCH 388 ...
2017.09.02, 12:36:38: Validation Accuracy = 0.964
2017.09.02, 12:37:02: EPOCH 389 ...
2017.09.02, 12:37:02: Validation Accuracy = 0.964
2017.09.02, 12:37:26: EPOCH 390 ...
2017.09.02, 12:37:26: Validation Accuracy = 0.963
2017.09.02, 12:37:51: EPOCH 391 ...
2017.09.02, 12:37:51: Validation Accuracy = 0.964
2017.09.02, 12:38:15: EPOCH 392 ...
2017.09.02, 12:38:15: Validation Accuracy = 0.964
2017.09.02, 12:38:39: EPOCH 393 ...
2017.09.02, 12:38:39: Validation Accuracy = 0.964
2017.09.02, 12:39:03: EPOCH 394 ...
2017.09.02, 12:39:03: Validation Accuracy = 0.965
2017.09.02, 12:39:28: EPOCH 395 ...
2017.09.02, 12:39:28: Validation Accuracy = 0.964
2017.09.02, 12:39:52: EPOCH 396 ...
2017.09.02, 12:39:52: Validation Accuracy = 0.963
2017.09.02, 12:40:16: EPOCH 397 ...
2017.09.02, 12:40:16: Validation Accuracy = 0.964
2017.09.02, 12:40:40: EPOCH 398 ...
2017.09.02, 12:40:40: Validation Accuracy = 0.964
2017.09.02, 12:41:05: EPOCH 399 ...
2017.09.02, 12:41:05: Validation Accuracy = 0.963
2017.09.02, 12:41:29: EPOCH 400 ...
2017.09.02, 12:41:29: Validation Accuracy = 0.964
2017.09.02, 12:41:54: EPOCH 401 ...
2017.09.02, 12:41:54: Validation Accuracy = 0.963
2017.09.02, 12:42:18: EPOCH 402 ...
2017.09.02, 12:42:18: Validation Accuracy = 0.964
2017.09.02, 12:42:42: EPOCH 403 ...
2017.09.02, 12:42:42: Validation Accuracy = 0.964
2017.09.02, 12:43:07: EPOCH 404 ...
2017.09.02, 12:43:07: Validation Accuracy = 0.964
2017.09.02, 12:43:31: EPOCH 405 ...
2017.09.02, 12:43:31: Validation Accuracy = 0.963
2017.09.02, 12:43:55: EPOCH 406 ...
2017.09.02, 12:43:55: Validation Accuracy = 0.965
2017.09.02, 12:44:19: EPOCH 407 ...
2017.09.02, 12:44:19: Validation Accuracy = 0.964
2017.09.02, 12:44:44: EPOCH 408 ...
2017.09.02, 12:44:44: Validation Accuracy = 0.964
2017.09.02, 12:45:08: EPOCH 409 ...
2017.09.02, 12:45:08: Validation Accuracy = 0.963
2017.09.02, 12:45:32: EPOCH 410 ...
2017.09.02, 12:45:32: Validation Accuracy = 0.963
2017.09.02, 12:45:56: EPOCH 411 ...
2017.09.02, 12:45:56: Validation Accuracy = 0.963
2017.09.02, 12:46:21: EPOCH 412 ...
2017.09.02, 12:46:21: Validation Accuracy = 0.963
2017.09.02, 12:46:45: EPOCH 413 ...
2017.09.02, 12:46:45: Validation Accuracy = 0.964
2017.09.02, 12:47:09: EPOCH 414 ...
2017.09.02, 12:47:09: Validation Accuracy = 0.965
2017.09.02, 12:47:34: EPOCH 415 ...
2017.09.02, 12:47:34: Validation Accuracy = 0.963
2017.09.02, 12:47:58: EPOCH 416 ...
```

