Traffic Sign Recognition

Version 1.0

January 13, 2021

İ. BAŞAR YARGICI

Table of Contents

1.0. Purpose

1.1. Introduction

This paper provides a description of my term project “Traffic Sign Classifier with Convolutional Neural Network” for Deep Learning and Classification Techniques lecture which is given by Assist. Prof. Dr. Dilek Göksel Duru.

1.2. Scope

Traffic Sign Classifier is designed to classify traffic signs present in the image into different categories.

1.3. Document overview

The remainder of this document is two chapters, the first providing a full description of the project. Second chapter contains the code of project.

2.0. Overall description

So, what is Traffic Sign Recognition and where is it used?

There are several different types of traffic signs like speed limits, no entry, traffic signals, turn left or right, children crossing, no passing of heavy vehicles, etc. Traffic signs classification is the process of identifying which class a traffic sign belongs to.

In the world of Artificial Intelligence and advancement in technologies, many researchers and big companies like Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi are working on autonomous vehicles and self-driving cars. So, for achieving accuracy in this technology, the vehicles should be able to interpret traffic signs and make decisions accordingly. With the help of CNN (Convolutional Neural Network) we can predict that what is the traffic sign on the road.

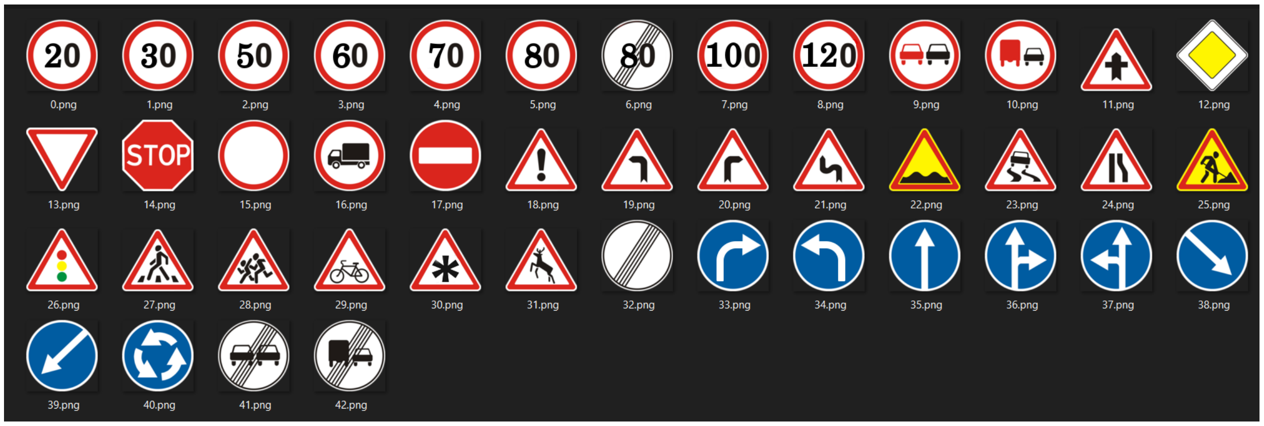
2.1. Dataset

We are using German Traffic Sign Recognition Benchmark (GTSRB) dataset from Kaggle.

Data Size & Shape

- Size of training set: 34,799 (67%)
- Size of the validation set: 4,410 (9%)
- Size of test set: 12,630 (24%)
- Shape of a traffic sign image: (30, 30, 3)
- Number of unique classes/labels: 43

Classes / labels



2.2. Prerequisites (modules-libraries)

- sklearn: machine learning library -> I used it to split data
- tensorflow: machine learning library, also contains keras library -> I used tensorflow.keras which contains functions that will be used in CNN(Convolutional Neural Network) algorithm implementation
- os: provides functions for interacting with the operating system
- pathlib: to interact with the filesystem
- numpy: adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
- matplotlib: plotting library -> I used it to show datas-images
- csv: to read and write files
- pandas: allows importing data from various file formats such as comma-separated values

2.3. Model Architecture Modifications - Hyperparameters

Model 1:

- Convolution layer with 5x5x32 kernel and later relu activation function
- Maxpooling with 2x2 size
- Dropout with 0.25 rate
- Convolution layer with 5x5x64 kernel and later relu activation function
- Maxpooling with 2x2 size
- Dropout with 0.25 rate
- Convolution layer with 4x4x64 kernel and later relu activation function
- Flatten layer to squeeze the layers into 1 dimension
- Dense Fully connected layer with 64 units and relu activation function
- Dense Fully connected layer with 43 units and softmax activation function

✓ With 32 batch size, test accuracy = 91.38559

Model 2:

