```
import numpy as np
import pandas as pd
import torch
import tensorflow as tf
import pickle
import random
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
import itertools
import seaborn as sns
from sklearn import metrics
from time import time
from torch.utils.data import DataLoader, TensorDataset
from torchvision import transforms
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remou
```

```
# size of batch --> bigger --> less time and less accuracy but stronger to outliers
#               --> smaller --> more time and more accuracy but outliers can damage model
batch_size = 8
# batch for test
test_batch_size = 1
# number of epochs
epochs = 30
# learning_rate --> too small --> very long learning
#               --> too big --> we can reach some point where our loss will start to increase and then become INF
learning_rate = 0.001
```

```
#momentum value
momentum = 0.5
# random seeds
random_seed = 42
random.seed(random_seed)
#interval for messages
logging_interval = 600
# cuda usage
no_cuda = False
use_cuda = not no_cuda and torch.cuda.is_available()

torch.manual_seed(random_seed)
device = torch.device("cuda" if use_cuda else "cpu")


# Function for loading dataset from 'pickle' file
def load_data(file):
    # Open 'pickle' file
    with open(file, 'rb') as f:
        d = pickle.load(f, encoding='latin1')
        """
        Data is a dictionary with four keys:
            'features' - is a 4D array with raw pixel data of the traffic sign images,
                         (number of examples, width, height, channels).
            'labels'   - is a 1D array containing the label id of the traffic sign image,
                         file label_names.csv contains id -> name mappings.
            'sizes'    - is a 2D array containing arrays (width, height),
                         representing the original width and height of the image.
            'coords'   - is a 2D array containing arrays (x1, y1, x2, y2),
                         representing coordinates of a bounding frame around the image.
        """
        # 4D tensor, for train = (34799, 32, 32, 3)
        input_data = d['features'].astype(np.float32)
        input_data = torch.from_numpy(input_data)
        # 1D tensor, for train = (34799,)
        target = d['labels'].astype(np.int64)
        target = torch.from_numpy(target)
        # 2D tensor, for train = (34799, 2)
        #sizes = d['sizes']
```

```
        #sizes = torch.from_numpy(sizes)
        # 2D tensor, for train = (34799, 4)
        #coords = d['coords']
        #coords = torch.from_numpy(coords)

    return input_data, target


# loading and dividing data to datasets --> dataloaders
label_names = pd.read_csv('/content/drive/My Drive/ML_LAB2/label_names.csv')
X_train, Y_train = load_data('/content/drive/My Drive/ML_LAB2/train.pickle')
train_dataset = TensorDataset(X_train.view(X_train.shape[0],X_train.shape[3],X_train.shape[1],X_train.shape[2]), Y_train)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
X_valid, Y_valid = load_data('/content/drive/My Drive/ML_LAB2/valid.pickle')
valid_dataset = TensorDataset(X_valid.view(X_valid.shape[0],X_valid.shape[3], X_valid.shape[1], X_valid.shape[2]), Y_valid)
valid_loader = DataLoader(valid_dataset, batch_size=1, shuffle=True)
print(np.array(label_names))
```

```
[[0 'Speed limit (20km/h)']
 [1 'Speed limit (30km/h)']
 [2 'Speed limit (50km/h)']
 [3 'Speed limit (60km/h)']
 [4 'Speed limit (70km/h)']
 [5 'Speed limit (80km/h)']
 [6 'End of speed limit (80km/h)']
 [7 'Speed limit (100km/h)']
 [8 'Speed limit (120km/h)']
 [9 'No passing']
 [10 'No passing for vehicles over 3.5 metric tons']
 [11 'Right-of-way at the next intersection']
 [12 'Priority road']
 [13 'Yield']
 [14 'Stop']
 [15 'No vehicles']
 [16 'Vehicles over 3.5 metric tons prohibited']
 [17 'No entry']
 [18 'General caution']
 [19 'Dangerous curve to the left']
 [20 'Dangerous curve to the right']
 [21 'Double curve']
 [22 'Bumpy road']
 [23 'Slippery road']
```

```
 [24 'Road narrows on the right']
 [25 'Road work']
 [26 'Traffic signals']
 [27 'Pedestrians']
 [28 'Children crossing']
 [29 'Bicycles crossing']
 [30 'Beware of ice/snow']
 [31 'Wild animals crossing']
 [32 'End of all speed and passing limits']
 [33 'Turn right ahead']
 [34 'Turn left ahead']
 [35 'Ahead only']
 [36 'Go straight or right']
 [37 'Go straight or left']
 [38 'Keep right']
 [39 'Keep left']
 [40 'Roundabout mandatory']
 [41 'End of no passing']
 [42 'End of no passing by vehicles over 3.5 metric tons']]
```

```
# printing all sings names
print(label_names)
```

```
    ClassId                                           SignName
0         0                               Speed limit (20km/h)
1         1                               Speed limit (30km/h)
2         2                               Speed limit (50km/h)
3         3                               Speed limit (60km/h)
4         4                               Speed limit (70km/h)
5         5                               Speed limit (80km/h)
6         6                        End of speed limit (80km/h)
7         7                              Speed limit (100km/h)
8         8                              Speed limit (120km/h)
9         9                                         No passing
10       10       No passing for vehicles over 3.5 metric tons
11       11              Right-of-way at the next intersection
12       12                                      Priority road
13       13                                              Yield
14       14                                               Stop
15       15                                        No vehicles
16       16           Vehicles over 3.5 metric tons prohibited
17       17                                           No entry
```

```
18      18                                    General caution
19      19                        Dangerous curve to the left
20      20                       Dangerous curve to the right
21      21                                       Double curve
22      22                                         Bumpy road
23      23                                      Slippery road
24      24                          Road narrows on the right
25      25                                          Road work
26      26                                    Traffic signals
27      27                                        Pedestrians
28      28                                  Children crossing
29      29                                  Bicycles crossing
30      30                                 Beware of ice/snow
31      31                              Wild animals crossing
32      32                End of all speed and passing limits
33      33                                   Turn right ahead
34      34                                    Turn left ahead
35      35                                         Ahead only
36      36                               Go straight or right
37      37                                Go straight or left
38      38                                         Keep right
39      39                                          Keep left
40      40                               Roundabout mandatory
41      41                                  End of no passing
42      42  End of no passing by vehicles over 3.5 metric ...
```

```
# reshaping images for usage in plt.imshow()
all_images = X_train.view(X_train.shape[0], X_train.shape[1], X_train.shape[2], 3)
it = iter(train_loader)
images, labels = it.next()
images = images.reshape(-1,32,32,3)


# showing some pictures from dataset
num_of_images = 120
plt.figure(figsize=(10, 16))
for index in range(1, num_of_images + 1):
    plt.subplot(12, 10, index)
    plt.axis('off')
    plt.imshow(all_images[random.randint(0,all_images.shape[0] - 1)].numpy().astype(int))
```

```
X_train.shape
```

```
torch.Size([34799, 32, 32, 3])
```

```
# Layer details for the neural network
# size of input data
input_size = X_train.shape[2] * X_train.shape[1]
canals = X_train.shape[3]
# numbers of neurons on each level
hidden_sizes = [2000, 750, 250]
# number of output neurons
output_size = label_names.shape[0]
```

```
class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size):
        super(NeuralNetwork, self).__init__()
        self.conv1 = nn.Conv2d(canals, 6, (7,7))  # result is 26x26
        self.conv2 = nn.Conv2d(6, 10, (5,5))      # result is 21x21
        self.linear1 = nn.Linear(10 * 22 * 22, hidden_sizes[0])
        self.linear2 = nn.Linear(hidden_sizes[0] , hidden_sizes[1])
        self.linear3 = nn.Linear(hidden_sizes[1] , hidden_sizes[2])
        self.linear4 = nn.Linear(hidden_sizes[2], output_size)
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.shape[0],-1)
        x = F.relu(self.linear1(x))
        x = F.relu(self.linear2(x))
        x = F.relu(self.linear3(x))
```

```
        x = F.relu(self.linear3(x))
        x = F.relu(self.linear4(x))
        return F.log_softmax(x, dim=1)

print(NeuralNetwork(input_size, hidden_sizes, output_size))
```

```
NeuralNetwork(
  (conv1): Conv2d(3, 6, kernel_size=(7, 7), stride=(1, 1))
  (conv2): Conv2d(6, 10, kernel_size=(5, 5), stride=(1, 1))
  (linear1): Linear(in_features=4840, out_features=2000, bias=True)
  (linear2): Linear(in_features=2000, out_features=750, bias=True)
  (linear3): Linear(in_features=750, out_features=250, bias=True)
  (linear4): Linear(in_features=250, out_features=43, bias=True)
)
```

```
def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % logging_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))


def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    pred = []
    tar = []
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)  # getting results from data
            test_loss += F.nll_loss(output, target, reduction='sum').item() # sum up batch loss and convert to number
```

```
            pred.append(output.argmax(dim=1, keepdim=True).item()) # get the index of the max log-probability in predictions
            tar.append(target.item())
    test_loss /= len(test_loader.dataset) # diving on batch_size in test dataset (in our case it is 1)
    print('\n---------------------------\nTrain\Valid set: Average loss: {:.4f}'.format(test_loss))
    return pred, tar, test_loss



model = NeuralNetwork(input_size, hidden_sizes, output_size).to(device)
optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)
metr = []
train_time = 0
for epoch in range(1, epochs + 1):
    print('\n---------------------------\n')
    epoch_time = time()
    train(model, device, train_loader, optimizer, epoch)
    pred, tar, loss = test(model, device, valid_loader)
    train_time += time() - epoch_time
    print('Epoch time: {:.4f} \n---------------------------\n'.format(time() - epoch_time))
    metr.append((metrics.accuracy_score(tar, pred),
                metrics.precision_score(tar, pred, average= 'macro', labels=np.unique(pred)),
                metrics.recall_score(tar, pred, average= 'macro', labels=np.unique(pred)),
                metrics.f1_score(tar, pred, average= 'macro', labels=np.unique(pred)),
                loss))
    #if epoch == 10:
    # learning_rate /= 10
    if epoch % 3 == 1:
      print('\n\n______________________________________________________________________________________________\n',
            metrics.classification_report(tar, pred, target_names= label_names['SignName']),
            '\n______________________________________________________________________________________________\n\n')
print('\n\n______________________________________________________________________________________________\n',
            metrics.classification_report(tar, pred, target_names= label_names['SignName']),
            '\n______________________________________________________________________________________________\n\n')
print('\nTrain time: ', train_time)
metr = torch.Tensor(metr)
```

```
     ---------------------------
```

