MACHINE LEARNING ENGINEER NANODEGREE

Capstone Project

Anuja Tupe

October 26th, 2018

German Traffic Sign Classification Using Keras

I. DEFINITION

PROJECT OVERVIEW

Self Driving Cars are disrupting the automobile and transportation industry. It is an active ongoing research field. Lot of companies like Google, Uber, Volkswagen etc. are working in this field. The area of study for self driving cars is broadly applicable to several industries like basic computer science, computer vision, mechanical engineering, machine learning, robotics. Self driving cars pose many real world problems that can be solved using computer vision and machine learning. As I work in the silicon valley, solving these problems can have a real impact in this field.

Following is one of the links to an academic paper where machine learning was applied to solve this problem - http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf

PROBLEM STATEMENT

Traffic signs are one of the most essential component for regulating traffic. They assist people in navigating the vehicle with discipline thus avoiding any chaos or accidents. One such problem that I am interested in solving is classifying traffic signs. If traffic signs are not correctly classified by driverless cars, it can pose a safety threat to people in and around the driverless car. It can lead to accidents and possibly can be fatal. For example - If the traffic sign for 'STOP' is misinterpreted as something else, the car won't stop and can lead to an accident.

METRICS

Metrics are a way to measure the performance of the machine learning model that you built. There are various kinds of metrics available. However, not all are applicable to all kinds of problems. For example - accuracy cannot always be used as a metric. If the class is highly imbalanced, accuracy can be the worst way to check if your model is performing better.

Since, our dataset is imbalanced, we are using F-beta score as the evaluation metrics. F-beta score is the weighted harmonic mean of precision and recall. The value of beta that we will be using is 1.0. This means that precision and recall are equally important for the model. Precision is the exactness or quality of the model and recall is the completeness of the model. Precision is true positives divided by the sum of true positives and false positives. Recall is number of true positives divided by the sum of true positives and false negatives.

Here are some of the definitions -

Precision is the fraction of relevant instances among the retrieved instances. It is number of true positives divided by the sum of true positives and false positives.

$$\text{Precision} = \frac{tp}{tp + fp}$$

Recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. It is number of true positives divided by the sum of true positives and false negatives.

$$\text{Recall} = \frac{tp}{tp + fn}$$

F-beta score is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst value at 0. The beta parameter determines the weight of precision in the combined score. beta < 1 lends more weight to precision, while beta > 1 favors recall (beta -> 0 considers only precision, beta -> inf only recall).

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$

Classification report is a text report of the main classification metrics. It is basically a summary of the precision, recall, F1 score for each class. We use the classification_report method from sklearn for our models.

Confusion matrix is another metric that we used to summarize the performance of the classification algorithm. It helps in visualizing the performance of our classification model. The rows represent the instances in a predicted class while column represents the instances in an actual class. However, since, the number of labels are high (43 class IDs) it can be a little cumbersome to present the confusion matrix and make sense of it. We have plotted the confusion matrix for our final model using matplotlib and seaborn. Matplotlib is a 2D plotting library which generate plots, histograms, power spectra, bar charts, error charts, scatterplots etc. Seaborn is a data visualization library which is based on matplotlib which is used for drawing statistical graphs.
We can see the misclassified classes using confusion matrix.

Additionally, we have picked up random images from the web and fed it to the model. The model will predict it's label and we can see what is the result. We have picked 16 images from web and used it to test our model.

II. ANALYSIS

DATA EXPLORATION

We have used the dataset provided by German Traffic Sign Recognition Benchmark. The dataset can be downloaded from here - http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset#Downloads.

This dataset includes 50000 images. The images are colored images and are provided in the ppm format. The dimensions of all the images are not the same. These images are split into train dataset and test dataset. There are 39209 images in the train dataset and 12630 images in the test dataset. There are 43 types of traffic signs that we will be training on. The images vary in lot of aspects - brightness, sizes, color. Each traffic sign image has a class ID (integer from 0 to 42) associated with it. This information is stored in the csv files.

The train dataset has folders for every type of traffic sign (43 folders). Each folder has the images of the respective traffic sign and a CSV file which maps the image name with a class ID.

In order to get meaningful information about what each traffic sign means, we will create a CSV file with the mapping between class IDs and class labels. We will use the CSV files from the 43 folders to create our class labels.

http://www.gettingaroundgermany.info/zeichen.shtml - We will use this website to get meaningful english names for each of the 43 traffic signs. For example - Class ID 1 is for traffic sign “Speed Limit 30”.

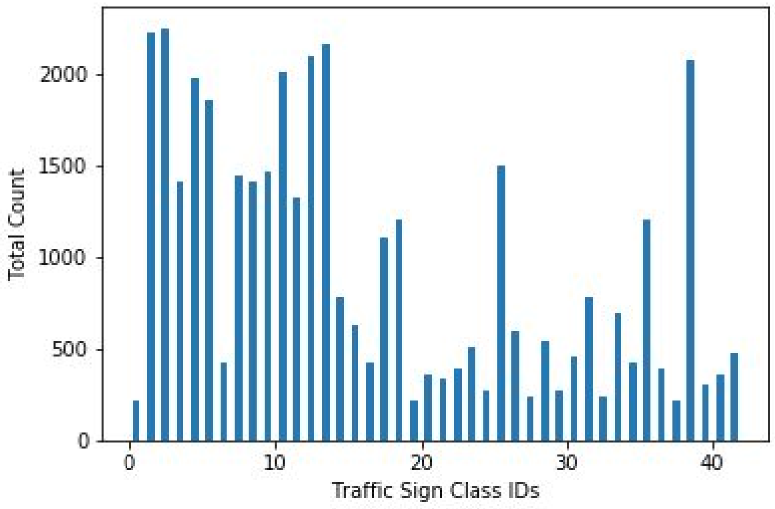
The test dataset has assorted images of traffic signs. It also has a CSV file which maps the image name with the correct class ID.