```
2017.09.02, 12:47:58: Validation Accuracy = 0.964
2017.09.02, 12:48:22: EPOCH 417 ...
2017.09.02, 12:48:22: Validation Accuracy = 0.964
2017.09.02, 12:48:46: EPOCH 418 ...
2017.09.02, 12:48:46: Validation Accuracy = 0.963
2017.09.02, 12:49:11: EPOCH 419 ...
2017.09.02, 12:49:11: Validation Accuracy = 0.964
2017.09.02, 12:49:35: EPOCH 420 ...
2017.09.02, 12:49:35: Validation Accuracy = 0.964
2017.09.02, 12:49:59: EPOCH 421 ...
2017.09.02, 12:49:59: Validation Accuracy = 0.964
2017.09.02, 12:50:24: EPOCH 422 ...
2017.09.02, 12:50:24: Validation Accuracy = 0.964
2017.09.02, 12:50:48: EPOCH 423 ...
2017.09.02, 12:50:48: Validation Accuracy = 0.964
2017.09.02, 12:51:12: EPOCH 424 ...
2017.09.02, 12:51:12: Validation Accuracy = 0.964
2017.09.02, 12:51:36: EPOCH 425 ...
2017.09.02, 12:51:36: Validation Accuracy = 0.964
2017.09.02, 12:52:00: EPOCH 426 ...
2017.09.02, 12:52:00: Validation Accuracy = 0.964
2017.09.02, 12:52:25: EPOCH 427 ...
2017.09.02, 12:52:25: Validation Accuracy = 0.963
2017.09.02, 12:52:49: EPOCH 428 ...
2017.09.02, 12:52:49: Validation Accuracy = 0.964
2017.09.02, 12:53:13: EPOCH 429 ...
2017.09.02, 12:53:13: Validation Accuracy = 0.964
2017.09.02, 12:53:37: EPOCH 430 ...
2017.09.02, 12:53:37: Validation Accuracy = 0.964
2017.09.02, 12:54:02: EPOCH 431 ...
2017.09.02, 12:54:02: Validation Accuracy = 0.963
2017.09.02, 12:54:26: EPOCH 432 ...
2017.09.02, 12:54:26: Validation Accuracy = 0.964
2017.09.02, 12:54:50: EPOCH 433 ...
2017.09.02, 12:54:50: Validation Accuracy = 0.964
2017.09.02, 12:55:15: EPOCH 434 ...
2017.09.02, 12:55:15: Validation Accuracy = 0.963
2017.09.02, 12:55:39: EPOCH 435 ...
2017.09.02, 12:55:39: Validation Accuracy = 0.965
2017.09.02, 12:56:03: EPOCH 436 ...
2017.09.02, 12:56:03: Validation Accuracy = 0.964
2017.09.02, 12:56:27: EPOCH 437 ...
2017.09.02, 12:56:27: Validation Accuracy = 0.965
2017.09.02, 12:56:52: EPOCH 438 ...
2017.09.02, 12:56:52: Validation Accuracy = 0.965
2017.09.02, 12:57:16: EPOCH 439 ...
2017.09.02, 12:57:16: Validation Accuracy = 0.964
2017.09.02, 12:57:40: EPOCH 440 ...
2017.09.02, 12:57:40: Validation Accuracy = 0.965
2017.09.02, 12:58:05: EPOCH 441 ...
2017.09.02, 12:58:05: Validation Accuracy = 0.964
2017.09.02, 12:58:29: EPOCH 442 ...
2017.09.02, 12:58:29: Validation Accuracy = 0.964
2017.09.02, 12:58:53: EPOCH 443 ...
2017.09.02, 12:58:53: Validation Accuracy = 0.964
2017.09.02, 12:59:17: EPOCH 444 ...
2017.09.02, 12:59:17: Validation Accuracy = 0.964
2017.09.02, 12:59:42: EPOCH 445 ...
2017.09.02, 12:59:42: Validation Accuracy = 0.965
2017.09.02, 13:00:06: EPOCH 446 ...
2017.09.02, 13:00:06: Validation Accuracy = 0.963
```