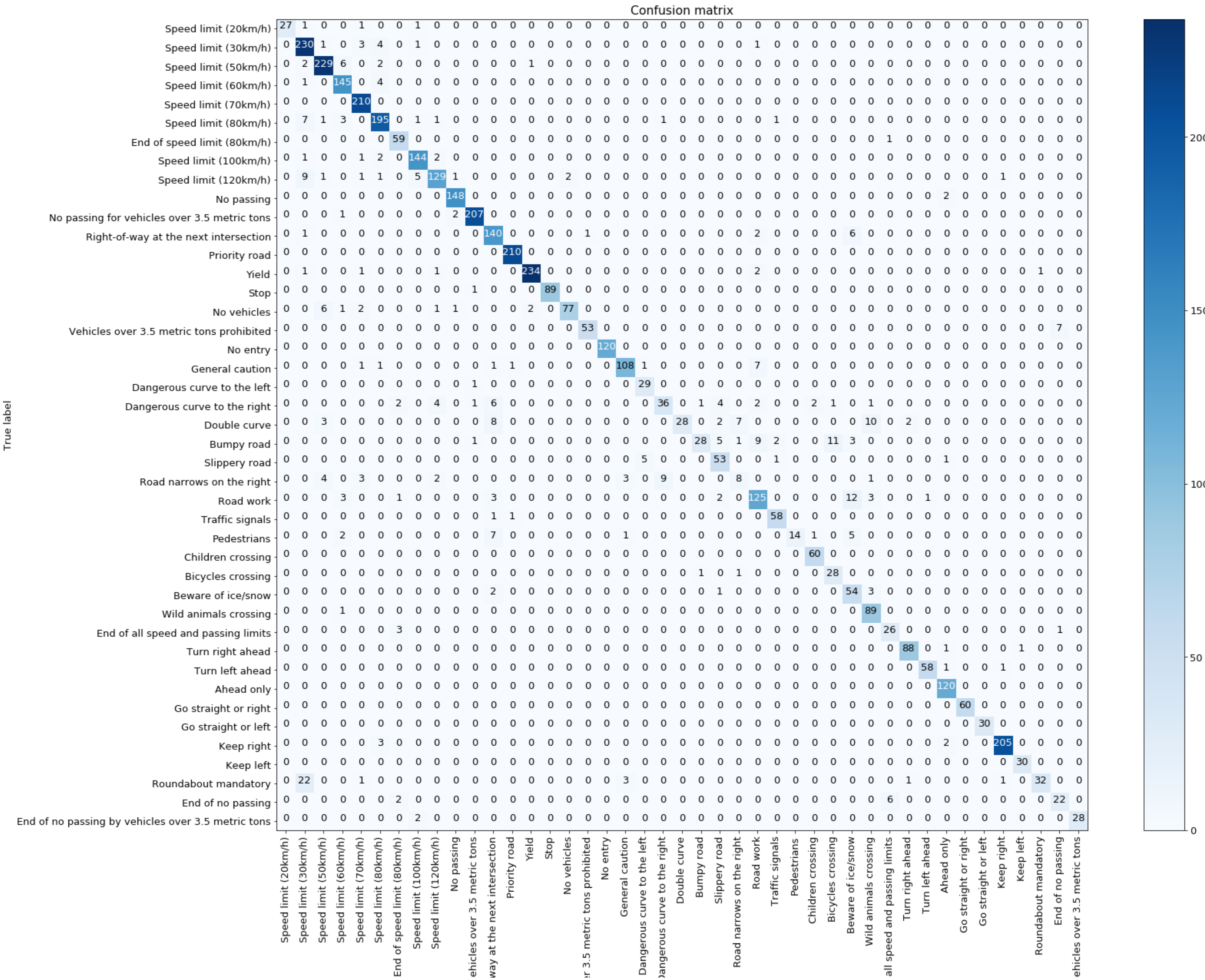
```
Train Epoch: 1 [0/34799 (0%)]	Loss: 3.957298
Train Epoch: 1 [4800/34799 (14%)]	Loss: 2.723220
Train Epoch: 1 [9600/34799 (28%)]	Loss: 1.080772
Train Epoch: 1 [14400/34799 (41%)]	Loss: 0.498705
Train Epoch: 1 [19200/34799 (55%)]	Loss: 0.841217
Train Epoch: 1 [24000/34799 (69%)]	Loss: 1.825979
Train Epoch: 1 [28800/34799 (83%)]	Loss: 0.011435
Train Epoch: 1 [33600/34799 (97%)]	Loss: 0.977474

----------------------------
Train\Valid set: Average loss: 0.8912
Epoch time: 12.5930
----------------------------
```