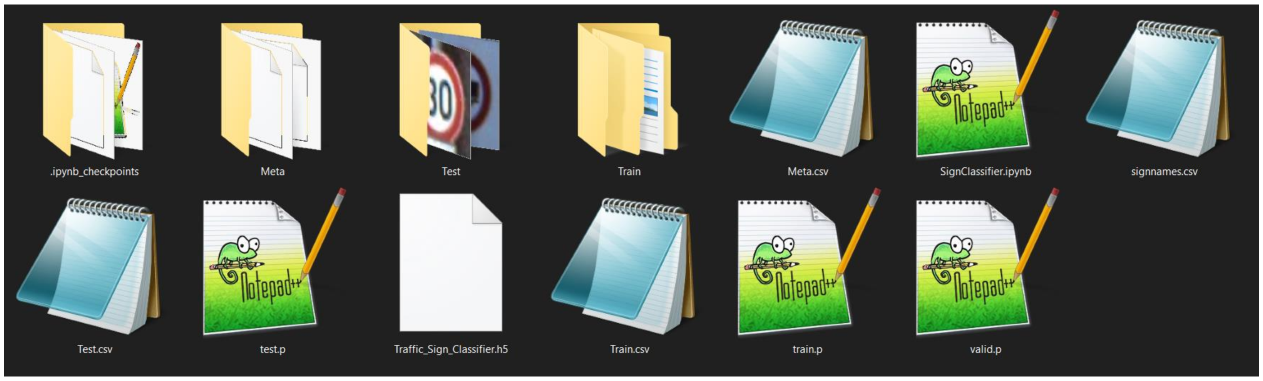
- Convolution layer with 3x3x32 kernel and later relu activation function
- Maxpooling with 2x2 size
- Dropout with 0.25 rate
- Convolution layer with 3x3x64 kernel and later relu activation function
- Maxpooling with 2x2 size
- Dropout with 0.25 rate
- Convolution layer with 3x3x64 kernel and later relu activation function
- Flatten layer to squeeze the layers into 1 dimension
- Dense Fully connected layer with 64 units and relu activation function
- Dense Fully connected layer with 43 units and softmax activation function

✓ With 32 batch size, test accuracy = 91.93191

- ✓ Model 2 gave better result. So, I played with batch sizes of Model 2. While batch size = 64 gave 92,62074 test set accuracy, batch size = 32 gave 91.93191.
- ✓ So, I selected model 2 with 64 batch size.