The images directory structure looks like this -

- GTSRB
    - Final_Training
        - Images
            - 00000
                - 00000_00000.ppm
                - ..
                - 00006_00029.ppm
                - GT-00000.csv
            - 00001
                - 00000_00000.ppm
                - ..
                - 00073_00029.ppm
                - GT-00001.csv
            - .
            - .
            - .
            -
            - 00042
                - 00000_00000.ppm
                - ..
                - 00007_00029.ppm
                - GT-00042.csv
            -
    - Final_Test
        - Images
            - 00000.ppm
            - ..
            - 00000.ppm
            - GT-final-test.csv
    - class_labels.csv

EXPLORATORY VISUALIZATION

The dataset that we have used is imbalanced. The plot below represents the frequency of the every traffic sign. The X axis represents the class IDs for the traffic sign images. The Y axis represents the total count of each of the traffic sign class ID.



Class 2 has the highest frequency and the frequency is 2250 and Class 0 has the lowest frequency and the frequency is 210.

http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset#Codesnippets - We used the sample code snippet provided by the GTSRB website to read the images. I have modified this code to preprocess the image after we read the image. We have stored this image in an array.

ALGORITHM AND TECHNIQUES

The traffic sign recognition problem is a classification problem. Since, it involves images, Convolutional Neural Network (CNN) is a great fit for solving this problem.

We have tried 3 models - a simple CNN model, Lenet-5 model and a modified version of the Lenet-5 model. The details of every model are explained latter section of the report.

To summarize our solution, we analysed the images and decided the preprocessing techniques that we would be using for the images. These preprocessed images are used for training the CNN model. Along with preprocessing the images, we also augment our images to increase the size of our dataset. This helps train our model better and it will also reduce overfitting. Finally, we tested it on the test dataset provided.

We also tested all the models we tried by picking some german traffic sign images from Internet. We print out the label predicted by our model for the test image.

BENCHMARK MODEL

In 2011, the IJCNN (International Joint Conference on Neural Networks) competition took place for traffic sign recognition. The IJCNN 2011 competition results can be a benchmark model.

http://benchmark.ini.rub.de/index.php?section=gtsrb&subsection=results&subsubsection=ijcnn - This link has the results of the competition. It has a table of teams that participated and their score with the highest score on top.

Additionally, we have tried out different CNN models and we could beat the performance every time we tried a new model. Please refer section for results in the project report below.

III. METHODOLOGY

DATA PREPROCESSING

There are a lot of preprocessing techniques available for images. We used the histogram equalization technique. If you notice the images in the dataset, they seem to be very dark and blurry. Histogram equalization is used for improving the contrast of the image. In order to do this, we first converted our image from RGB to HSV. We then used the equalize_hist() method of skimage to equalize each channel in the image.

Cropping an image is another preprocessing step that we executed. We cropped the image to be in the square shape. The center part of the image is the important part. Hence, we cropped the image to be a square image considering the center of the image. The logic to do this is - Select the smaller side of the image. For example - if the dimension of the image is 40x50, we selected the smaller side i.e. 40. We cropped the image to be 40x40. In order to consider the center of the image, we selected 20 on the left side of the center and 20 on the right side of the center. Similarly, we selected 20 on the top of the center and bottom of the center.

Lastly, we resized the images. All images are not of the same size. The dimensions of the images vary. For example - Say one image can be of size 60x60 and the other image can be of size 80x80. We will resize it to be of size 32x32.

IMPLEMENTATION

Data Split for Training and Validation

Since, our dataset is imbalanced we have to make sure the dataset split for training and validation is done in such a way that a considerable amount of images for each class ID is given for training the model. For example - In our case, Class 2 has the highest frequency and the frequency is 2250. Class 0 has the lowest frequency and the frequency is 210. During the test train split, it can happen that the train dataset has say only 10 samples of the class 0 images and has 1100 images of the class 2 images. This can lead to an inaccurate model, since the model is not trained well on the class 0 images. To solve this issue, we used the StratifiedShuffleSplit method provided by sklearn. This method helps us create a train validation set which is equally balanced in terms of the classes.

In addition, we also used class weights when we fit the model. When our model comes across a training image from the class which has less samples, our model will pay more attention when calculating the loss.
We calculated the class weights by using the compute_class_weights method of the sklearn.utils module.

Data Augmentation

There can be a lot of inconsistencies in images. For example - In one image, the traffic sign can be to the left in the image. In some other image, the traffic sign can be at an angle. There can be many such variations with respect to size, angles, contrast. Hence, it is a good idea to use data augmentation to train our model. In augmentation, we produce various versions of the same image and thereby increase the amount of information learnt by the model. This avoids the overfitting problem by training the model with different versions of the same image. Keras has an ability to generate variations of the images and we used that for augmenting our data. The ImageDataGenerator class from Keras creates batches of image data with real-time data augmentation and we have used this to augment images.

CNN Models

Since, our problem definition revolves around classifying images, we have used a neural network called Convolutional Neural Network (CNN). We have used Keras for building CNNs. Keras is a neural network API written in Python. We have tried out different CNNs for this problem statement and compared the performance of each.

Simple CNN Model -

We made a basic CNN model with some convolutional layers.

We increased the number of filters with every convolutional layer which in turn increases the depth of every convolutional layer. We alternated every convolutional layer with a max pooling layer. Pooling layers are used to reduce the spatial size of the representation to reduce the amount of parameters and computation.
I also added a global average pooling layer. This will reduce the 3D array to a vector without losing any information. The last layer is the output layer (dense layer) and it has 43 nodes (as there are 43 types of traffic signs).

Notice, that we do not use any data augmentation technique with this model. We even use the basic train_test_split method to split our dataset into test and validation sets.

The model does not have a decent performance.

Lenet Model -

This was one of the very first CNN which powered the field of neural networks. It was used to recognise handwritten digits and was initially used in Banks. I used the Lenet model architecture. We referred the following paper to understand the Lenet architecture - http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf.