	precision	recall	f1-score	support
Speed limit (20km/h)	0.64	0.90	0.75	30
Speed limit (30km/h)	0.70	0.92	0.80	240
Speed limit (50km/h)	0.90	0.85	0.87	240
Speed limit (60km/h)	0.76	0.77	0.76	150
Speed limit (70km/h)	0.87	0.92	0.90	210
Speed limit (80km/h)	0.91	0.67	0.77	210
End of speed limit (80km/h)	0.92	0.90	0.91	60
Speed limit (100km/h)	0.77	0.77	0.77	150
Speed limit (120km/h)	0.64	0.83	0.73	150
No passing	0.99	0.93	0.96	150
No passing for vehicles over 3.5 metric tons	0.89	0.98	0.93	210
Right-of-way at the next intersection	0.67	0.93	0.78	150
Priority road	0.92	1.00	0.96	210
Yield	0.98	0.98	0.98	240
Stop	0.97	0.93	0.95	90
No vehicles	0.93	0.71	0.81	90
Vehicles over 3.5 metric tons prohibited	1.00	0.50	0.67	60
No entry	1.00	0.97	0.99	120
General caution	0.88	0.82	0.85	120
Dangerous curve to the left	0.50	0.57	0.53	30
Dangerous curve to the right	0.41	0.23	0.30	60
Double curve	1.00	0.20	0.33	60
Bumpy road	0.00	0.00	0.00	60
Slippery road	0.34	0.63	0.44	60

```
                Road narrows on the right       0.00      0.00      0.00        30
                                Road work       0.66      0.70      0.68       150
                          Traffic signals       0.75      0.97      0.85        60
                              Pedestrians       0.50      0.20      0.29        30
                        Children crossing       0.79      0.98      0.87        60
                        Bicycles crossing       0.74      0.77      0.75        30
                       Beware of ice/snow       0.82      0.97      0.89        60
                    Wild animals crossing       0.75      0.84      0.80        90
      End of all speed and passing limits       1.00      0.33      0.50        30
                         Turn right ahead       1.00      0.80      0.89        90
                          Turn left ahead       0.48      0.48      0.48        60
```