```
2017.09.02, 13:00:30: EPOCH 447 ...
2017.09.02, 13:00:30: Validation Accuracy = 0.964
2017.09.02, 13:00:54: EPOCH 448 ...
2017.09.02, 13:00:54: Validation Accuracy = 0.964
2017.09.02, 13:01:19: EPOCH 449 ...
2017.09.02, 13:01:19: Validation Accuracy = 0.964
2017.09.02, 13:01:43: EPOCH 450 ...
2017.09.02, 13:01:43: Validation Accuracy = 0.964
2017.09.02, 13:02:07: EPOCH 451 ...
2017.09.02, 13:02:07: Validation Accuracy = 0.964
2017.09.02, 13:02:32: EPOCH 452 ...
2017.09.02, 13:02:32: Validation Accuracy = 0.963
2017.09.02, 13:02:56: EPOCH 453 ...
2017.09.02, 13:02:56: Validation Accuracy = 0.964
2017.09.02, 13:03:20: EPOCH 454 ...
2017.09.02, 13:03:20: Validation Accuracy = 0.964
2017.09.02, 13:03:45: EPOCH 455 ...
2017.09.02, 13:03:45: Validation Accuracy = 0.964
2017.09.02, 13:04:09: EPOCH 456 ...
2017.09.02, 13:04:09: Validation Accuracy = 0.964
2017.09.02, 13:04:33: EPOCH 457 ...
2017.09.02, 13:04:33: Validation Accuracy = 0.964
2017.09.02, 13:04:57: EPOCH 458 ...
2017.09.02, 13:04:57: Validation Accuracy = 0.964
2017.09.02, 13:05:22: EPOCH 459 ...
2017.09.02, 13:05:22: Validation Accuracy = 0.964
2017.09.02, 13:05:46: EPOCH 460 ...
2017.09.02, 13:05:46: Validation Accuracy = 0.965
2017.09.02, 13:06:10: EPOCH 461 ...
2017.09.02, 13:06:10: Validation Accuracy = 0.964
2017.09.02, 13:06:34: EPOCH 462 ...
2017.09.02, 13:06:34: Validation Accuracy = 0.964
2017.09.02, 13:06:59: EPOCH 463 ...
2017.09.02, 13:06:59: Validation Accuracy = 0.964
2017.09.02, 13:07:23: EPOCH 464 ...
2017.09.02, 13:07:23: Validation Accuracy = 0.964
2017.09.02, 13:07:47: EPOCH 465 ...
2017.09.02, 13:07:47: Validation Accuracy = 0.964
2017.09.02, 13:08:12: EPOCH 466 ...
2017.09.02, 13:08:12: Validation Accuracy = 0.964
2017.09.02, 13:08:36: EPOCH 467 ...
2017.09.02, 13:08:36: Validation Accuracy = 0.964
2017.09.02, 13:09:00: EPOCH 468 ...
2017.09.02, 13:09:00: Validation Accuracy = 0.965
2017.09.02, 13:09:25: EPOCH 469 ...
2017.09.02, 13:09:25: Validation Accuracy = 0.965
2017.09.02, 13:09:49: EPOCH 470 ...
2017.09.02, 13:09:49: Validation Accuracy = 0.964
2017.09.02, 13:10:13: EPOCH 471 ...
2017.09.02, 13:10:13: Validation Accuracy = 0.964
2017.09.02, 13:10:37: EPOCH 472 ...
2017.09.02, 13:10:37: Validation Accuracy = 0.964
2017.09.02, 13:11:02: EPOCH 473 ...
2017.09.02, 13:11:02: Validation Accuracy = 0.964
2017.09.02, 13:11:26: EPOCH 474 ...
2017.09.02, 13:11:26: Validation Accuracy = 0.964
2017.09.02, 13:11:50: EPOCH 475 ...
2017.09.02, 13:11:50: Validation Accuracy = 0.964
2017.09.02, 13:12:14: EPOCH 476 ...
2017.09.02, 13:12:14: Validation Accuracy = 0.964
2017.09.02, 13:12:39: EPOCH 477 ...
```