The Lenet model has the following layers -

1. Convolution Layer 1 - The input shape for this layer is 32x32. This layer has 6 feature maps and each feature map has the dimension 5x5 with relu as the activation function. The output shape of this layer is (6, 28, 28)
2. Subsampling Layer 1 - This layer has 6 feature maps and has a pool size and stride of 2. The output shape of this layer is (6, 14, 14)
3. Convolution Layer 2 - This layer has 16 feature maps and each feature map has the dimension 5x5 with relu as the activation function. The output shape of this layer is (16, 10, 10)
4. Subsampling Layer 2 - This layer has 16 feature maps has a pool size and stride of 2. The output shape of this layer is (16, 5, 5)

5. Flatter layer 1 - is used to flatten 3D to 1D.
6. Fully Connected Layer 1 - has 120 feature maps with relu as the activation function
7. Fully Connected Layer 2 - has 84 feature maps with relu as the activation function
8. Fully Connected Layer 3 - has 43 feature maps with softmax as the activation function. This is the last layer (output layer). 43 feature maps correspond to the 43 types of traffic signs (43 classes).

We can use the following equations to calculate the output of the layers. We know input width and input height (32x32). We know the padding (P), the Stride (S), the filter width (F_w) and filter height (F_h). Using this we can find the output width and output height.

$$\text{output width} = \frac{W - F_w + 2P}{S_w} + 1$$

$$\text{output height} = \frac{H - F_h + 2P}{S_h} + 1$$

For example - This is how you calculate the output of the first convolution layer. Our input image has the dimensions 32x32, so W = 32 and H = 32, padding (P) = 0, Stride (S) = 1, filter width (F_w) = 5 and filter height (F_h) = 5. If we substitute all these values in the above two equations, we will get output height = 28 and output width = 28.

This is how you calculate the output shape for each layer. It works the same way for pooling layers too.

We have used SGD optimizer as it has better generalization than adaptive optimizers. We are using a decreasing learning rate with SGD. The learning rate decreases with every epoch. This helps us take smaller steps towards the solution. As we go closer to the solution, we want to take smaller steps towards it so that we do not jump over the solution.

Modified Lenet-5 model -

Each filter in CNN is a feature map. Increasing the filters helps in capturing more detailed features of the dataset. We changed the number of filters in the first convolutional layer and second convolutional layer. We doubled the number of filters in both the layers. We changed 6 filters to 12 filters in the first layer and 16 filters to 32 filters in the second layer.

We also added a dropout layer of 0.5 after the flatten layer. Dropout is a regularization technique. It helps avoid overfitting by ignoring randomly selected neurons when the model is getting trained. This helps in building a network which generalizes better.

We have used the same SGD optimizer as explained above.

PROJECT STRUCTURE

I used the AWS credits provided by Udacity to run my project on AWS EC2 Instance. The AMI that I chose for my project is - Deep Learning AMI with Source Code (CUDA 8, Ubuntu). I followed the instructions provided in Lesson 2 of the Deep Learning module of the Machine Learning Nanodegree program. This helped me setup my

EC2 instance where I could start my jupyter notebook and access it on my machine.

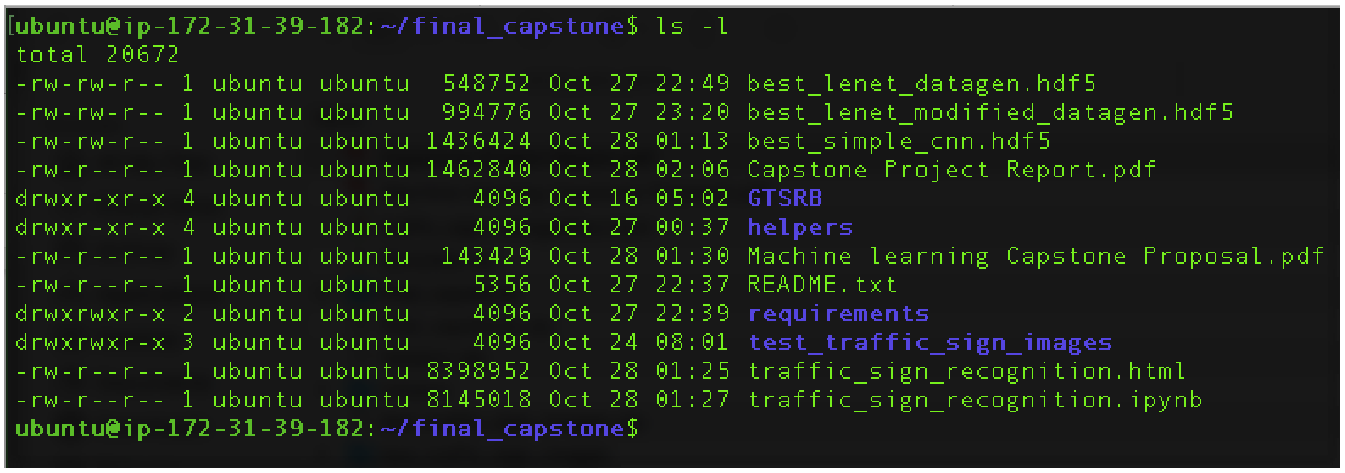
Classroom link for Lesson 2 of the Deep Learning module of the MLND program - https://classroom.udacity.com/nanodegrees/nd009t/parts/0ac87c1d-350a-417b-93c8-392dbf9cb8c2/modules/f5e5f204-ae46-415c-bd9b-a160eecf044f/lessons/29df00d8-01c2-4995-92fa-a4afd020be90/concepts/a20383fd-1e14-4311-824b-3d7981d99dee

Since, the dataset is big, we have mentioned instructions below on how to download it and place it in the project folder.

How to make the project structure?