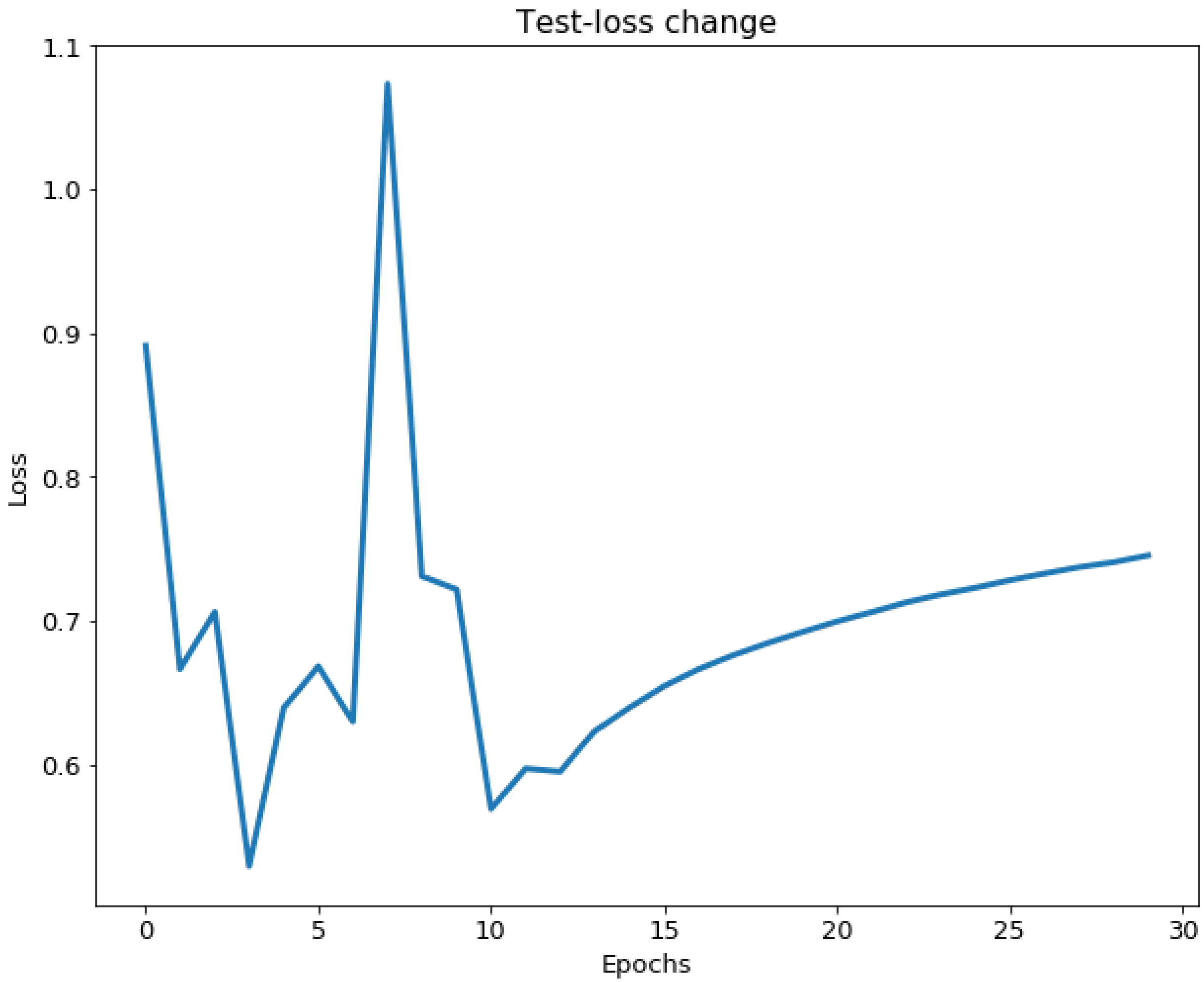
```
#resultive confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

#showing confusion matrix of model
font = {'size' : 13}
```