```
2017.09.02, 13:12:39: Validation Accuracy = 0.965
2017.09.02, 13:13:03: EPOCH 478 ...
2017.09.02, 13:13:03: Validation Accuracy = 0.964
2017.09.02, 13:13:27: EPOCH 479 ...
2017.09.02, 13:13:27: Validation Accuracy = 0.965
2017.09.02, 13:13:51: EPOCH 480 ...
2017.09.02, 13:13:51: Validation Accuracy = 0.964
2017.09.02, 13:14:16: EPOCH 481
2017.09.02, 13:14:40: EPOCH 482 ...
2017.09.02, 13:14:40: Validation Accuracy = 0.964
2017.09.02, 13:15:04: EPOCH 483 ...
2017.09.02, 13:15:04: Validation Accuracy = 0.963
2017.09.02, 13:15:28: EPOCH 484 ...
2017.09.02, 13:15:28: Validation Accuracy = 0.965
2017.09.02, 13:15:53: EPOCH 485 ...
2017.09.02, 13:15:53: Validation Accuracy = 0.965
2017.09.02, 13:16:17: EPOCH 486 ...
2017.09.02, 13:16:17: Validation Accuracy = 0.964
2017.09.02, 13:16:41: EPOCH 487 ...
2017.09.02, 13:16:41: Validation Accuracy = 0.963
2017.09.02, 13:17:05: EPOCH 488 ...
2017.09.02, 13:17:05: Validation Accuracy = 0.964
2017.09.02, 13:17:30: EPOCH 489 ...
2017.09.02, 13:17:30: Validation Accuracy = 0.964
2017.09.02, 13:17:54: EPOCH 490 ...
2017.09.02, 13:17:54: Validation Accuracy = 0.963
2017.09.02, 13:18:18: EPOCH 491 ...
2017.09.02, 13:18:18: Validation Accuracy = 0.964
2017.09.02, 13:18:42: EPOCH 492 ...
2017.09.02, 13:18:42: Validation Accuracy = 0.964
2017.09.02, 13:19:07: EPOCH 493 ...
2017.09.02, 13:19:07: Validation Accuracy = 0.964
2017.09.02, 13:19:31: EPOCH 494 ...
2017.09.02, 13:19:31: Validation Accuracy = 0.964
2017.09.02, 13:19:55: EPOCH 495 ...
2017.09.02, 13:19:55: Validation Accuracy = 0.965
2017.09.02, 13:20:20: EPOCH 496 ...
2017.09.02, 13:20:20: Validation Accuracy = 0.964
2017.09.02, 13:20:44: EPOCH 497 ...
2017.09.02, 13:20:44: Validation Accuracy = 0.965
2017.09.02, 13:21:08: EPOCH 498 ...
2017.09.02, 13:21:08: Validation Accuracy = 0.965
2017.09.02, 13:21:32: EPOCH 499 ...
2017.09.02, 13:21:32: Validation Accuracy = 0.963
2017.09.02, 13:21:57: EPOCH 500 ...
2017.09.02, 13:21:57: Validation Accuracy = 0.965
2017.09.02, 13:21:57: Model saved
```

Test

In [49]:

```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))

    test_accuracy = evaluate(X_test_prep, y_test)
    log("Test Accuracy = {:.3f}".format(test_accuracy))
```

```
INFO:tensorflow:Restoring parameters from ./lenet
2017.09.02, 13:21:57: Test Accuracy = 0.947
```

Step 3: Test a Model on New Images

To give yourself more insight into how your model is working, download at least five pictures of German traffic signs from the web and use your model to predict the traffic sign type.

You may find `signnames.csv` useful as it contains mappings from the class id (integer) to the actual sign name.

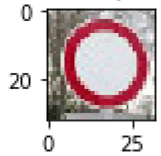
Load and Output the Images

In [50]:

```
import scipy

X_test_custom = []
y_test_custom = []

for image_name in [12, 15, 17, 35, 39]:
    im = scipy.ndimage.imread("test_images/" + str(image_name) + ".jpg")
    im_label = image_name

    X_test_custom.append(im)
    y_test_custom.append(im_label)


    plt.figure(figsize=(1,1))
    plt.title("Actual class " + str(im_label) + "(" + get_sign_name(im_label) + ")"
    plt.imshow(im.squeeze(), cmap="gray")

X_test_custom = np.array(X_test_custom)
y_test_custom = np.array(y_test_custom)
```

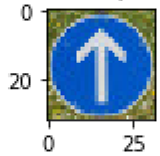
Actual class 12(Priority road)

Actual class 15(No vehicles)

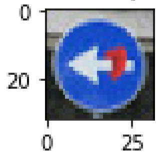


Actual class 17(No entry)

Actual class 35(Ahead only)



Actual class 39(Keep left)



In [51]:

```
X_test_custom_prep = preprocess_images(X_test_custom)
```

```
/home/q372283/anaconda3/lib/python3.5/site-packages/skimage/util/dtyp
e.py:122: UserWarning: Possible precision loss when converting from fl
oat32 to uint16
  .format(dtypeobj_in, dtypeobj_out))
```

Predict the Sign Type for Each Image

In [ ]:

```
### Run the predictions here and use the model to output the prediction for each im
### Make sure to pre-process the images with the same pre-processing pipeline used
### Feel free to use as many code cells as needed.
```

In [52]:

```
def evaluate_non_badge(X_data, y_data):

    with tf.Session() as sess:
        # Restore variables from disk.
        saver.restore(sess, "./lenet")
        print("Model restored.")

        classification = sess.run(tf.argmax(logits, 1), feed_dict={x: X_data, y: y_
        print(classification)
        return classification
```

In [53]:

```
predictions = evaluate_non_badge(X_test_custom_prep, y_test_custom)
```

```
INFO:tensorflow:Restoring parameters from ./lenet
Model restored.
[12 15 17 35 40]
```

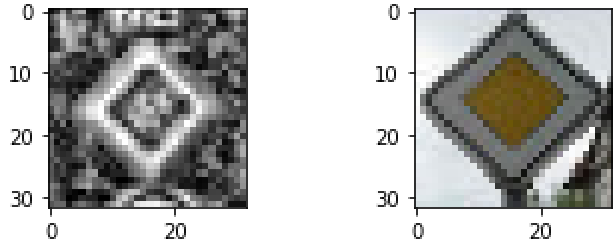
In [54]:

```
def plot_random_image_from_trainset(X, y, label_to_plot, predicted_image):
    img1 = (X[y == label_to_plot][1])
    img2 = predicted_image

    # one time
    fig = plt.figure()

    # subplots
    ax1 = fig.add_subplot(2,2,1)
    ax1.imshow(img1.squeeze(), cmap="gray")
    ax2 = fig.add_subplot(2,2,2)
    ax2.imshow(img2.squeeze(), cmap="gray")

    # show
    plt.show()
```

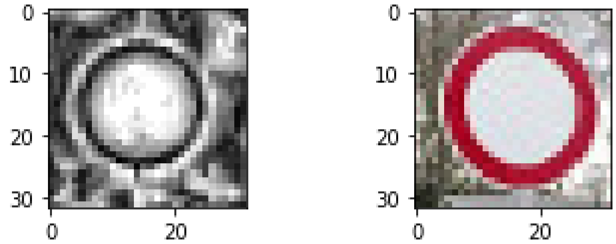
In [55]:

```
for actual, prediction in zip(y_test_custom, predictions):
    print("Class {} was predicted as {}".format(actual, prediction))
    test_image = X_test_custom[y_test_custom==actual]
    plot_random_image_from_trainset(X_train, y_train, actual, test_image)
```

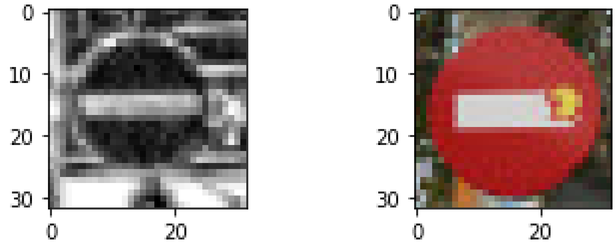
```
Class 12 was predicted as 12
```



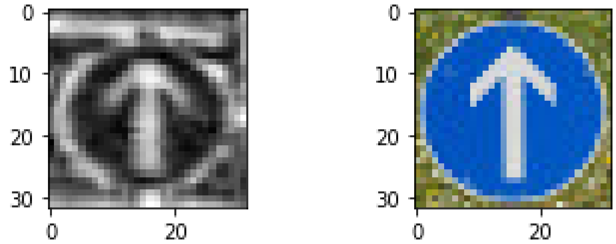
```
Class 15 was predicted as 15
```



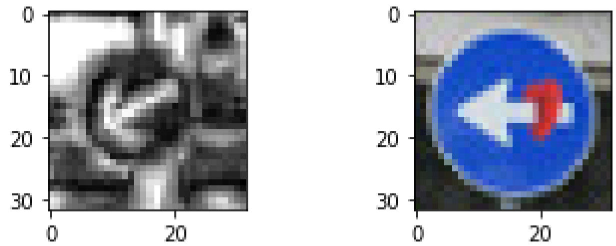
```
Class 17 was predicted as 17
```