1. Unzip final_capstone.zip. This will create a **final_capstone** folder.
    a. `unzip final_capstone.zip`
2. The project structure after unzipping should look like this -
3. Change directory to final_capstone folder
    a. `cd  final_capstone`
4. Download the training and testing dataset and arrange it using the below instructions
    a. Download the zip files for training
        i. wget http://benchmark.ini.rub.de/Dataset/GTSRB_Final_Training_Images.zip
    b. Download the zip files for testing
        i. wget http://benchmark.ini.rub.de/Dataset/GTSRB_Final_Test_Images.zip
    c. Unzip **GTSRB_Final_Training_Images.zip** - This will unzip contents in a folder named **GTSRB**
        i. `unzip GTSRB_Final_Training_Images.zip`
    d. Unzip **GTSRB_Final_Test_Images.zip -** This will unzip contents in a folder named **GTSRB**
        i. `unzip GTSRB_Final_Test_Images.zip`
5. Move the **class_labels.csv** file to **GTSRB** folder
    a. `mv class_labels.csv GTSRB/.`

So, the **final_capstone** folder should look like this -



```
[ubuntu@ip-172-31-39-182:~/final_capstone$ ls -l
total 20672
-rw-rw-r-- 1 ubuntu ubuntu  548752 Oct 27 22:49 best_lenet_datagen.hdf5
-rw-rw-r-- 1 ubuntu ubuntu  994776 Oct 27 23:20 best_lenet_modified_datagen.hdf5
-rw-rw-r-- 1 ubuntu ubuntu 1436424 Oct 28 01:13 best_simple_cnn.hdf5
-rw-r--r-- 1 ubuntu ubuntu 1462840 Oct 28 02:06 Capstone Project Report.pdf
drwxr-xr-x 4 ubuntu ubuntu    4096 Oct 16 05:02 GTSRB
drwxr-xr-x 4 ubuntu ubuntu    4096 Oct 27 00:37 helpers
-rw-r--r-- 1 ubuntu ubuntu  143429 Oct 28 01:30 Machine learning Capstone Proposal.pdf
-rw-r--r-- 1 ubuntu ubuntu    5356 Oct 27 22:37 README.txt
drwxrwxr-x 2 ubuntu ubuntu    4096 Oct 27 22:39 requirements
drwxrwxr-x 3 ubuntu ubuntu    4096 Oct 24 08:01 test_traffic_sign_images
-rw-r--r-- 1 ubuntu ubuntu 8398952 Oct 28 01:25 traffic_sign_recognition.html
-rw-r--r-- 1 ubuntu ubuntu 8145018 Oct 28 01:27 traffic_sign_recognition.ipynb
ubuntu@ip-172-31-39-182:~/final_capstone$
```

Exploring the project structure -

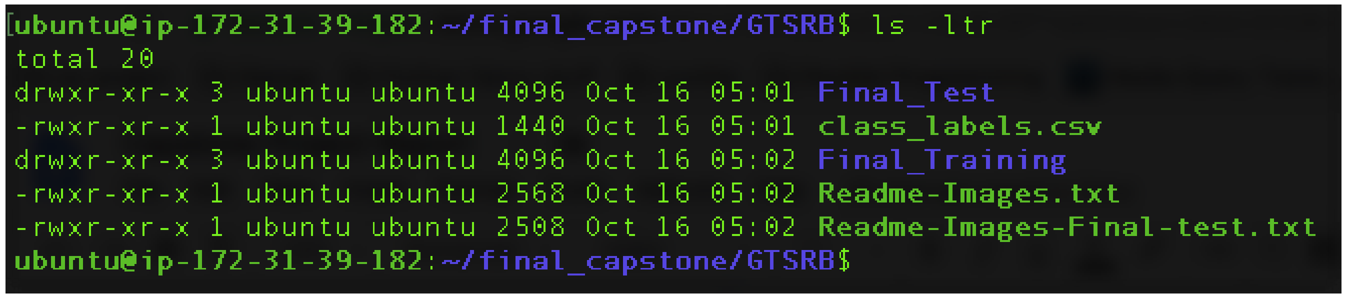
class_labels.csv -

This file was not part of the dataset that we downloaded. This CSV file has the mapping between the class IDs and class names (meaningful German traffic sign names). I created this file for showing the predicted class label for the random images from web.

GTSRB -

This folder has all the training and test images for the project. These images are downloaded from the http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

The contents of the **GTSRB** folder should look like this -



```
[ubuntu@ip-172-31-39-182:~/final_capstone/GTSRB$ ls -ltr
total 20
drwxr-xr-x 3 ubuntu ubuntu 4096 Oct 16 05:01 Final_Test
-rwxr-xr-x 1 ubuntu ubuntu 1440 Oct 16 05:01 class_labels.csv
drwxr-xr-x 3 ubuntu ubuntu 4096 Oct 16 05:02 Final_Training
-rwxr-xr-x 1 ubuntu ubuntu 2568 Oct 16 05:02 Readme-Images.txt
-rwxr-xr-x 1 ubuntu ubuntu 2508 Oct 16 05:02 Readme-Images-Final-test.txt
ubuntu@ip-172-31-39-182:~/final_capstone/GTSRB$
```

requirements -

This folder has the requirements-gpu.txt file. As per Lesson 2 of the Deep learning module, we copy pasted this file here. This file has the versions of the different libraries that need to be installed.

helpers -

This folder has all the python files. All the python code written as a part of this project are inside these file.

Following is the list of the python files. We will describe what each file does.

1. **classDetails.py** - This file has the method to show the details of the dataset. We print out the frequency of each class and we also print out a meaningful traffic sign name for each class. We plot the class IDs vs frequency plot. It shows the imbalance nature of our dataset.
2. **classLabels.py** - This file has the method to read the CSV file which has the mapping between class IDs and class labels.
3. **imageDetails.py** - This file is used to print the image details of any random image from train or test dataset. It prints details like the image array itself, the image shape, the number of rows and columns in the image array, plotting the image itself.
4. **lenetModel.py** - This file has the method to return Lenet-5 model.
5. modelPerformance.py - This file has the method to print out different model performance metrics. It shows the precision, recall and F-beta score of the mode. It also prints out the classification report of the model. We calculate the confusion matrix for the model and plot it.
6. **modifiedLenet.py** - This file has the method to return the modified Lenet-5 model.
7. **preprocessImage.py** - This file has all the methods to read the images from the training and testing folders and preprocess the images.
8. **simpleCNN.py** - This file has the method to return a simple CNN model.
9. **stratify.py** - This file has the method to return train and validation sets such that train and validation dataset are equally balanced in terms of the classes.

