PersonalCNNComplete

July 11, 2020

[1]:
```
# imports for array-handling and plotting
import numpy as np
import matplotlib
import csv
import time
matplotlib.use('agg')
import matplotlib.pyplot as plt
%matplotlib inline

# let's keep our keras backend tensorflow quiet
import os

# keras imports for the dataset and building our neural network
import tensorflow as tf
from tensorflow.keras import models, layers, datasets, optimizers
from tensorflow.keras.callbacks import TensorBoard, EarlyStopping,␣
 ↪ModelCheckpoint
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout, Activation, Conv2D,␣
 ↪MaxPooling2D, Flatten
from keras.utils import np_utils
from tensorflow.keras import backend as K
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
Using TensorFlow backend.
```

[2]:
```
#Impone ordinamento cartelle se hanno nomi di lunghezza diversa
def classificator (directory):
    childDirectories = next(os.walk(directory))[1]
    for x in range(len(childDirectories)):
        childDirectories[x]=int(childDirectories[x])
    childDirectories.sort()
    for x in range(len(childDirectories)):
        childDirectories[x]=str(childDirectories[x])
    return childDirectories
```

[3]:
```
#PREAMBOLO:
# 1. versione del dataset
# 2. risorse (CPU/GPU)
# 3. augmentation sul dataset di train e/o di test
# 4. early stopping o meno

# IMPOSTO MACROPARAMETRI E VARIABILI GLOBALI
img_width, img_height = 32, 32
epoche = 15
batch_size = 100
split_per_validazione = 0.15

# 1. versione del dataset
# settare 'sceltaDataset' come "10RandomClasses", "10PoorestClasses" o␣
 ↪"43Classes"
sceltaDataset = "10RandomClasses"

# 2. risorse (CPU/GPU)
# settare 'programmatore' come "LN" per impostare le GPU, altrimenti qualsiasi␣
 ↪altro modo per settare le CPU
programmatore = "LP"

# 3. augmentation sul dataset di train e/o di test
# Se desideri l'augmentation del train o nel test imposta trainAg e/o testAg␣
 ↪come 'SI'
trainAg = 'SI'
testAg = 'SI'

# 4. early stopping o meno
# Se desideri earlyStopping imposta la variabile a 'SI'
earlyStopping = 'SI'
```

[4]:
```
#CONTROLLI VARI

#CONTROLLO E SETTAGGIO DEL DATASET:
if sceltaDataset == "10RandomClasses":
    numeroClassi = 10
    origine = 'datasets/TrafficSignClassification-10RandomClasses'
    nb_train_samples = 33000
    nb_test_samples = 3300
elif sceltaDataset == "10PoorestClasses":
    numeroClassi = 10
    origine = 'datasets/TrafficSignClassification-10PoorestClasses'
    nb_train_samples = 4400
    nb_test_samples = 720
elif sceltaDataset == "43Classes":
    numeroClassi = 43
```

```
    origine = 'datasets/TrafficSignClassification-43Classes'
    nb_train_samples = 65000
    nb_test_samples = 9000
else:
    raise SystemExit("Dataset non riconosciuto")

#Imposto directory del dataset già decompresso dallo zip
train_data_dir = origine+'/train'
test_data_dir = origine+'/test'
label_info_dir = origine+'/labels.csv'

#Attivazione della CPU o della GPU
if programmatore == "LN":
    # for testing on GPU
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
else:
    # for testing on CPU
    os.environ['CUDA_VISIBLE_DEVICES'] = '-1'


#CREAZIONE PATH DI SALVATAGGIO COMUNE
creoNomeAddestramento='PersonalCNN-'+sceltaDataset+'-'+str(epoche)+'Epoche-EarlyStopping'+earl
pathSalvaModello=creoNomeAddestramento+'.h5'
pathStampaModello='models/modelloDi'+sceltaDataset+'.png'


#IMPOSTO L'AUGMENTANTION:
#Default:
train_shear_range, test_shear_range = 0.0, 0.0
train_zoom_range, test_zoom_range = 0.0, 0.0
train_width_shift_range, test_width_shift_range = 0.0, 0.0
train_height_shift_range, test_height_shift_range = 0.0, 0.0
train_fill_mode, test_fill_mode = "constant", "constant" #riempe i contorni in␣
 ↪caso di modifica, messo costant così è nero
#Attivazione augmentation se richiesta
if trainAg == 'SI':
    train_shear_range=0.15 #Distorsione angolare. Ne distorce un po' la forma,␣
 ↪quindi la tengo
    train_zoom_range=0.15 #Zoom da applicare alla foto. Ne distorce un po' la␣
 ↪forma, quindi la tengo
    train_width_shift_range=0.15 #spostamento orizzontale della foto per␣
 ↪analizzarla anche da tagliata
    train_height_shift_range=0.15 #spostamento verticale della foto per␣
 ↪analizzarla anche da tagliata
if testAg == 'SI':
    test_shear_range=0.1
    test_zoom_range=0.1
```

```
    test_width_shift_range=0.1
    test_height_shift_range=0.1


#IMPOSTO EARLYSTOPPING:
if earlyStopping == 'SI':
    #Implemento l'EarlyStopping
    tf.keras.callbacks.EarlyStopping(
        monitor='val_loss', min_delta=0, patience=0, verbose=0, mode='auto',
        baseline=None, restore_best_weights=False)
    checkpoint_filepath = '/tmp/checkpoint'
    epocheSalvataggio=5
    model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
            filepath=checkpoint_filepath, save_weights_only=True,
            monitor='val_loss', save_freq=epocheSalvataggio,
            save_best_only=True)
    callback = [ TensorBoard(update_freq=521),
                    EarlyStopping(monitor='val_accuracy', patience=2),
                    ModelCheckpoint("results/"+pathSalvaModello,␣
 ↪save_best_only=True), ]
else:
    callback = None


print("Nome Addestramento:")
print(creoNomeAddestramento)
```

```
Nome Addestramento:
PersonalCNN-10RandomClasses-15Epoche-EarlyStoppingSI-TrainAgSI-TestAgSI
```