3.0. Code

Project Structure

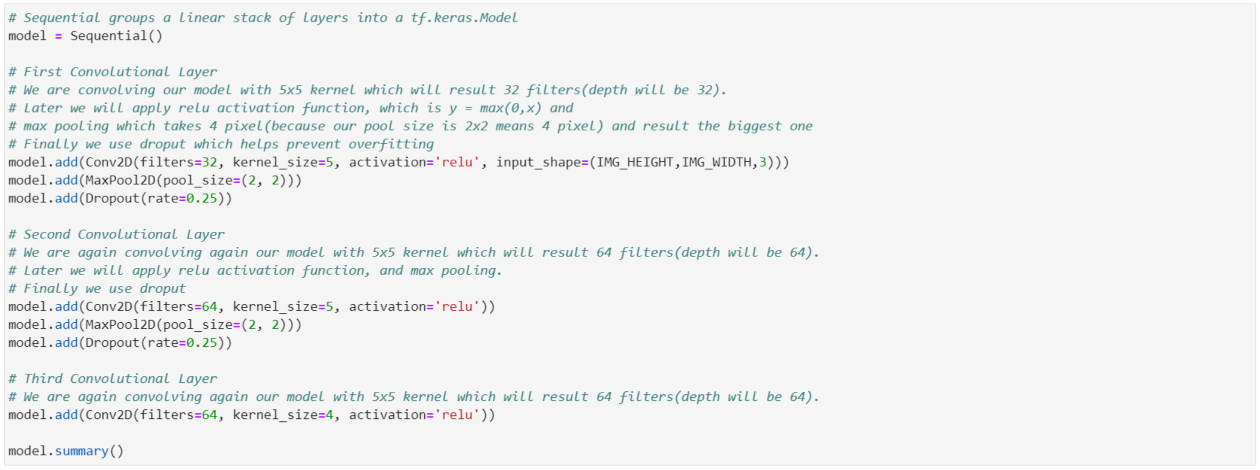


3.1. Last Version of Code

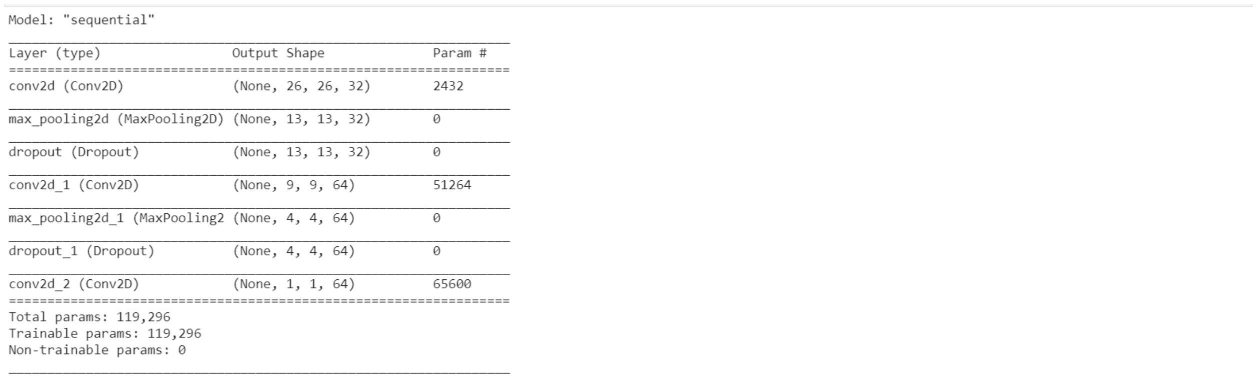
I could not add code properly to paper. Therefore am sharing link of repo to access the code: https://github.com/basarYargici/traffic_sign_classifier_GTSRB

3.2. Modification on model

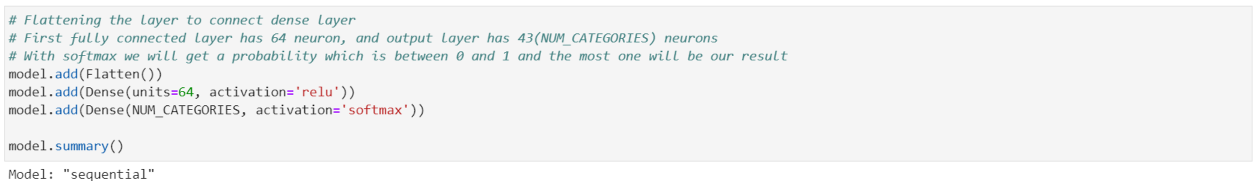
For model 1, following images are the part of training and test part with batch size=32:



```
# Sequential groups a linear stack of layers into a tf.keras.Model
model = Sequential()

# First Convolutional Layer
# We are convolving our model with 5x5 kernel which will result 32 filters(depth will be 32).
# Later we will apply relu activation function, which is y = max(0,x) and
# max pooling which takes 4 pixel(because our pool size is 2x2 means 4 pixel) and result the biggest one
# Finally we use dropout which helps prevent overfitting
model.add(Conv2D(filters=32, kernel_size=5, activation='relu', input_shape=(IMG_HEIGHT,IMG_WIDTH,3)))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

# Second Convolutional Layer
# We are again convolving again our model with 5x5 kernel which will result 64 filters(depth will be 64).
# Later we will apply relu activation function, and max pooling.
# Finally we use dropout
model.add(Conv2D(filters=64, kernel_size=5, activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

# Third Convolutional Layer
# We are again convolving again our model with 5x5 kernel which will result 64 filters(depth will be 64).
model.add(Conv2D(filters=64, kernel_size=4, activation='relu'))

model.summary()
```

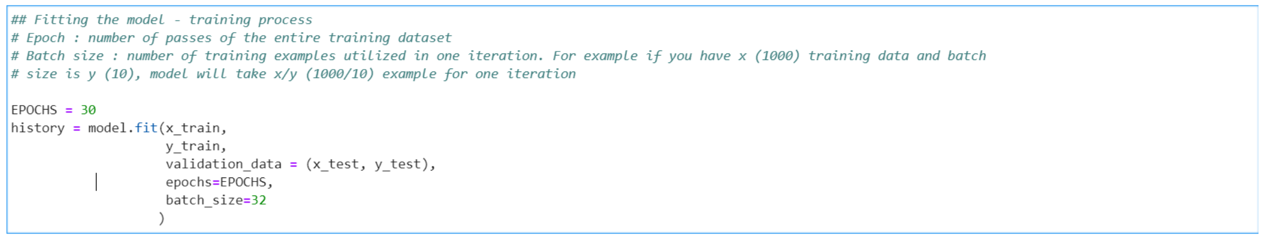


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 9, 9, 64)          51264
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 1, 1, 64)          65600
=================================================================
Total params: 119,296
Trainable params: 119,296
Non-trainable params: 0
_________________________________________________________________
```

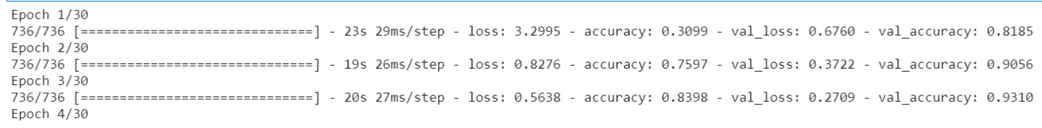