10. **testImages.py** - This file has the methods to predict the class label for a random image of german traffic sign from the internet.

traffic_sign_recognition.ipynb -

This is jupyter notebook which calls all our methods for the project.

traffic_sign_recognition.html -

This is the exported html version of the jupyter notebook.

test_traffic_sign_images -

This folder has the German traffic sign images from the web that we got for testing.

Capstone Project Report.pdf -

This is the project report PDF file.

Machine learning Capstone Proposal.pdf -

This is the project proposal PDF file.

REFINEMENT

This is how we kept refining the model that we used for our problem statement.

We started with a simple CNN model and we observed that it did not perform well. The validation loss did not improve. Hence, we stopped training the model after 15 epochs. It could correctly classify only 8 out of 16 images from the web.

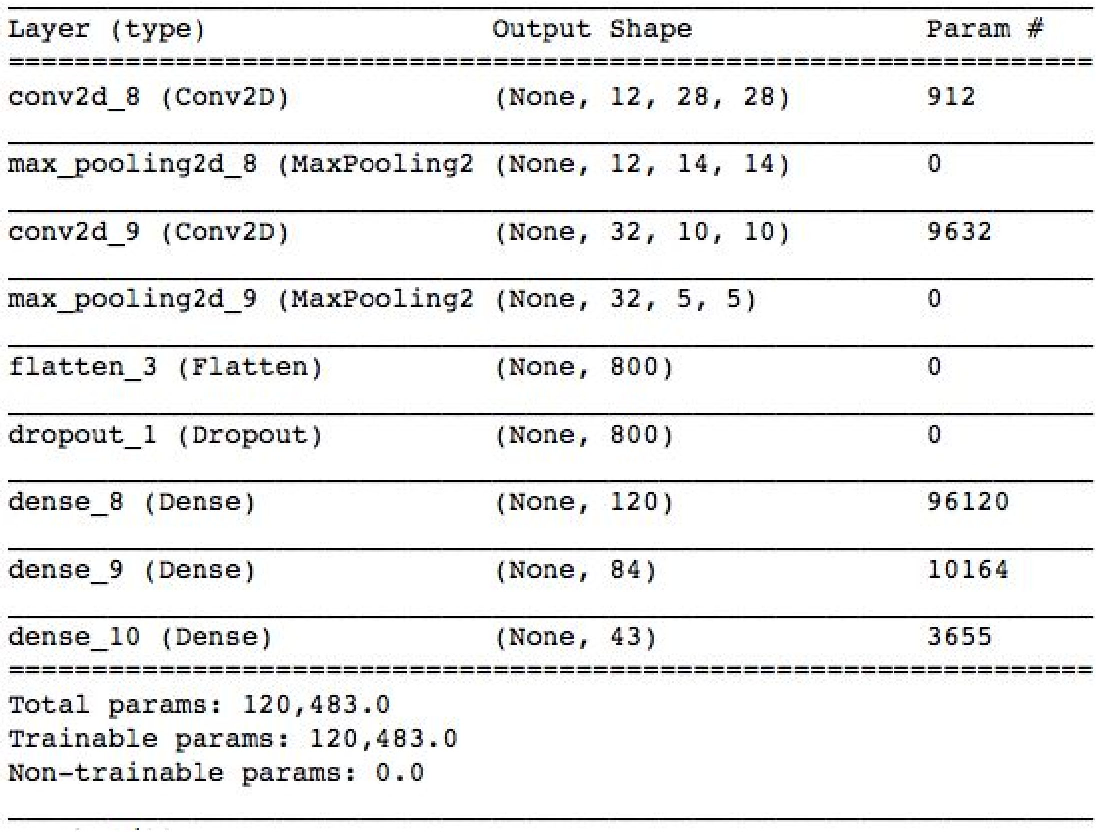
We know that Lenet-5 is a known model and it was used in Banks for recognizing handwritten digits. We used this model on our dataset. The validation loss kept improving and we stopped training at the 25th epoch. It considerably improved the performance of the model. However, we thought we could improve the performance of this model as it could correctly classify 14 out of 16 images from the web.

Hence, we modified this Lenet-5 model to add more number of filters and a dropout layer. The validation loss kept improving and we stopped training at the 30th epoch. This model performed slightly better on the test dataset. Also, it performed better on the images from the web. It could correctly classify 15 out of 16 images from the web.