[5]:
```
def make_train_generator():
    # Configurazione di augmentation per il training
    train_datagen = ImageDataGenerator(
        rescale=1. / 255,
        validation_split=split_per_validazione, #Quanto deve splittare il train␣
 ↪per creare la validation
        shear_range=train_shear_range,
        zoom_range=train_zoom_range,
        width_shift_range=train_width_shift_range,
        height_shift_range=train_height_shift_range,
        fill_mode=train_fill_mode,
        horizontal_flip=False,
        vertical_flip=False
        #rotation_range=20, #Rotazione massima, espressa in intero. Disattiva␣
 ↪perché i cartelli non si trovano ruotati
        )
    train_generator = train_datagen.flow_from_directory(
```

```
        train_data_dir,
        target_size=(img_width, img_height),
        color_mode="rgb",
        batch_size=int(nb_train_samples*(1-split_per_validazione)),
        class_mode='categorical',
        subset='training')
    return train_generator

def make_validation_generator():
    # Configurazione di augmentation per il training
    train_datagen = ImageDataGenerator(
        rescale=1. / 255,
        validation_split=split_per_validazione, #Quanto deve splittare il train␣
↪per creare la validation
        shear_range=train_shear_range,
        zoom_range=train_zoom_range,
        width_shift_range=train_width_shift_range,
        height_shift_range=train_height_shift_range,
        fill_mode=train_fill_mode,
        horizontal_flip=False,
        vertical_flip=False
        #rotation_range=20, #Rotazione massima, espressa in intero. Disattiva␣
↪perché i cartelli non si trovano ruotati
        )
    validation_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        color_mode="rgb",
        batch_size=int(nb_train_samples*split_per_validazione),
        class_mode='categorical',
        subset='validation')
    return validation_generator

def make_test_generator():
    # Configurazione di augmentation per il test (solo rescaling nel caso non si␣
↪voglia augmentation)
    test_datagen = ImageDataGenerator(
        rescale=1. / 255,
        shear_range=test_shear_range,
        zoom_range=test_zoom_range,
        width_shift_range=test_width_shift_range,
        height_shift_range=test_height_shift_range,
        fill_mode=test_fill_mode,
        horizontal_flip=False,
        vertical_flip=False
        #rotation_range=20, #Rotazione massima, espressa in intero. Disattiva␣
↪perché i cartelli non si trovano ruotati
```

```
        )
    test_generator = test_datagen.flow_from_directory(
        test_data_dir,
        target_size=(img_width, img_height),
        color_mode="rgb",
        batch_size=nb_test_samples,
        class_mode="categorical")
    return test_generator

#IMPORTANTE: PER MAGGIOR INFO SULL'AUGMENTATION DELLA IMAGEDATAGENERATOR, E'␣
 ↪RIPORTATO IL LINK:
#https://fairyonice.github.io/Learn-about-ImageDataGenerator.html
```