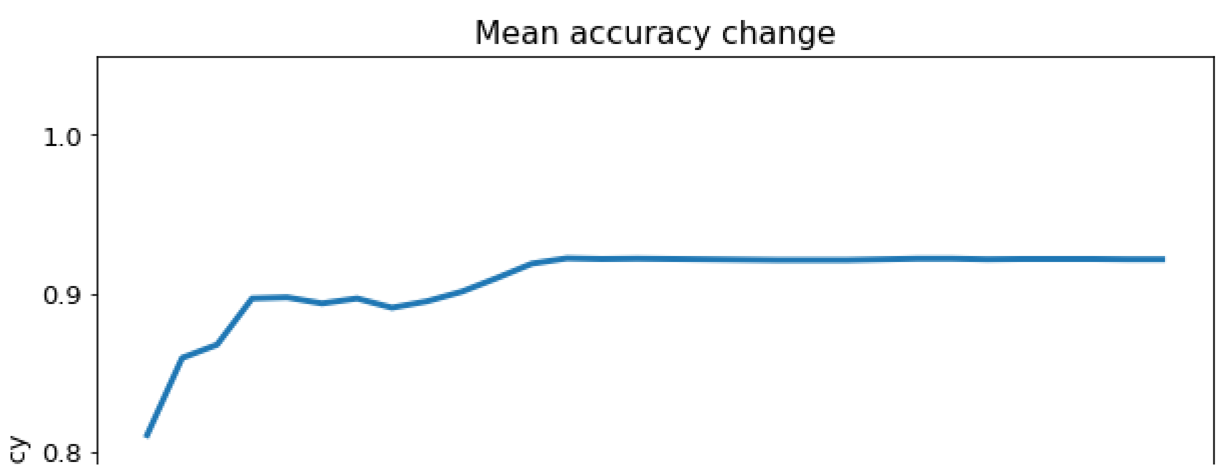
```
plt.rc('font', **font)
cnf_matrix = metrics.confusion_matrix(tar, pred)
plt.figure(figsize=(25, 20))
plot_confusion_matrix(cnf_matrix, classes= label_names['SignName'],
                      title='Confusion matrix')
plt.show()
```

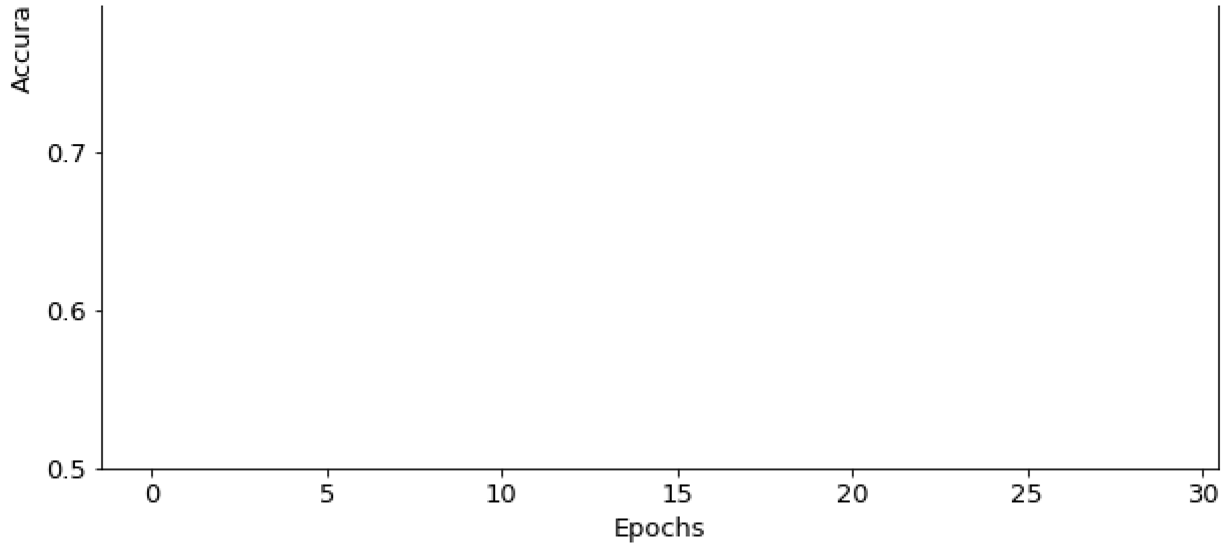

Confusion matrix
True label
Speed limit (20km/h)
Speed limit (30km/h)
Speed limit (50km/h)
Speed limit (60km/h)
Speed limit (70km/h)
Speed limit (80km/h)
End of speed limit (80km/h)
Speed limit (100km/h)
Speed limit (120km/h)
No passing
No passing for vehicles over 3.5 metric tons
Right-of-way at the next intersection
Priority road
Yield
Stop
No vehicles
Vehicles over 3.5 metric tons prohibited
No entry
General caution
Dangerous curve to the left
Dangerous curve to the right
Double curve
Bumpy road
Slippery road
Road narrows on the right
Road work
Traffic signals
Pedestrians
Children crossing
Bicycles crossing
Beware of ice/snow
Wild animals crossing
End of all speed and passing limits
Turn right ahead
Turn left ahead
Ahead only
Go straight or right
Go straight or left
Keep right
Keep left
Roundabout mandatory
End of no passing
End of no passing by vehicles over 3.5 metric tons
200
150
100
50
0