```
Class 35 was predicted as 35
```



```
Class 39 was predicted as 40
```



Analyze Performance

In [42]:

```
### Calculate the accuracy for these 5 new images.
### For example, if the model predicted 1 out of 5 signs correctly, it's 20% accura
```

In [56]:

```
TP = sum([ x==y for (x,y) in zip(y_test_custom, predictions)])
N = len(y_test_custom)
accuracy = TP / N
```

In [57]:

```
print("accuracy of custom signs is {}%".format(round(accuracy*1000)/10))
```

```
accuracy of custom signs is 80.0%
```

Output Top 5 Softmax Probabilities For Each Image Found on the Web

For each of the new images, print out the model's softmax probabilities to show the **certainty** of the model's predictions (limit the output to the top 5 probabilities for each image). [`tf.nn.top_k`](https://www.tensorflow.org/versions/r0.12/api_docs/python/nn.html#top_k) (https://www.tensorflow.org/versions/r0.12/api_docs/python/nn.html#top_k) could prove helpful here.

The example below demonstrates how tf.nn.top_k can be used to find the top k predictions for each image.

`tf.nn.top_k` will return the values and indices (class ids) of the top k predictions. So if k=3, for each sign, it'll return the 3 largest probabilities (out of a possible 43) and the correspoding class ids.

Take this numpy array as an example. The values in the array represent predictions. The array contains softmax probabilities for five candidate images with six possible classes. `tf.nn.top_k` is used to choose the three classes with the highest probability:

```
# (5, 6) array
a = np.array([[ 0.24879643,  0.07032244,  0.12641572,  0.34763842,  0.07893
497,
         0.12789202],
       [ 0.28086119,  0.27569815,  0.08594638,  0.0178669 ,  0.18063401,
         0.15899337],
       [ 0.26076848,  0.23664738,  0.08020603,  0.07001922,  0.1134371 ,
         0.23892179],
       [ 0.11943333,  0.29198961,  0.02605103,  0.26234032,  0.1351348 ,
         0.16505091],
       [ 0.09561176,  0.34396535,  0.0643941 ,  0.16240774,  0.24206137,
         0.09155967]])
```

Running it through `sess.run(tf.nn.top_k(tf.constant(a), k=3))` produces:

```
TopKV2(values=array([[ 0.34763842,  0.24879643,  0.12789202],
       [ 0.28086119,  0.27569815,  0.18063401],
       [ 0.26076848,  0.23892179,  0.23664738],
       [ 0.29198961,  0.26234032,  0.16505091],
       [ 0.34396535,  0.24206137,  0.16240774]]), indices=array([[3, 0, 5],
       [0, 1, 4],
       [0, 5, 1],
       [1, 3, 5],
       [1, 4, 3]], dtype=int32))
```

Looking just at the first row we get `[ 0.34763842, 0.24879643, 0.12789202]`, you can confirm these are the 3 largest probabilities in a. You'll also notice `[3, 0, 5]` are the corresponding indices.

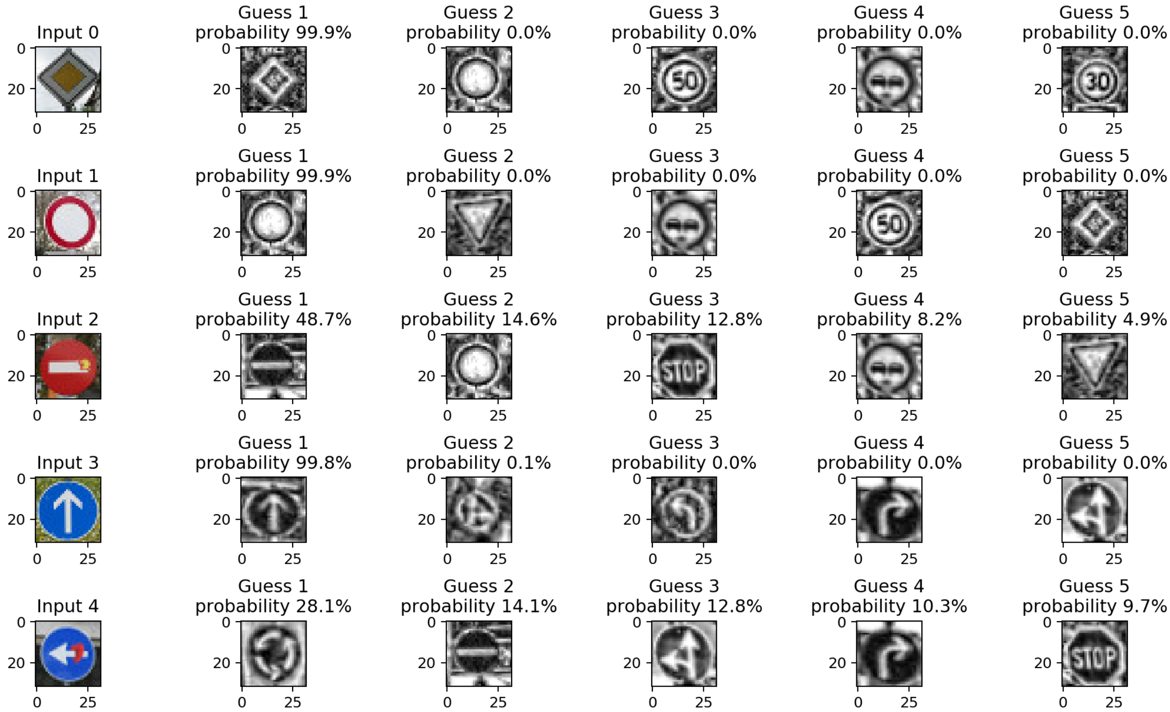
In [58]:

```
def evaluate_non_badge_softmax(X_data, y_data):

    with tf.Session() as sess:
        # Restore variables from disk.
        saver.restore(sess, "./lenet")
        print("Model restored.")

        softmax_logits = tf.nn.softmax(logits)
        top_k = tf.nn.top_k(softmax_logits, k=5)

        #In case it doesnt work uncomment following line
        # my_softmax_logits = sess.run(softmax_logits, feed_dict={x: X_data, y: y_d
        my_top_k = sess.run(top_k, feed_dict={x: X_data, keep_prob: 1.0})

        return my_top_k
```

In [59]:

```
%matplotlib inline
my_top_k = evaluate_non_badge_softmax(X_test_custom_prep, y_test_custom)


# one time
fig = plt.figure(figsize=(15.0, 8.0), dpi=180)
subplot_grid_n_col = 6 # input + 5 * highest probabilities
subplot_grid_n_row = 5 # 5 sample images
#fig, axs = plt.subplots(len(X_test_custom_prep),4, figsize=(12, 14))
fig.subplots_adjust(hspace = 1.2, wspace=0)
#axs = axs.ravel()

for i, image in enumerate(X_test_custom_prep):
    input_image = X_test_custom[i]
    ax_input = fig.add_subplot(subplot_grid_n_row,subplot_grid_n_col,i * subplot_gr
    ax_input.imshow(input_image.squeeze(), cmap="gray")
    ax_input.set_title("Input {}".format(i))