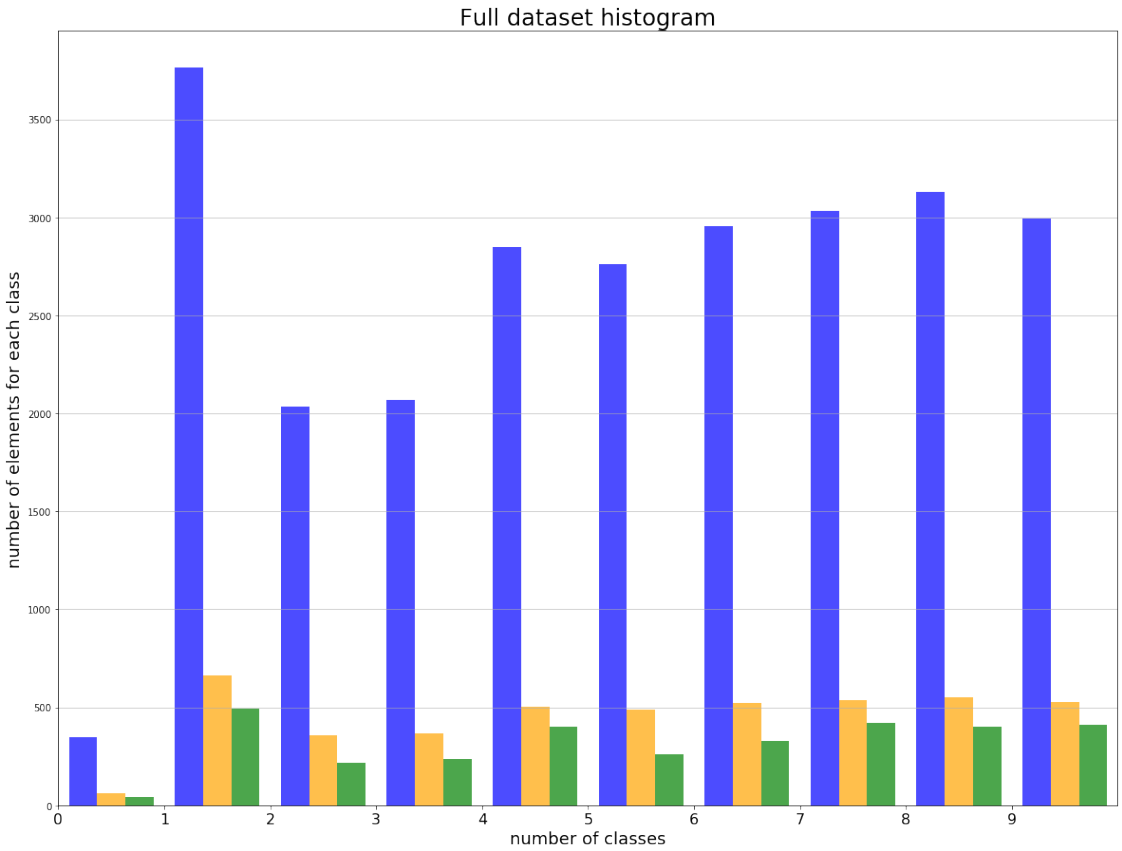
[6]:
```
#Importo i nomi delle classi
with open(label_info_dir, newline='') as f:
    reader = csv.reader(f)
    data = list(reader)
#elementi le prime entry che non sono dati utili
data.pop(0)
data.pop(0)
listaNomiClassi = [];
for i in range(len(data)):
    #Divido la riga per poi prendere l'elemento che mi serve
    rigaInEsame = data[i][0].split(";")
    listaNomiClassi+=[rigaInEsame[1]]
```

[7]:
```
#import del dataset usando tf.data.Dataset
train_dataset = tf.data.Dataset.from_generator(make_train_generator, (tf.
 ↪float32, tf.float32))
validation_dataset = tf.data.Dataset.from_generator(make_validation_generator,␣
 ↪(tf.float32, tf.float32))
test_dataset = tf.data.Dataset.from_generator(make_test_generator, (tf.float32,␣
 ↪tf.float32))
```

[8]:
```
#ricavare immagini e label dai dataset creati con i generator
for images, labels in train_dataset.take(1):
    X_train = images.numpy()
    Y_train = labels.numpy()

for images, labels in validation_dataset.take(1):
    X_valid = images.numpy()
    Y_valid = labels.numpy()

for images, labels in test_dataset.take(1):
    X_test = images.numpy()
    Y_test = labels.numpy()
```

```
Found 25939 images belonging to 10 classes.
Found 4573 images belonging to 10 classes.
Found 3209 images belonging to 10 classes.
```

[9]:
```
#HISTOGRAM REPRESENTATION
#preliminari: conversione da one-hot-encoding a interi per visionare il numero␣
 ↪di sample per dataset
y_train = np.empty(len(Y_train),dtype=int)
y_valid = np.empty(len(Y_valid),dtype=int)
y_test = np.empty(len(Y_test),dtype=int)
for j in range(len(Y_train)):
    y_train[j]=(np.where(Y_train[j]==1)[0][0])

for j in range(len(Y_valid)):
    y_valid[j]=(np.where(Y_valid[j]==1)[0][0])

for j in range(len(Y_test)):
    y_test[j]=(np.where(Y_test[j]==1)[0][0])

#Quest'ultima cosa serve per la stampa finale
immaginiPerLaStampaFinale=X_test

#genero Valori Asse X
valAsseX = []
for i in range(numeroClassi):
    valAsseX+=[i]

#creo allineamento per i risultati dell'asse x
def bins_labels(bins, **kwargs):
    bin_w = (max(bins) - min(bins)) / (len(bins) - 1)
    plt.xticks(np.arange(min(bins)+bin_w/2, max(bins), bin_w), bins, **kwargs)
    plt.xlim(bins[0], bins[-1])

plt.figure(num=None, figsize=(20, 15))
bins = range(numeroClassi+1)
plt.hist([y_train, y_valid, y_test], bins=bins, color=['blue', 'orange',␣
 ↪'green'], alpha=0.7)
bins_labels(bins, fontsize=16)
plt.xlabel("number of classes", fontsize=18)
plt.ylabel("number of elements for each class", fontsize=18)
plt.xticks(valAsseX)
plt.title('Full dataset histogram', fontsize=24)
plt.grid(axis='y', alpha=0.7)
#plt.savefig('models/hist'+sceltaDataset+'.png')
plt.show()
```