```
# Flattening the layer to connect dense layer
# First fully connected layer has 64 neuron, and output layer has 43(NUM_CATEGORIES) neurons
# With softmax we will get a probability which is between 0 and 1 and the most one will be our result
model.add(Flatten())
model.add(Dense(units=64, activation='relu'))
model.add(Dense(NUM_CATEGORIES, activation='softmax'))

model.summary()
```

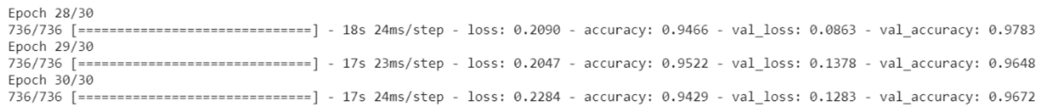
```
Model: "sequential"
```



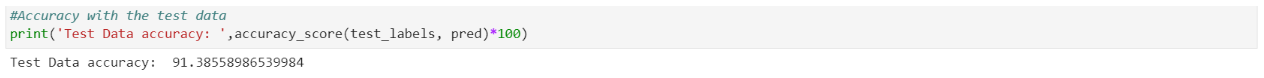
```
## Fitting the model - training process
# Epoch : number of passes of the entire training dataset
# Batch size : number of training examples utilized in one iteration. For example if you have x (1000) training data and batch
# size is y (10), model will take x/y (1000/10) example for one iteration

EPOCHS = 30
history = model.fit(x_train,
                    y_train,
                    validation_data = (x_test, y_test),
                    epochs=EPOCHS,
                    batch_size=32
                   )
```



```
Epoch 1/30
736/736 [==============================] - 23s 29ms/step - loss: 3.2995 - accuracy: 0.3099 - val_loss: 0.6760 - val_accuracy: 0.8185
Epoch 2/30
736/736 [==============================] - 19s 26ms/step - loss: 0.8276 - accuracy: 0.7597 - val_loss: 0.3722 - val_accuracy: 0.9056
Epoch 3/30
736/736 [==============================] - 20s 27ms/step - loss: 0.5638 - accuracy: 0.8398 - val_loss: 0.2709 - val_accuracy: 0.9310
Epoch 4/30
```



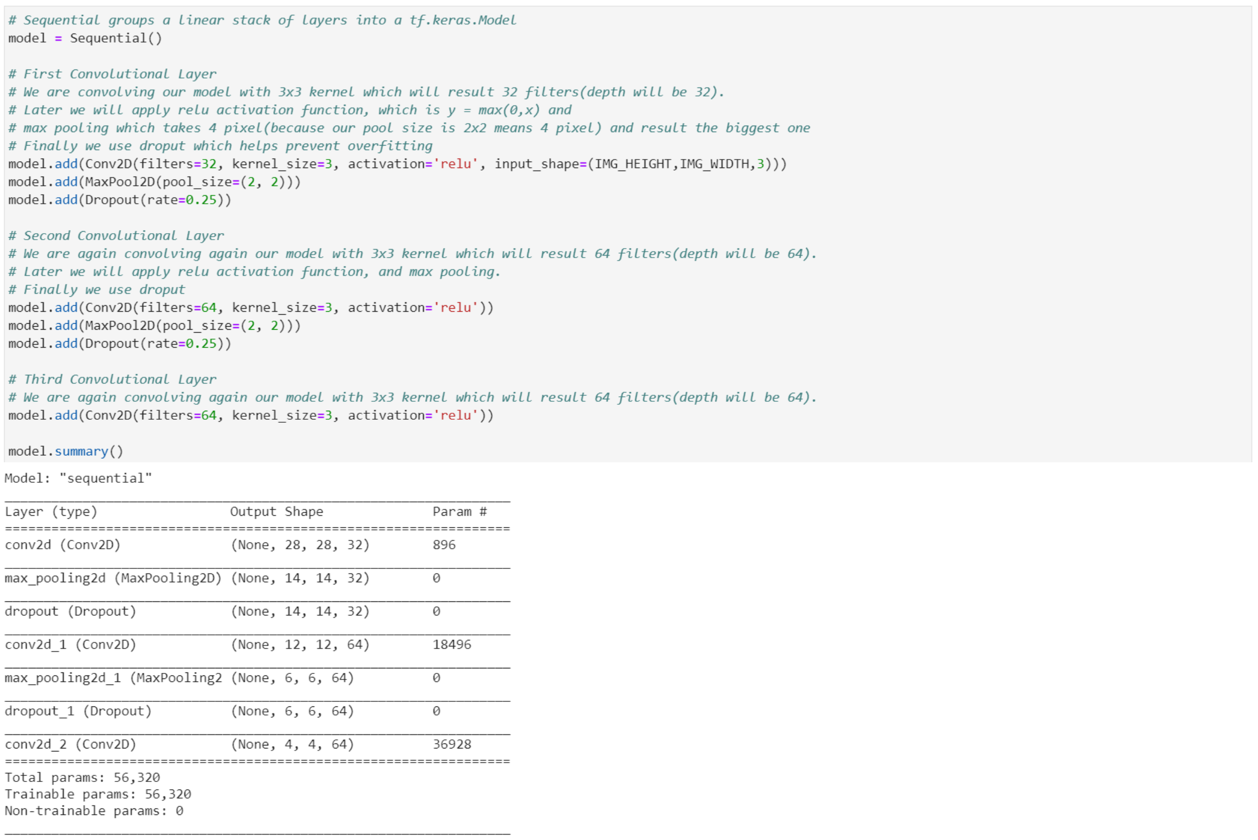
```
Epoch 28/30
736/736 [==============================] - 18s 24ms/step - loss: 0.2090 - accuracy: 0.9466 - val_loss: 0.0863 - val_accuracy: 0.9783
Epoch 29/30
736/736 [==============================] - 17s 23ms/step - loss: 0.2047 - accuracy: 0.9522 - val_loss: 0.1378 - val_accuracy: 0.9648
Epoch 30/30
736/736 [==============================] - 17s 24ms/step - loss: 0.2284 - accuracy: 0.9429 - val_loss: 0.1283 - val_accuracy: 0.9672
```