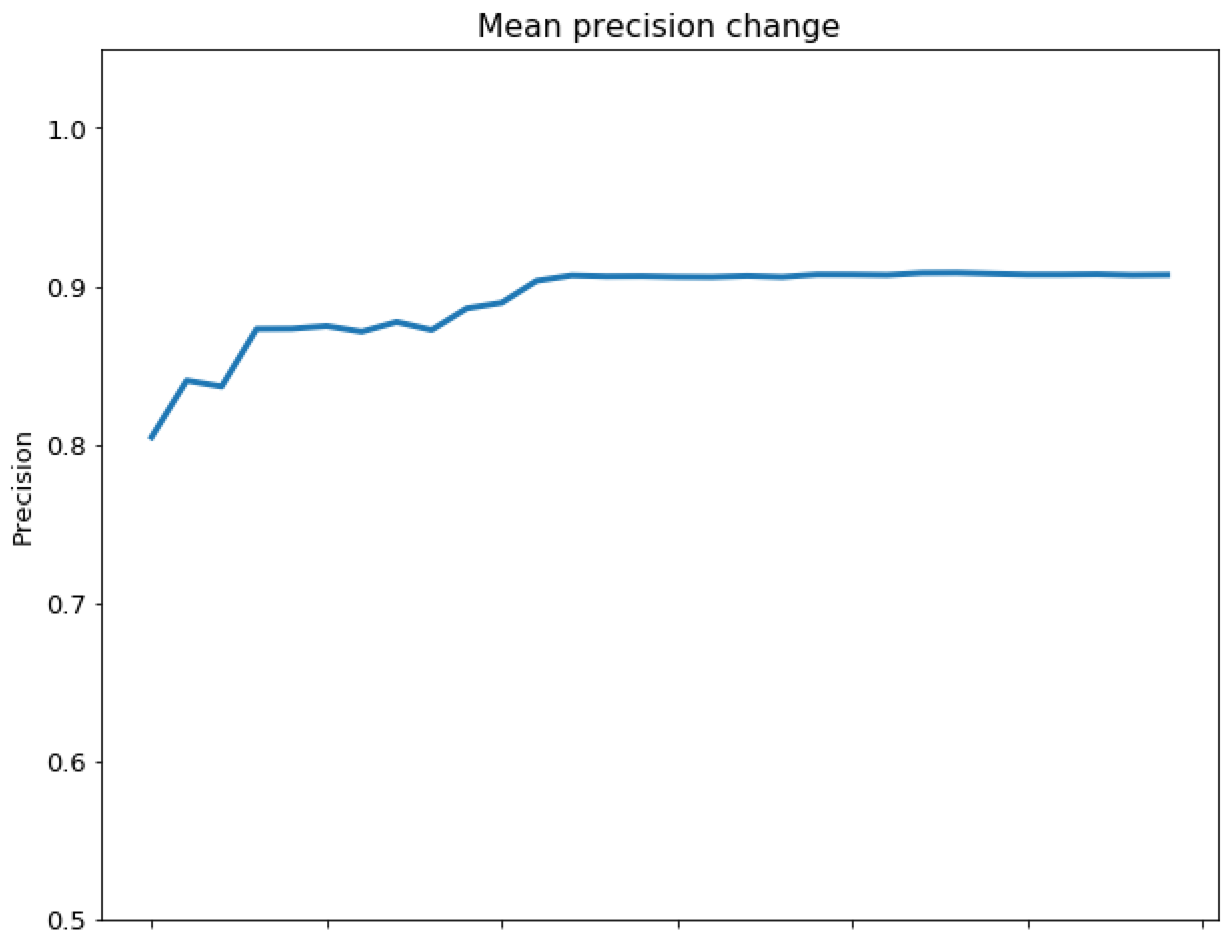

```
#metrics
#loss change of last
lw = 3
plt.figure(figsize=(10, 8))
plt.plot(range(0,epochs), metr[ : , 4:5].tolist(), lw=lw, label='F1-score change')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Test-loss change')
plt.show()
#accuracy change
plt.figure(figsize=(10, 8))
plt.plot(range(0,epochs), metr[ : , :1].tolist(), lw=lw, label='Precision change')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1.05])
plt.title('Mean accuracy change')
plt.show()
#precision change
plt.figure(figsize=(10, 8))
plt.plot(range(0,epochs), metr[ : , 1:2].tolist(), lw=lw, label='Precision change')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.ylim([0.5, 1.05])
plt.title('Mean precision change')
plt.show()
#recall change
plt.figure(figsize=(10, 8))
plt.plot(range(0,epochs), metr[ : , 2:3].tolist(), lw=lw, label='Recall change')
plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.ylim([0.5, 1.05])
plt.title('Mean recall change')
```