[10]:
```
#Stampa prima delle classi presenti, con la corrispondente digits, e poi le␣
 ↪immagini con la corrispondente digits
listaClassiCheVerrannoStampate = []
for i in range(9):
    #Creo la lista di classi e valori che verranno stampate
    valoreDigits=np.where(Y_train[i]==1)[0][0]
    nomeDigits=listaNomiClassi[valoreDigits]
    if [valoreDigits, nomeDigits] not in listaClassiCheVerrannoStampate :
        listaClassiCheVerrannoStampate+=[[valoreDigits, nomeDigits]]

    #Stampa classica delle digits
    plt.subplot(3,3,i+1)
    plt.tight_layout()
    plt.imshow(X_train[i], interpolation='none')
    plt.title("Digit: {}".format(valoreDigits))
    plt.xticks([])
    plt.yticks([])

#Prima stampo le classi presenti, con la corrispondente digits in ordine di␣
 ↪digits
```

```
for i in range(numeroClassi):
    for j in range(len(listaClassiCheVerrannoStampate)):
        if listaClassiCheVerrannoStampate[j][0]==i:
            print("Digit: '{}' corrisponde a '{}'".
 ↪format(listaClassiCheVerrannoStampate[j][0],
                                                                  ␣
 ↪listaClassiCheVerrannoStampate[j][1]))
#Stampo i cartelli
plt.show()
```

```
Digit: '1' corrisponde a 'Speed limit (30km/h)'
Digit: '2' corrisponde a 'Speed limit (50km/h)'
Digit: '4' corrisponde a 'Speed limit (70km/h)'
Digit: '5' corrisponde a 'Speed limit (80km/h)'
Digit: '6' corrisponde a 'No passing for vechiles over 3.5 metric tons'
Digit: '7' corrisponde a 'Priority road'
Digit: '9' corrisponde a 'Keep right'
```

[11]:
```
# Creazione un modello di addestramento sequenziale
def model_sequential():
#per usare leaky relu: activation=tf.nn.leaky_relu
    model = Sequential()
    model.add(Conv2D(32,kernel_size=(7, 7),activation=tf.nn.leaky_relu␣
 ↪,input_shape=(img_width, img_height, 3)))
```

```
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32,kernel_size=(5, 5),activation=tf.nn.leaky_relu))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32,kernel_size=(3, 3),activation=tf.nn.leaky_relu))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(512, activation=tf.nn.leaky_relu))
    model.add(Dense(numeroClassi, activation='softmax'))
    return model
```

[12]:
```
nostroModello = model_sequential()
nostroModello.summary()
tf.keras.utils.plot_model(nostroModello, to_file=pathStampaModello)
nostroModello.
 ↪compile(loss="categorical_crossentropy",optimizer="adam",metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        4736
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 9, 9, 32)          25632
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 2, 2, 32)          9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 512)               16896
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 61,642
Trainable params: 61,642
Non-trainable params: 0
_________________________________________________________________
```

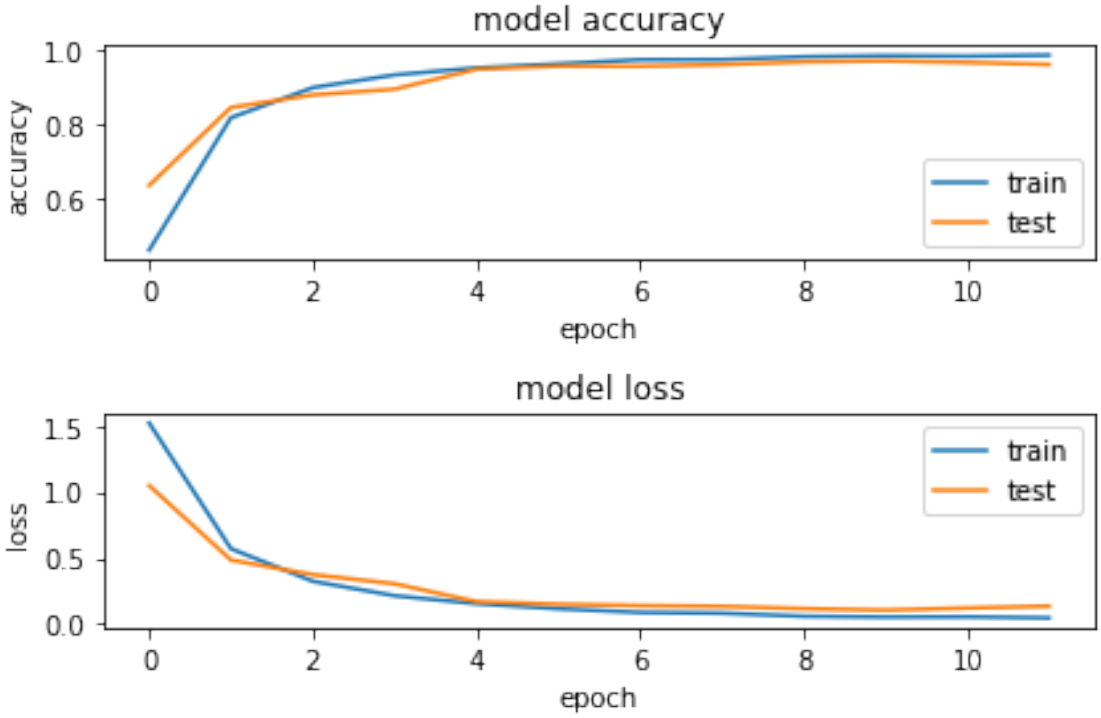
[13]:
```
#Inizio a contare il tempo di addestramento
t0 = time.time()

#Avvio Addestramento
```