IV. RESULTS

MODEL EVALUATION AND VALIDATION

Our final model's (**Lenet-5 modified model)** parameters are as given below -



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 12, 28, 28)        912
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 12, 14, 14)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 10, 10)        9632
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 32, 5, 5)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 800)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 800)               0
_________________________________________________________________
dense_8 (Dense)              (None, 120)               96120
_________________________________________________________________
dense_9 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_10 (Dense)             (None, 43)                3655
=================================================================
Total params: 120,483.0
Trainable params: 120,483.0
Non-trainable params: 0.0
_________________________________________________________________
```

There are a total of 120483 trainable parameters.

This model is better than the other models that we have tried (Lenet-5 and Simple CNN model).

We trained the model over 30 epochs. We stopped at 30 because we noticed that the validation loss did not improve after this. It can be seen in the loss vs epochs plot that we plotted.

We are using the SGD optimizer for this model with a learning rate of 0.01 and it keeps decreasing with every epoch.

Additionally, we also provide class weights as one of the parameters while fitting the model to help with the imbalance of dataset.

The batch size that we chose for our model is 32. Usually, a good default value for batch size is between 32 to 512 and most of the papers use a value between this range.

We use 80% of images for training and 20% of images for validation. The total number of images for training are 31360. The general way to find the value for steps per epoch is train_length // batch_size i.e. 31360//32 i.e. 980. However, since, I am using data augmentation, I increased the steps per epoch by 4 times. I multiply it enough and at the same time I do not hamper the speed of training the model. Hence, the value that we chose for steps per epoch parameter is (train_length // batch_size) * 4 i.e. (31360/32) * 4 = 3920.

JUSTIFICATION

We implemented 3 CNN models as part of the solution. The best model amongst them was the modified Lenet-5 Model.

The **simple CNN** model does not have good performance and the metrics are as follows -
precision: 0.6523313976657656
recall: 0.6697545526524149
fscore: 0.6280193204371762

The **Lenet-5 model** has a comparatively better performance and the metrics are as follows -
precision: 0.9462952742817614
recall: 0.9426761678543151
fscore: 0.9426501460331946

The **modified Lenet-5 model** has a little better performance than the Lenet-5 model and the metrics are as follows -
precision: 0.9623634471864845
recall: 0.9598574821852731
fscore: 0.9596077460914391

Confusion Matrix Results -
Confusion matrix is a good and a detailed way of knowing which classes are getting misclassified.

Since, the number of classes is high for our dataset, it is tedious to see which classes are misclassified the most. Just for giving an idea, we have displayed the number of times a class was misclassified for the Lenet-5 modified model. We did this for a few classes which have a high number of misclassified images. We have not shown the misclassification number for every class.

Lenet-5 modified Model -