```
#Accuracy with the test data
print('Test Data accuracy: ',accuracy_score(test_labels, pred)*100)
```

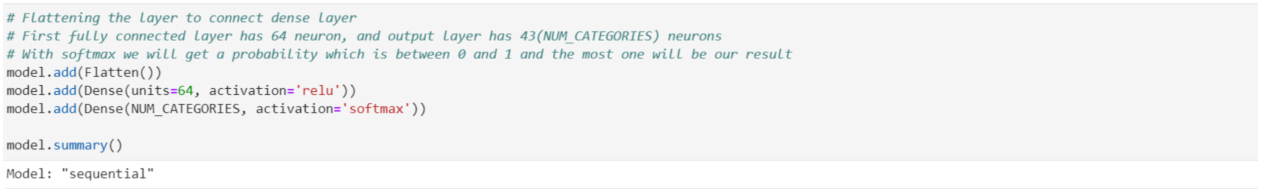
```
Test Data accuracy:  91.38558986539984
```

For model 2, following images are the part of training and test part with batch size=32, test size = 0,4:



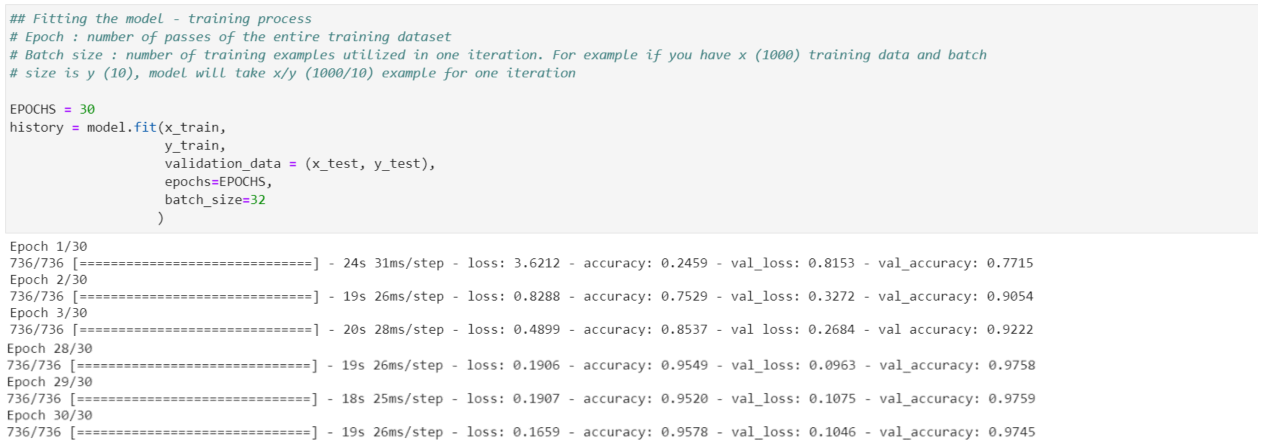
```
# Sequential groups a linear stack of layers into a tf.keras.Model
model = Sequential()

# First Convolutional Layer
# We are convolving our model with 3x3 kernel which will result 32 filters(depth will be 32).
# Later we will apply relu activation function, which is y = max(0,x) and
# max pooling which takes 4 pixel(because our pool size is 2x2 means 4 pixel) and result the biggest one
# Finally we use dropout which helps prevent overfitting
model.add(Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=(IMG_HEIGHT,IMG_WIDTH,3)))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