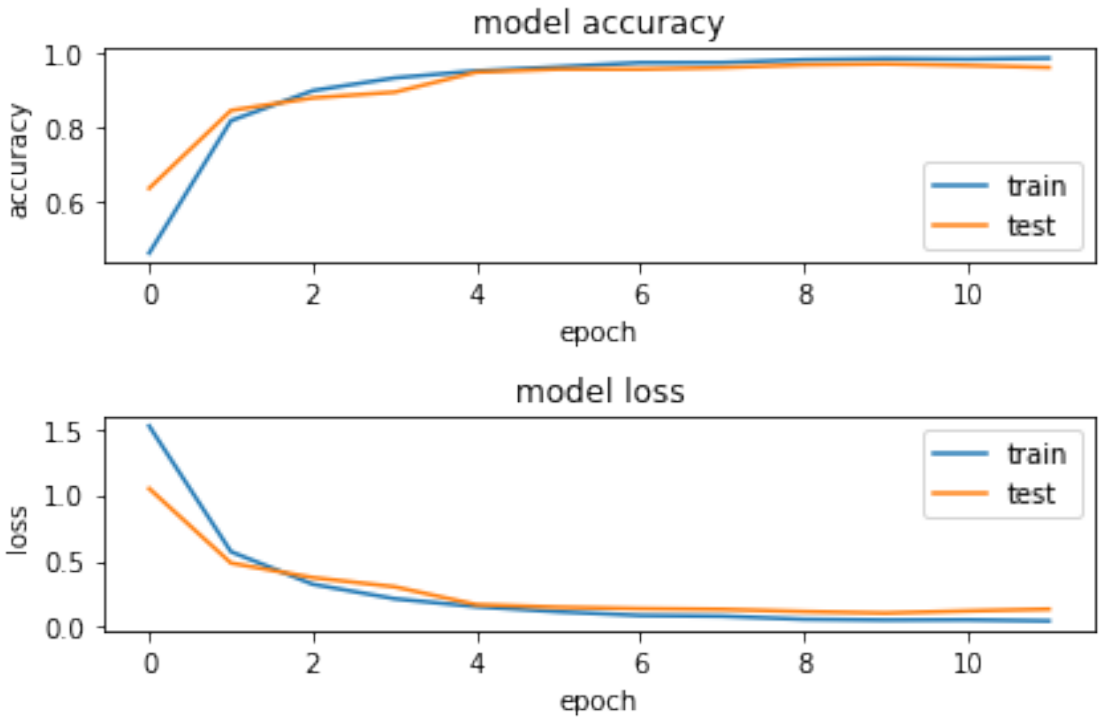
```
history = nostroModello.fit(X_train, Y_train,
                    batch_size=batch_size,
                    epochs=epoche,
                    verbose=2,
                    validation_data=(X_valid, Y_valid),
                    callbacks=callback)


# Stampo l'andamento del train
fig = plt.figure()
plt.subplot(2,1,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')

plt.tight_layout()

fig
```

```
Train on 25939 samples, validate on 4573 samples
Epoch 1/15
25939/25939 - 13s - loss: 1.5229 - accuracy: 0.4629 - val_loss: 1.0488 -
val_accuracy: 0.6363
Epoch 2/15
25939/25939 - 11s - loss: 0.5700 - accuracy: 0.8174 - val_loss: 0.4843 -
val_accuracy: 0.8447
Epoch 3/15
25939/25939 - 13s - loss: 0.3233 - accuracy: 0.8978 - val_loss: 0.3740 -
val_accuracy: 0.8784
Epoch 4/15
25939/25939 - 13s - loss: 0.2138 - accuracy: 0.9318 - val_loss: 0.3044 -
val_accuracy: 0.8937
Epoch 5/15
25939/25939 - 11s - loss: 0.1536 - accuracy: 0.9508 - val_loss: 0.1693 -
val_accuracy: 0.9473
Epoch 6/15
```

```
25939/25939 - 14s - loss: 0.1153 - accuracy: 0.9614 - val_loss: 0.1496 -
val_accuracy: 0.9554
Epoch 7/15
25939/25939 - 11s - loss: 0.0872 - accuracy: 0.9727 - val_loss: 0.1390 -
val_accuracy: 0.9552
Epoch 8/15
25939/25939 - 9s - loss: 0.0817 - accuracy: 0.9729 - val_loss: 0.1334 -
val_accuracy: 0.9595
Epoch 9/15
25939/25939 - 12s - loss: 0.0582 - accuracy: 0.9806 - val_loss: 0.1166 -
val_accuracy: 0.9670
Epoch 10/15
25939/25939 - 12s - loss: 0.0523 - accuracy: 0.9832 - val_loss: 0.1053 -
val_accuracy: 0.9694
Epoch 11/15
25939/25939 - 14s - loss: 0.0539 - accuracy: 0.9821 - val_loss: 0.1215 -
val_accuracy: 0.9652
Epoch 12/15
25939/25939 - 13s - loss: 0.0460 - accuracy: 0.9847 - val_loss: 0.1347 -
val_accuracy: 0.9600
```

[13]:





[14]:
```
#CALCOLO IL TEMPO DI ADDESTRAMENTO
t1 = time.time()
hours, rem = divmod(t1-t0, 3600)
minutes, seconds = divmod(rem, 60)

print("Tempo d'addestramento in secondi:")
print (t1 - t0)
print("Tempo d'addestramento in ore-minuti-secondi-scartoRimanente:")
print("{:0>2}:{:0>2}:{:05.2f}".format(int(hours),int(minutes),seconds))
```

```
Tempo d'addestramento in secondi:
146.61130118370056
Tempo d'addestramento in ore-minuti-secondi-scartoRimanente:
00:02:26.61
```

[15]:
```
# Salvataggio dell'addestramento e la sua struttura in un file esterno
# VIENE GIÀ FATTO DALL'EARLY STOPPING
if earlyStopping != 'SI':
    save_dir = "results/"
    model_name = pathSalvaModello
    model_path = os.path.join(save_dir, model_name)
    nostroModello.save(model_path)
    print('Saved trained model at %s ' % model_path)
else:
```

```
    print("Modello già precedentemente salvato dall'early stopping all'ottimo")
```

```
Modello già precedentemente salvato dall'early stopping all'ottimo
```

[16]:
```
# Uso l'addestramento appena fatto per calcolare la loss e l'accuratezza sul test
#modello_cartelli = load_model("results/"+pathSalvaModello)

loss_and_metrics = nostroModello.evaluate(X_test, Y_test, verbose=2)

print("Test Loss", loss_and_metrics[0])
print("Test Accuracy", loss_and_metrics[1])
```

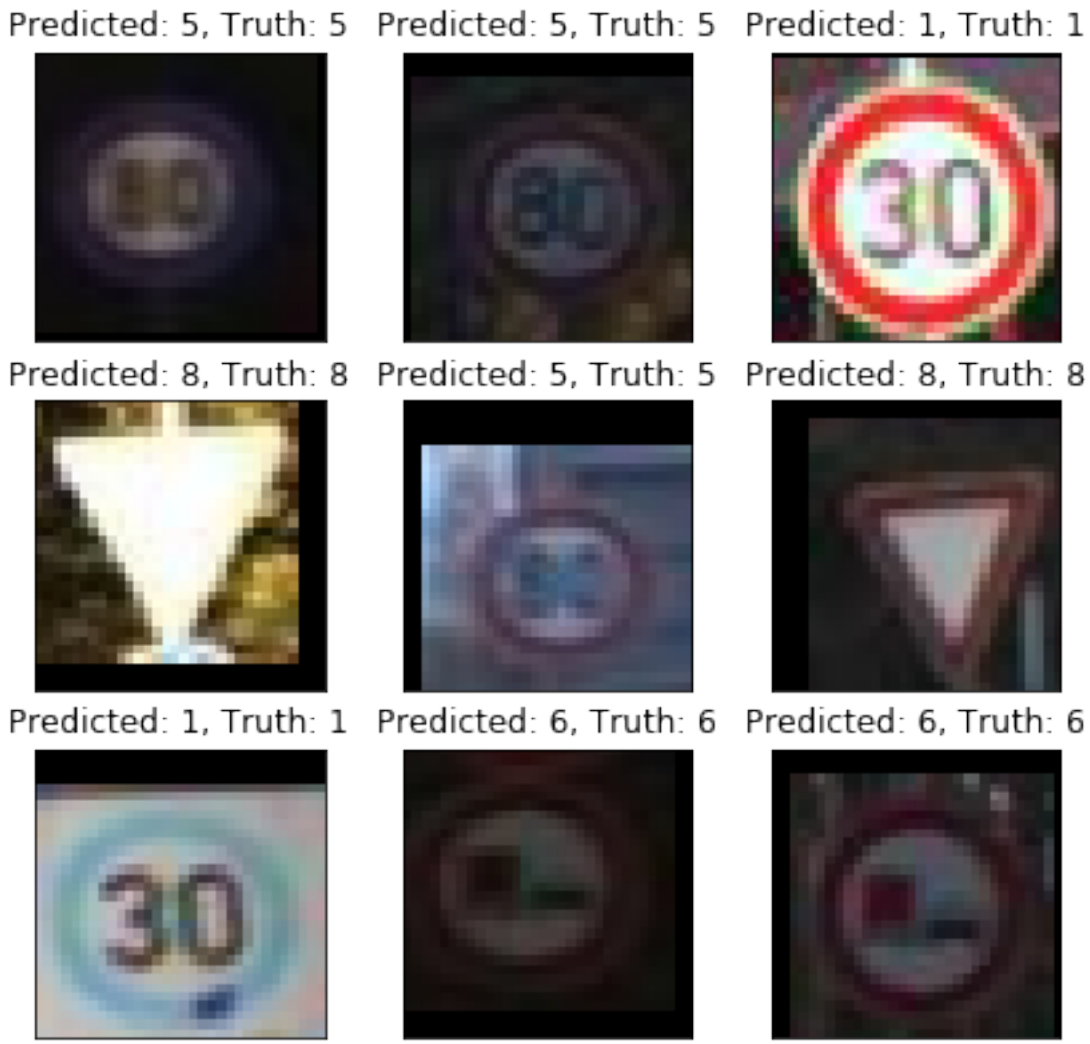
```
3209/3209 - 2s - loss: 0.0645 - accuracy: 0.9832
Test Loss 0.06453229174881933
Test Accuracy 0.98317236
```