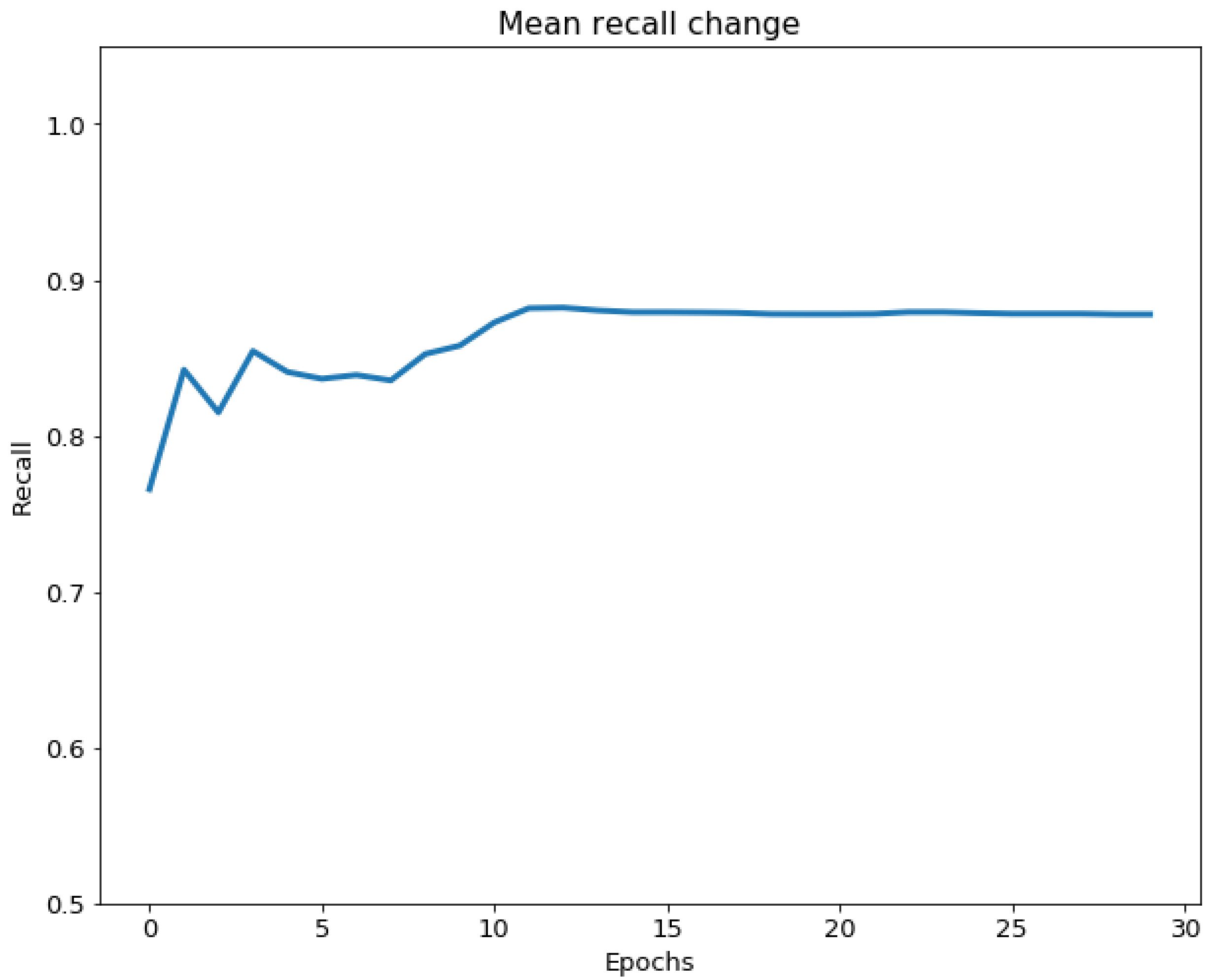
```
plt.show()
#f1 change
plt.figure(figsize=(10, 8))
plt.plot(range(0,epochs), metr[ : , 3:4].tolist(), lw=lw, label='F1-score change')
plt.xlabel('Epochs')
plt.ylabel('F1-score')
plt.ylim([0.5, 1.0])
plt.title('Mean F1-score change')
plt.show()
```