Some of the classes that are misclassified are mentioned below -

Class ID which is misclassfied	Number of times it is misclassified
Class 1	24
Class 5	55
Class 14	29
Class 20	42

Web Images Results -
Also, the modified Lenet-5 model performed better on the images from the web than the basic Lenet-5 model and simple CNN model. The table below shows the number of correctly classified web images for all 3 models.

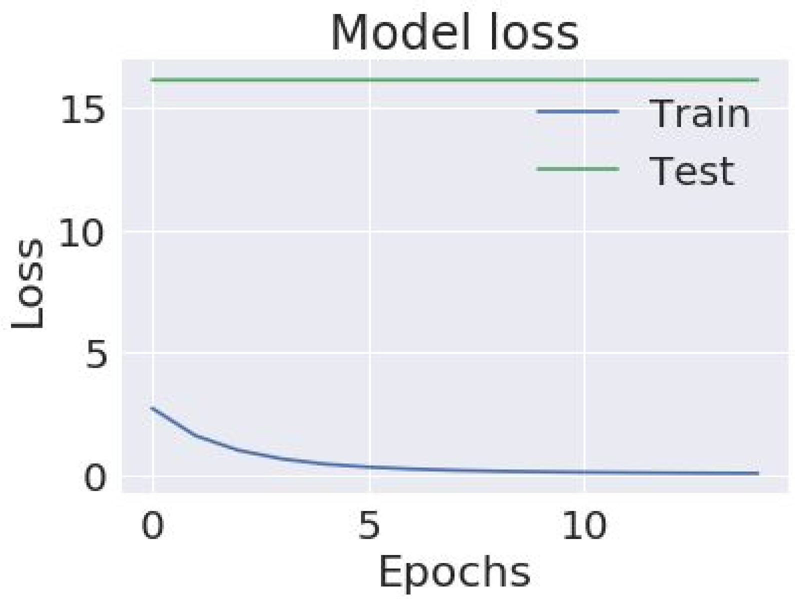
Model	**Correctly Classified Images**	**Percentage of correctly classified Images**
Simple CNN model	8 out of 16	50.00%
Lenet-5 model	14 out of 16	87.50%
Modified Lenet-5 Model	15 out of 16	93.75%

V. CONCLUSIONS

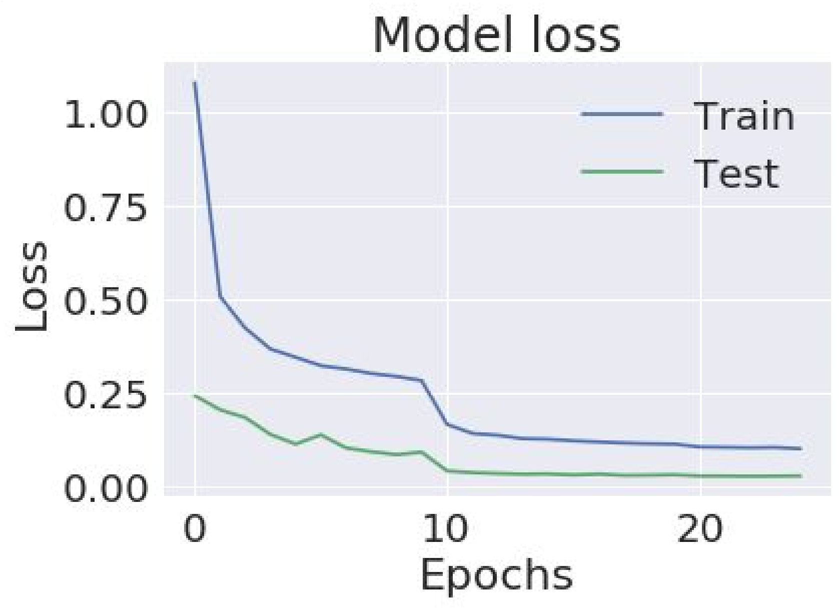
FREE-FORM VISUALIZATION

We plotted training and validation loss vs number of epochs. The validation loss for the simple CNN model does not improve at all. However, you can see the validation loss for Lenet-5 modified model is slightly better than Lenet-5 model.

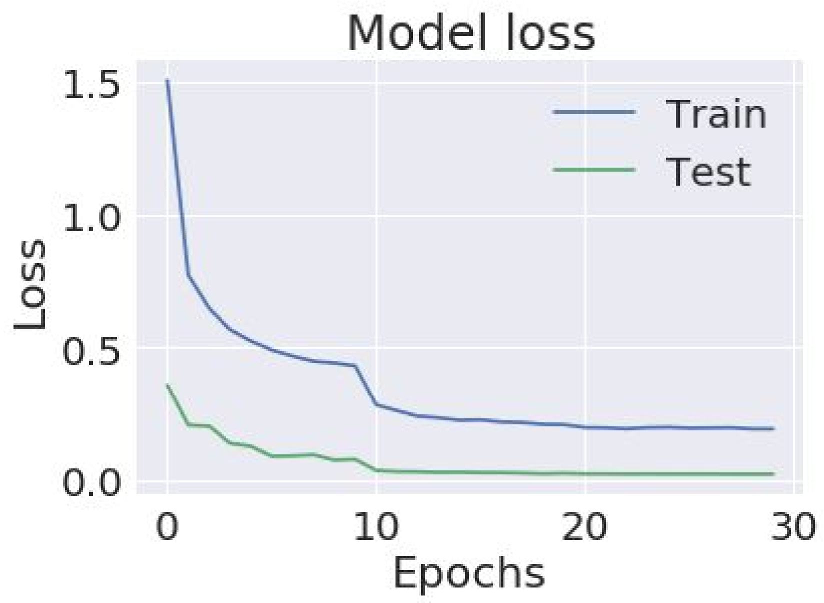
Plot for Simple CNN model -



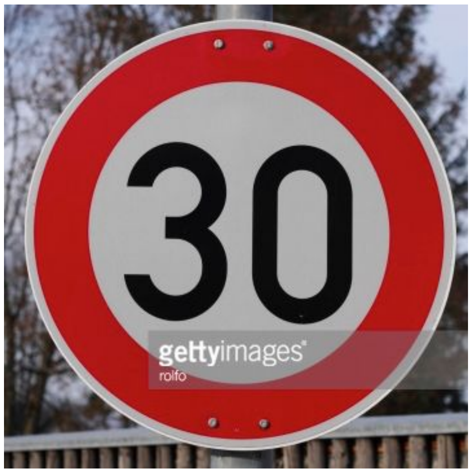
Plot for Lenet-5 model -



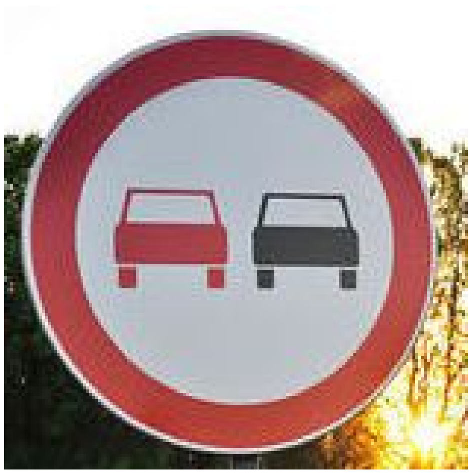
Plot for Lenet-5 modified model -



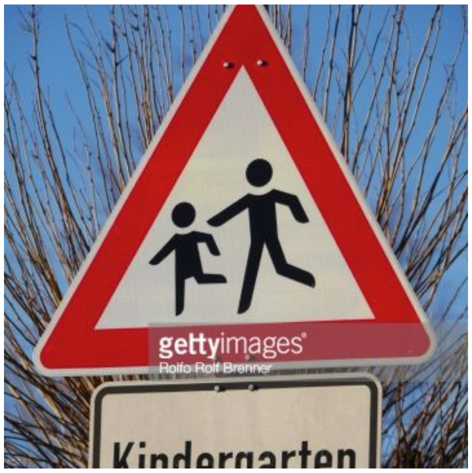
Below are the results of our final model (Lenet-5 modified model). These are the results are of the model predicting which traffic sign is in the image. You can see that only 1 of them of the images is wrongly classified by our final model (Lenet-5 modified model). It could correctly classify 15 out of 16 images (93.75% of images are classified correctly)



The predicted traffic sign is ”Speed Limit 30”



The predicted traffic sign is ”No Passing”

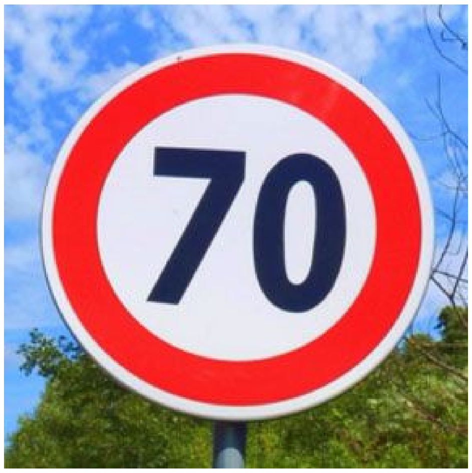


The predicted traffic sign is ”Watch for Children”

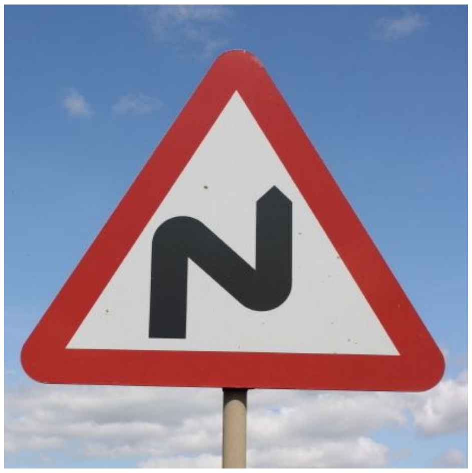
The predicted traffic sign is ”Do Not Enter”

The predicted traffic sign is ”Priority Road”

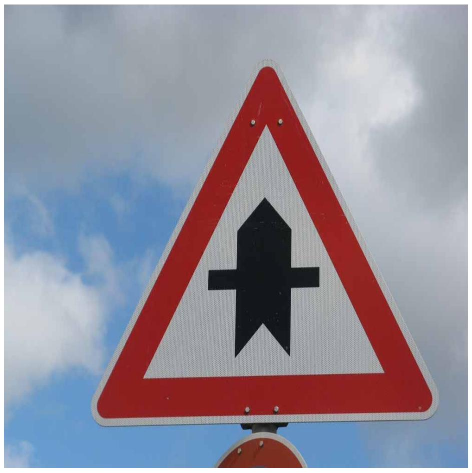
The predicted traffic sign is ”Do Not Enter”



The predicted traffic sign is ”Speed Limit 70”



The predicted traffic sign is ”Watch for Children”

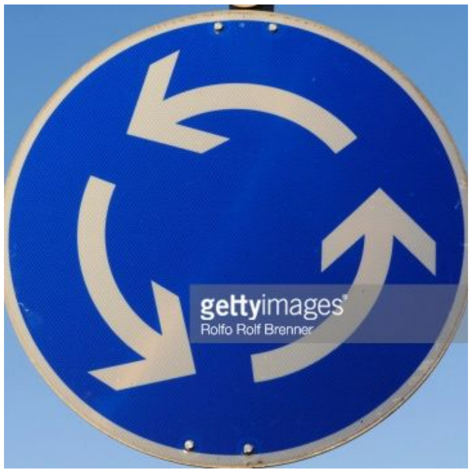


The predicted traffic sign is ”Priority”

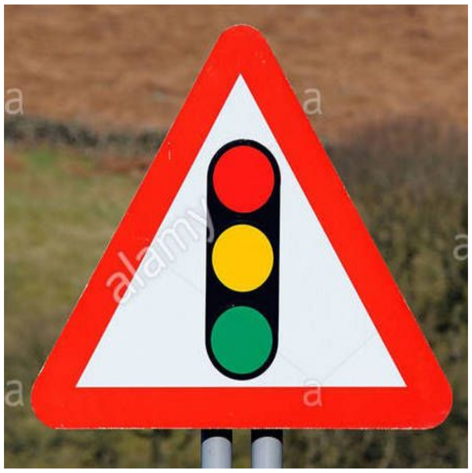
The predicted traffic sign is ”Pass by on right”

The predicted traffic sign is ”Road Work”

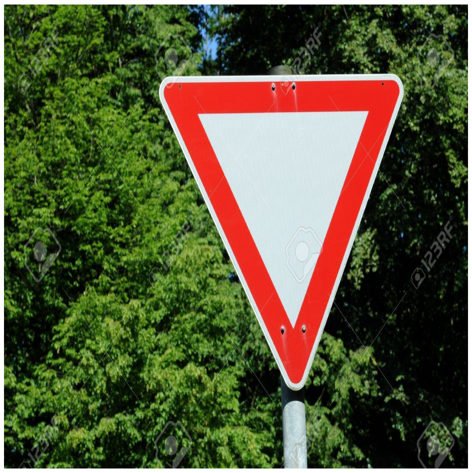
Es wird Zeit, die Probleme in Europa wieder zu lösen, statt sie auszusitzen.
SPD
CDU
The predicted traffic sign is ”Roundabout”



The predicted traffic sign is ”Roundabout”



The predicted traffic sign is ”Traffic signals ahead”



The predicted traffic sign is ”Yield”

The predicted traffic sign is ”Yield”

REFLECTION

To summarize our solution, we analysed the images and decided the preprocessing techniques that we would be using for the images. These preprocessed images are used for training the Lenet-5 modified model. Along with preprocessing the images, we also augment our images to increase the size of our dataset. This helps train our model better and it will also reduce overfitting. Finally, we tested it on the test dataset provided. Also, we tested it on the traffic sign images from the web.

One aspect of the project that I found difficult was to deal with the imbalance nature of the dataset. In day to day world, we will very rarely get a dataset which is balanced. So, it is an important problem to solve. We cannot remove images from the classes which have more frequency because that would hamper the performance of the model and it would underfit. A good way to solve it would be to add more images for the classes which have less images. However, this is a time consuming task. We used data augmentation to deal with the imbalanced nature of the dataset.