[17]:
```
# Creazione delle predizioni sul test set sulla base del modello caricato
predicted_classes = nostroModello.predict_classes(X_test)

# Distinguo cosa è stato predetto bene e cosa no
correct_indices = np.nonzero(predicted_classes == y_test)[0]
incorrect_indices = np.nonzero(predicted_classes != y_test)[0]
print()
print(len(correct_indices)," classified correctly")
print(len(incorrect_indices)," classified incorrectly")

# adapt figure size to accomodate 18 subplots
plt.rcParams['figure.figsize'] = (7,14)
figure_evaluation = plt.figure()

print("\n-------------------------------------------------\n")
print("Le prossime predizioni sono giuste")
print("\n-------------------------------------------------\n")

listaClassiCheVerrannoStampate = []
# Stampa delle 9 predizioni corrette
for i, correct in enumerate(correct_indices[:9]):
    #Creo la lista di classi e valori che verranno stampate
    valoreDigits=predicted_classes[correct]
    nomeDigits=listaNomiClassi[valoreDigits]
    if [valoreDigits, nomeDigits] not in listaClassiCheVerrannoStampate :
        listaClassiCheVerrannoStampate+=[[valoreDigits, nomeDigits]]

    #Stampa classica delle digits
    plt.subplot(6,3,i+1)
    plt.imshow(immaginiPerLaStampaFinale[correct], interpolation='none')
    plt.title(
```

```
      "Predicted: {}, Truth: {}".format(predicted_classes[correct],
                                        y_test[correct]))
    plt.xticks([])
    plt.yticks([])

#Prima stampo le classi presenti, con la corrispondente digits in ordine di␣
 ↪digits
for i in range(numeroClassi):
    for j in range(len(listaClassiCheVerrannoStampate)):
        if listaClassiCheVerrannoStampate[j][0]==i:
            print("Digit: '{}' corrisponde a '{}'".
 ↪format(listaClassiCheVerrannoStampate[j][0],
                                                                  ␣
 ↪listaClassiCheVerrannoStampate[j][1]))

#Stampo i cartelli
plt.show()

print("\n-------------------------------------------------\n")
print("Le prossime predizioni sono sbagliate")
print("\n-------------------------------------------------\n")

listaClassiCheVerrannoStampate = []
# Stampa delle 9 predizioni incorrette
for i, incorrect in enumerate(incorrect_indices[:9]):
    #Creo la lista di classi e valori che verranno stampate
    valoreDigits=predicted_classes[incorrect]
    nomeDigits=listaNomiClassi[valoreDigits]
    if [valoreDigits, nomeDigits] not in listaClassiCheVerrannoStampate :
        listaClassiCheVerrannoStampate+=[[valoreDigits, nomeDigits]]
    valoreDigits=y_test[incorrect]
    nomeDigits=listaNomiClassi[valoreDigits]
    if [valoreDigits, nomeDigits] not in listaClassiCheVerrannoStampate :
        listaClassiCheVerrannoStampate+=[[valoreDigits, nomeDigits]]

    plt.subplot(6,3,i+10)
    plt.imshow(immaginiPerLaStampaFinale[incorrect], interpolation='none')
    plt.title(
      "Predicted {}, Truth: {}".format(predicted_classes[incorrect],
                                       y_test[incorrect]))
    plt.xticks([])
    plt.yticks([])

#Prima stampo le classi presenti, con la corrispondente digits in ordine di␣
 ↪digits
for i in range(numeroClassi):
    for j in range(len(listaClassiCheVerrannoStampate)):
```

```
        if listaClassiCheVerrannoStampate[j][0]==i:
            print("Digit: '{}' corrisponde a '{}'".
 ↪format(listaClassiCheVerrannoStampate[j][0],
                                                                  ␣
 ↪listaClassiCheVerrannoStampate[j][1]))

#Stampo i cartelli
plt.show()
```

```
3155  classified correctly
54  classified incorrectly

--------------------------------------------------

Le prossime predizioni sono giuste

--------------------------------------------------

Digit: '1' corrisponde a 'Speed limit (30km/h)'
Digit: '5' corrisponde a 'Speed limit (80km/h)'
Digit: '6' corrisponde a 'No passing for vechiles over 3.5 metric tons'
Digit: '8' corrisponde a 'Yield'
```