# Second Convolutional Layer
# We are again convolving again our model with 3x3 kernel which will result 64 filters(depth will be 64).
# Later we will apply relu activation function, and max pooling.
# Finally we use dropout
model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

# Third Convolutional Layer
# We are again convolving again our model with 3x3 kernel which will result 64 filters(depth will be 64).
model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        896       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928     
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```



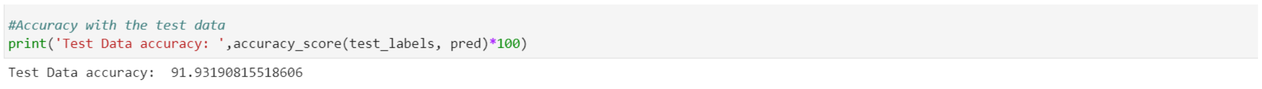
```
# Flattening the layer to connect dense layer
# First fully connected layer has 64 neuron, and output layer has 43(NUM_CATEGORIES) neurons
# With softmax we will get a probability which is between 0 and 1 and the most one will be our result
model.add(Flatten())
model.add(Dense(units=64, activation='relu'))
model.add(Dense(NUM_CATEGORIES, activation='softmax'))

model.summary()
```

```
Model: "sequential"
```



```
## Fitting the model - training process
# Epoch : number of passes of the entire training dataset
# Batch size : number of training examples utilized in one iteration. For example if you have x (1000) training data and batch
# size is y (10), model will take x/y (1000/10) example for one iteration

EPOCHS = 30
history = model.fit(x_train,
                    y_train,
                    validation_data = (x_test, y_test),
                    epochs=EPOCHS,
                    batch_size=32
                   )
```

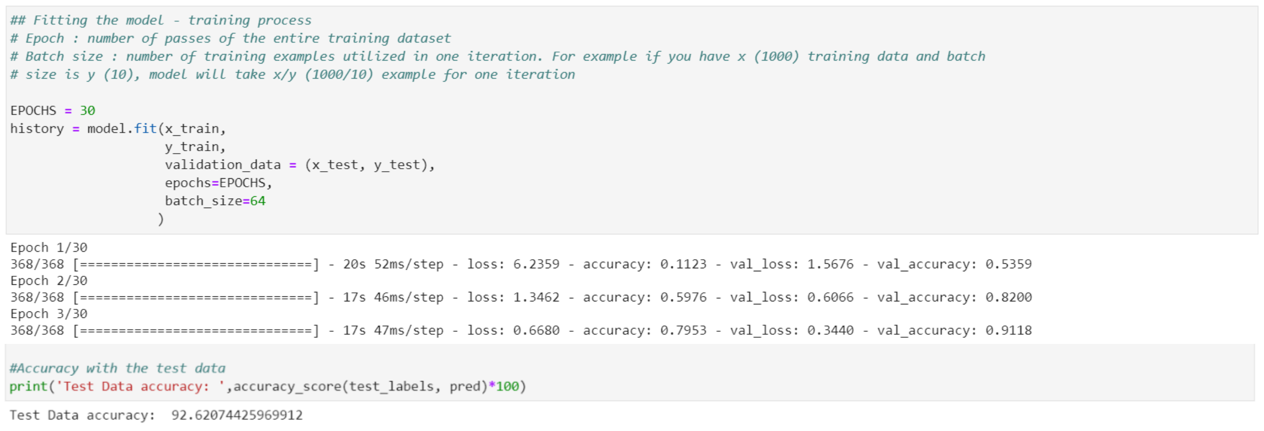
```
Epoch 1/30
736/736 [==============================] - 24s 31ms/step - loss: 3.6212 - accuracy: 0.2459 - val_loss: 0.8153 - val_accuracy: 0.7715
Epoch 2/30
736/736 [==============================] - 19s 26ms/step - loss: 0.8288 - accuracy: 0.7529 - val_loss: 0.3272 - val_accuracy: 0.9054
Epoch 3/30
736/736 [==============================] - 20s 28ms/step - loss: 0.4899 - accuracy: 0.8537 - val loss: 0.2684 - val accuracy: 0.9222
Epoch 28/30
736/736 [==============================] - 19s 26ms/step - loss: 0.1906 - accuracy: 0.9549 - val_loss: 0.0963 - val_accuracy: 0.9758
Epoch 29/30
736/736 [==============================] - 18s 25ms/step - loss: 0.1907 - accuracy: 0.9520 - val_loss: 0.1075 - val_accuracy: 0.9759
Epoch 30/30
736/736 [==============================] - 19s 26ms/step - loss: 0.1659 - accuracy: 0.9578 - val_loss: 0.1046 - val_accuracy: 0.9745
```



```
#Accuracy with the test data
print('Test Data accuracy: ',accuracy_score(test_labels, pred)*100)
```

```
Test Data accuracy:  91.93190815518606
```