    # plot top 5 predictions
    for j, (top_pred_label, top_pred_proba) in enumerate(zip(my_top_k[1][i], my_top
        pred_image = (X_train[y_train == top_pred_label][1])

        ax_pred = fig.add_subplot(subplot_grid_n_row,subplot_grid_n_col, (i * subpl
        ax_pred.imshow(pred_image.squeeze(), cmap="gray")
        ax_pred.set_title("Guess {0}\nprobability {1:.1f}%".format(j+1, top_pred_pr

plt.show()
```

```
INFO:tensorflow:Restoring parameters from ./lenet
Model restored.
```



Project Writeup

Once you have completed the code implementation, document your results in a project writeup using this template (https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project/blob/master/writeup_template.md) as a guide. The writeup can be in a markdown or pdf file.

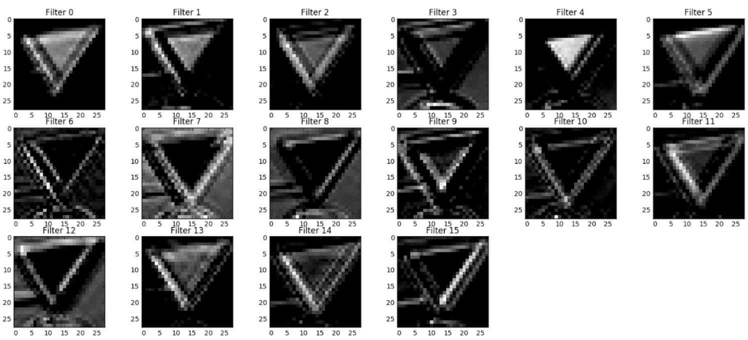
> **Note**: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

Step 4 (Optional): Visualize the Neural Network's State with Test Images

This Section is not required to complete but acts as an additional excersise for understaning the output of a neural network's weights. While neural networks can be a great learning device they are often referred to as a black box. We can understand what the weights of a neural network look like better by plotting their feature maps. After successfully training your neural network you can see what it's feature maps look like by plotting the output of the network's weight layers in response to a test stimuli image. From these plotted feature maps, it's possible to see what characteristics of an image the network finds interesting. For a sign, maybe the inner network feature maps react with high activation to the sign's boundary outline or to the contrast in the sign's painted symbol.

Provided for you below is the function code that allows you to get the visualization output of any tensorflow weight layer you want. The inputs to the function should be a stimuli image, one used during training or a new one you provided, and then the tensorflow variable name that represents the layer's state during the training process, for instance if you wanted to see what the LeNet lab's (https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/601ae704-1035-4287-8b11-e2c2716217ad/concepts/d4aca031-508f-4e0b-b493-e7b706120f81) feature maps looked like for it's second convolutional layer you could enter conv2 as the tf_activation variable.

For an example of what feature map outputs look like, check out NVIDIA's results in their paper End-to-End Deep Learning for Self-Driving Cars (https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/) in the section Visualization of internal CNN State. NVIDIA was able to show that their network's inner weights had high activations to road boundary lines by comparing feature maps from an image with a clear path to one without. Try experimenting with a similar test to show that your trained network's weights are looking for interesting features, whether it's looking at differences in feature maps from images with or without a sign, or even what feature maps look like in a trained network vs a completely untrained one on the same sign image.



Your output should look something like this (above)

In [ ]:

```
### Visualize your network's feature maps here.
### Feel free to use as many code cells as needed.

# image_input: the test image being fed into the network to produce the feature map
# tf_activation: should be a tf variable name used during your training procedure t
# activation_min/max: can be used to view the activation contrast in more detail, b
# plt_num: used to plot out multiple different weight feature map sets on the same 

def outputFeatureMap(image_input, tf_activation, activation_min=-1, activation_max=
    # Here make sure to preprocess your image_input in a way your network expects
    # with size, normalization, ect if needed
    # image_input =
    # Note: x should be the same name as your network's tensorflow data placeholder
    # If you get an error tf_activation is not defined it may be having trouble acc
    activation = tf_activation.eval(session=sess,feed_dict={x : image_input})
    featuremaps = activation.shape[3]
    plt.figure(plt_num, figsize=(15,15))
    for featuremap in range(featuremaps):
        plt.subplot(6,8, featuremap+1) # sets the number of feature maps to show on
        plt.title('FeatureMap ' + str(featuremap)) # displays the feature map numbe
        if activation_min != -1 & activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmin
        elif activation_max != -1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmax
        elif activation_min !=-1:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmin
        else:
            plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", cmap
```

What to do to improve, TODO

- experiment different network architectures
- changes dimensions of LeNet
- Tune Hyperparameters
- Augment data by rotate, color, etc.

In [ ]:

```
```