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

Le prossime predizioni sono sbagliate

------------------------------------------------

Digit: '1' corrisponde a 'Speed limit (30km/h)'
Digit: '2' corrisponde a 'Speed limit (50km/h)'
Digit: '4' corrisponde a 'Speed limit (70km/h)'
Digit: '5' corrisponde a 'Speed limit (80km/h)'
Digit: '9' corrisponde a 'Keep right'
```

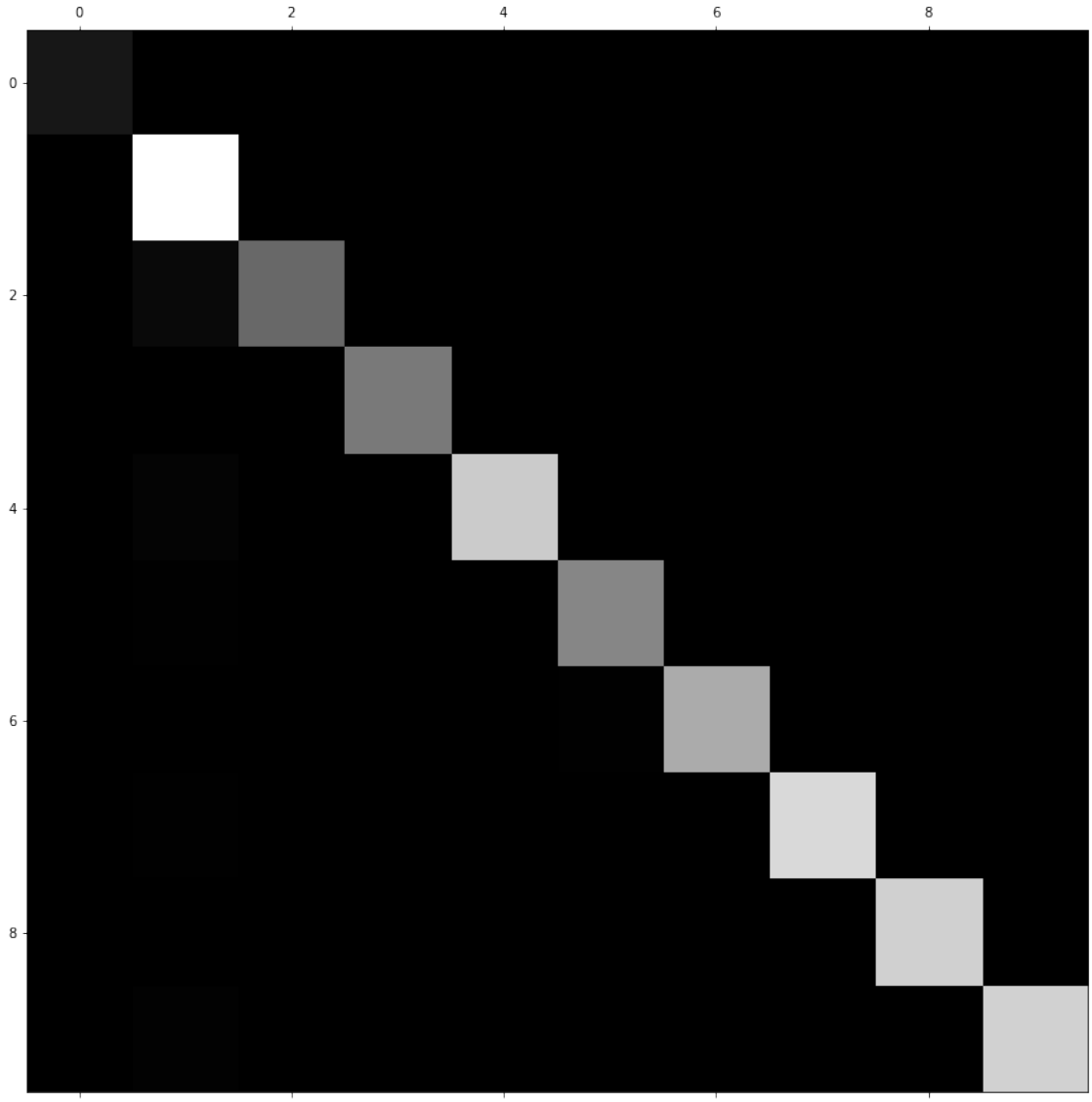
[18]:
```
confusion_matrix = tf.math.confusion_matrix(y_test, predicted_classes,␣
↪num_classes=numeroClassi)
tf.print(confusion_matrix, summarize=-1)
```

```
[[45 0 0 0 0 0 0 0 0 0]
 [1 489 1 0 1 0 0 0 0 0]
 [0 19 199 0 0 1 0 0 0 0]
 [1 0 1 234 0 0 0 0 0 0]
 [1 9 0 1 388 0 0 0 0 0]
 [0 3 0 1 0 256 0 0 0 0]
 [0 0 0 0 0 2 328 0 0 0]
 [0 2 0 0 1 0 0 416 0 1]
 [0 0 0 0 0 0 0 0 399 0]
 [1 4 1 0 1 0 0 0 1 401]]
```

[19]:
```
# ricerca dell'elemento con più errori (se può servire)
row = 0
col = 0
item = 0
for r in range(len(confusion_matrix)):
    for c in range(len(confusion_matrix[r])):
        if (r != c and confusion_matrix[r][c] > item):
            item = confusion_matrix[r][c]
            row = r
            col = c
print("elemento con più errori è in posizione [", row, "][", col, "] con numero␣
 ↪errori pari a:",item.numpy())
```

```
elemento con più errori è in posizione [ 2 ][ 1 ] con numero errori pari a: 19
```

[20]:
```
def plot_confusion_matrix(matrix):
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    cax = ax.matshow(matrix)
    fig.colorbar(cax)

plt.matshow(confusion_matrix.numpy(), cmap=plt.cm.gray)
#plt.savefig('models/confusion_matrix'+sceltaDataset+'.png')
plt.show()
```



[21]:
```
print("Nome Addestramento:")
print(creoNomeAddestramento)
```

```
Nome Addestramento:
PersonalCNN-10RandomClasses-15Epoche-EarlyStoppingSI-TrainAgSI-TestAgSI
```