For model 2, following images are the part of training and test part with batch size=64, test size = 0,4:



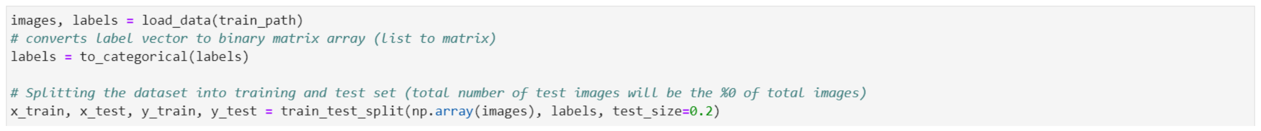
```
## Fitting the model - training process
# Epoch : number of passes of the entire training dataset
# Batch size : number of training examples utilized in one iteration. For example if you have x (1000) training data and batch
# size is y (10), model will take x/y (1000/10) example for one iteration

EPOCHS = 30
history = model.fit(x_train,
                    y_train,
                    validation_data = (x_test, y_test),
                    epochs=EPOCHS,
                    batch_size=64
                   )
```

```
Epoch 1/30
368/368 [==============================] - 20s 52ms/step - loss: 6.2359 - accuracy: 0.1123 - val_loss: 1.5676 - val_accuracy: 0.5359
Epoch 2/30
368/368 [==============================] - 17s 46ms/step - loss: 1.3462 - accuracy: 0.5976 - val_loss: 0.6066 - val_accuracy: 0.8200
Epoch 3/30
368/368 [==============================] - 17s 47ms/step - loss: 0.6680 - accuracy: 0.7953 - val_loss: 0.3440 - val_accuracy: 0.9118
```

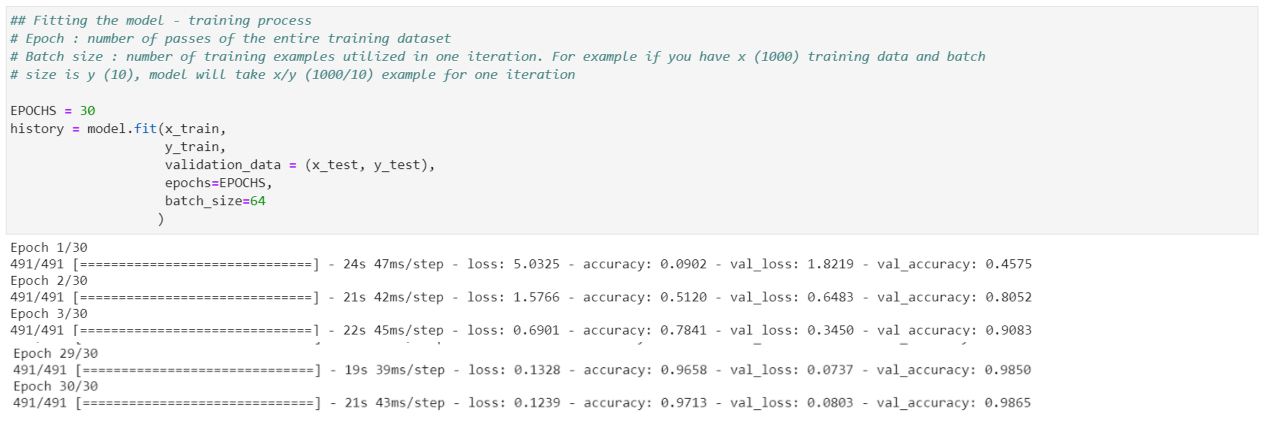
```
#Accuracy with the test data
print('Test Data accuracy: ',accuracy_score(test_labels, pred)*100)
```

```
Test Data accuracy:  92.62074425969912
```

For model 2, following images are the part of training and test part with batch size=64, test size = 0,2:

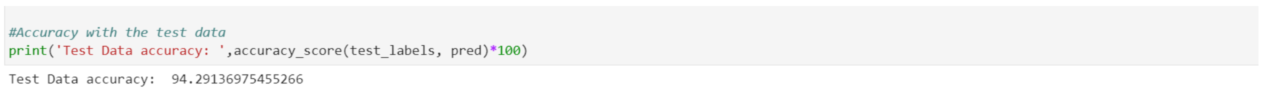


```
images, labels = load_data(train_path)
# converts label vector to binary matrix array (list to matrix)
labels = to_categorical(labels)

# Splitting the dataset into training and test set (total number of test images will be the %0 of total images)
x_train, x_test, y_train, y_test = train_test_split(np.array(images), labels, test_size=0.2)
```



```
## Fitting the model - training process
# Epoch : number of passes of the entire training dataset
# Batch size : number of training examples utilized in one iteration. For example if you have x (1000) training data and batch
# size is y (10), model will take x/y (1000/10) example for one iteration

EPOCHS = 30
history = model.fit(x_train,
                    y_train,
                    validation_data = (x_test, y_test),
                    epochs=EPOCHS,
                    batch_size=64
                   )
```

```
Epoch 1/30
491/491 [==============================] - 24s 47ms/step - loss: 5.0325 - accuracy: 0.0902 - val_loss: 1.8219 - val_accuracy: 0.4575
Epoch 2/30
491/491 [==============================] - 21s 42ms/step - loss: 1.5766 - accuracy: 0.5120 - val_loss: 0.6483 - val_accuracy: 0.8052
Epoch 3/30
491/491 [==============================] - 22s 45ms/step - loss: 0.6901 - accuracy: 0.7841 - val_loss: 0.3450 - val_accuracy: 0.9083
Epoch 29/30
491/491 [==============================] - 19s 39ms/step - loss: 0.1328 - accuracy: 0.9658 - val_loss: 0.0737 - val_accuracy: 0.9850
Epoch 30/30
491/491 [==============================] - 21s 43ms/step - loss: 0.1239 - accuracy: 0.9713 - val_loss: 0.0803 - val_accuracy: 0.9865
```



```
#Accuracy with the test data
print('Test Data accuracy: ',accuracy_score(test_labels, pred)*100)
```

```
Test Data accuracy:  94.29136975455266
```

On conclusion,

- ✓ I see that bigger training set affects the test data accuracy.
- ✓ Compared to %30 of test data, %20 gave better result and increasing batch size gave better result.

3.3. Output Examples

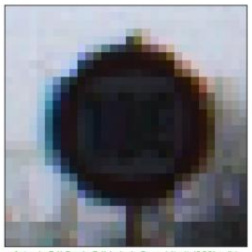
Actual=21 || Pred=21 || Label=Double curve

Actual=33 || Pred=33 || Label=Turn right ahead

Actual=25 || Pred=31 || Label=Wild animals crossing

Actual=13 || Pred=13 || Label=Yield

Actual=25 || Pred=25 || Label=Road work


Actual=7 || Pred=7 || Label=Speed limit (100km/h)

Actual=15 || Pred=15 || Label=No vehicles

Actual=9 || Pred=9 || Label=No passing

Actual=9 || Pred=9 || Label=No passing

Actual=16 || Pred=16 || Label=Vehicles over 3.5 metric tons prohibited

Actual=5 || Pred=5 || Label=Speed limit (80km/h)

Actual=28 || Pred=25 || Label=Road work

Actual=38 || Pred=38 || Label=Keep right

Actual=10 || Pred=10 || Label=No passing for vehicles over 3.5 metric tons

Actual=11 || Pred=11 || Label=Right-of-way at the next intersection

Actual=25 || Pred=25 || Label=Road work

4.0. References

[1] R. Alake, 2020, “Implementing AlexNet CNN Architecture Using TensorFlow 2.0+ and Keras”, Towards Data Science, https://towardsdatascience.com/implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98

[2] Tensorflow Team, 2020, “Module: tf.keras.losses”, Tensorflow Org, https://www.tensorflow.org/api_docs/python/tf/keras/losses

[3] J. Collis, 2017, “Glossary of Deep Learning: Batch Normalisation”, Medium, https://medium.com/deeper-learning/glossary-of-deep-learning-batch-normalisation-8266dcd2fa82

[4] S. Nayak, 2018, “Understanding AlexNet”, Learn OpenCv, https://www.learnopencv.com/understanding-alexnet/

[5] T. Tracey, 2019, “Recognizing Traffic Signs with CNNs”, Medium, https://medium.com/@thomastracey/recognizing-traffic-signs-with-cnns-23a4ac66f7a7

[6] A. Rosebrock, 2018, “Keras Conv2D and Convolutional Layers”, Py Image Search, https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

[7] S. Vasudevan, 2020, “AlexNet-CNN Explained and Implemented”, YouTube, https://www.youtube.com/watch?v=8GheVe2UmUM

[8] Nachiket, 2019, “Build a Traffic Sign Recognition Classifier”, GitHub, https://github.com/nachiket273/Self_Driving_Car/tree/master/CarND-Traffic-Sign-Classifier-Project

[9] Mykola, 2019, “GTSRB-German Traffic Sign Recognition Benchmark Multi-class, single-image classification challenge”, Kaggle, https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

[10] K. Shah, 2020, “Traffic Signal Recognizer”, Kaggle, https://www.kaggle.com/kar0n7/traffic-signal-recognizer-98-accuracy

[11] Data Flair Team, 2019, “Python Project on Traffic Signs Recognition with 95% Accuracy using CNN & Keras”, Data Flair, https://data-flair.training/blogs/python-project-traffic-signs-recognition/