Test-loss change
Loss
Epochs
1.1
1.0
0.9
0.8
0.7
0.6
0
5
10
15
20
25
30


Mean accuracy change
1.0
0.9
0.8


Accura
0.7
0.6
0.5
0
5
10
15
20
25
30
Epochs


Mean precision change
1.0
0.9
0.8
0.7
0.6
0.5
Precision



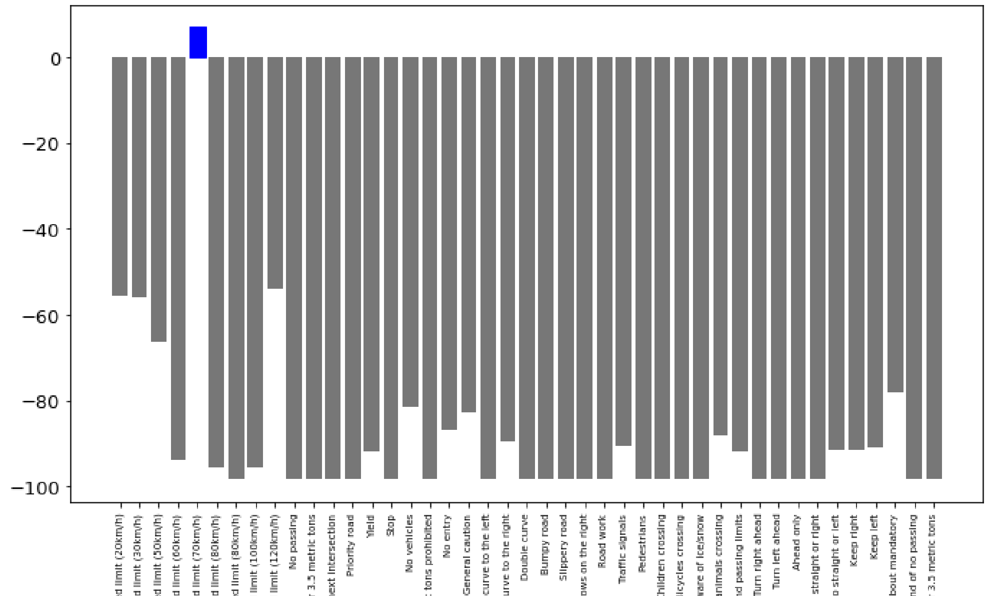

```
# func that builds plot of image with true and predicted labels
def plot_image(predictions_array, true_label, img):
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(img)
  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
```

```
  else:
    color = 'red'
  plt.xlabel("{} {:2.0f}% \n {}".format(label_names['SignName'][predicted_label],
                               100*np.max(predictions_array),
                               label_names['SignName'][true_label]),
                               color=color, fontsize = 15)
# func that builds plot of prediction
def plot_value_array(predictions_array, true_label):
  tick_marks = np.arange(len(label_names['SignName']))
  plt.xticks(tick_marks, label_names['SignName'], rotation=90, fontsize = 7)
  thisplot = plt.bar(range(43), predictions_array[0], color="#777777")
  predicted_label = np.argmax(predictions_array)
  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

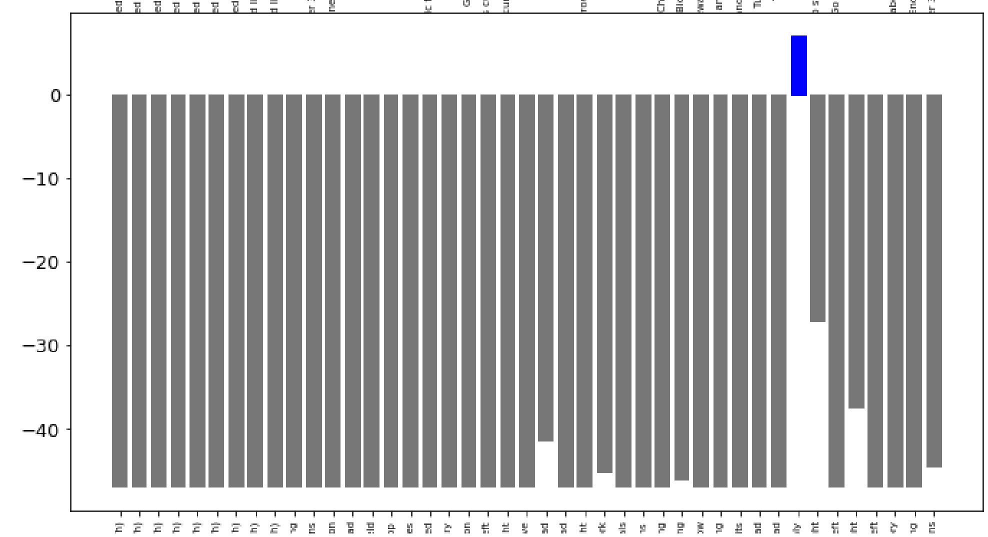
```
# results of the model on certain examples
num_rows = 5
num_cols = 2
num_images = num_rows*num_cols
pred_time = 0
plt.figure(figsize = (25, num_rows * 15))
for k in range(num_images):
  i = random.randint(0, X_train.shape[0] - 1)
  pred_time = time()
  output = (model(train_dataset[i][0].view(1,3,32,32).to(device)) + 7).tolist()
  pred_time = time() - pred_time
  plt.subplot(2 * num_rows, num_cols, 2*k+1)
  plot_image(output, train_dataset[i][1].item(), train_dataset[i][0].view(32,32,3).numpy().astype(int))
  plt.subplot(2 * num_rows, num_cols, 2*k+2)
  plot_value_array(output, train_dataset[i][1].item())
  print("Time for predition ", k, " : ",pred_time)
```

```
Time for predition  0  :  0.005733489990234375
Time for predition  1  :  0.0012807846069335938
Time for predition  2  :  0.0011508464813232422
Time for predition  3  :  0.0010459423065185547
Time for predition  4  :  0.001005411148071289
Time for predition  5  :  0.0008223056793212891
Time for predition  6  :  0.0010573863983154297
Time for predition  7  :  0.0007305145263671875
Time for predition  8  :  0.0010480880737304688
Time for predition  9  :  0.00078582763671875
```

Speed limit (70km/h) 700%
Speed limit (70km/h)



Ahead only 700%



Conclusion:

1. Edge cases:

```
This model works good only with such kind of picture(like in dataset).
  It means that turned images of signes, signes which were partly closed by something/one or grayscale images... can be wrongly predicted.
  We can't just turn them all, because our dataset contains different signs.
  If we turn sign 'forward' to left than it become 'turn left sign'. And we will get wrong prediction.
  Grayscale images will have problem with classification because some signs are the same in Grayscale, but different in RGB(differs only in color)
```

2. My results:

```
  The best result I have received by changing(big number of times) all possible variables: 93%(92% here). (accuracy)
  Metrics: Presicion - 0.89
           Recall    - 0.87
           F1-score  - 0.87
  Time for 1 prediction is: ~(0.0005 - 0.002).
  Epoch time: ~ 17.5
  Train time:  536.5 ( it can be decreased to 266.5 (15 epoch) and result will be same)
  If we look on confusion matrix we can see there that there are some similar signs which are problematic for our model to predict.
  It can also be seen on the predictions examples(probabilities are close to chosen).
  For example, while I was trying to find best params for my model, I got situation when all Speed limit (20km/h) signs were predicted as Speed li
  It can be because there is different number of each sign pictures and model just don't get a punishment for it's faulties.
  This small number of picture is like outliers for model(especially if batch_size is big).
  Also we can see from metrics that, in this case, them grow with each epoch and then just stop to do it(model don't become better).
  There often were situation when model become worse and worse with each epoch. I tried to fix it decreasing learning_rate on, for instance, 10 ep
```