This imbalance nature of the dataset affects lot of other aspects of the project too. **For example** -

1. Accuracy cannot be simply used as one of the metrics. We have to use different metrics like F-beta score (https://towardsdatascience.com/accuracy-paradox-897a69e2dd9b)
2. We cannot use the regular train_test_split method to split our dataset into train and validation sets as it does not take into consideration the imbalance nature of the dataset.
3. We also use class weights when we fit the model. So, when our model comes across a training image from the class which has less samples, our model will pay more attention when calculating the loss.

Since, I had never dealt with an imbalanced dataset in any of the projects provided by Udacity, it was kind of difficult for me to start with it. However, eventually I learnt about the techniques to deal with it.

IMPROVEMENT

We can try out the following improvements in the solution.

1. Since, there is a imbalance in the dataset, we could refine our solution by getting more images for classes which have less images.
2. Preprocessing of images can also be one of the areas that can be experimented with. We can find

different ways to preprocess the images. For example - we can try converting the images into grayscale.
3. We could refine our model by
    a. Adding more dropout layers
    b. Experimenting with the number of filters
    c. Trying out different model altogether for solving the problem

Future scope - This project can also be extended to detect a traffic sign in an image. Right now, our project revolves around recognizing which traffic sign is the image. We can extend our project to actually locate where the traffic sign is in the image or video and then recognize which traffic sign it is.

VI. REFERENCES

1. http://benchmark.ini.rub.de/?section=gtsrb&subsection=news
2. http://www.gettingaroundgermany.info/zeichen.shtml
3. https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/
4. https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1
5. https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258
6. https://stackoverflow.com/questions/47117179/histogram-equalization-skimage?rq=1
7. https://chsasank.github.io/keras-tutorial.html
8. http://jokla.me/robotics/traffic-signs/
9. https://www.quora.com/Why-are-CNNs-used-more-for-computer-vision-tasks-than-other-tasks
10. https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/
11. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
12. https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
13. https://stackoverflow.com/questions/43237124/role-of-flatten-in-keras
14. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
15. https://stats.stackexchange.com/questions/201129/training-loss-goes-down-and-up-again-what-is-happening
16. https://medium.com/@thongonary/how-to-compute-f1-score-for-each-epoch-in-keras-a1acd17715a2
17. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html
18. https://stackoverflow.com/questions/29438265/stratified-train-test-split-in-scikit-learn
19. https://www.analyticsvidhya.com/blog/2018/03/essentials-of-deep-learning-visualizing-convolutional-neural-networks/
20. https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/
21. https://www.kaggle.com/danbrice/keras-plot-history-full-report-and-grid-search
22. https://alexisbcook.github.io/2017/using-transfer-learning-to-classify-images-with-keras/
23. https://www.svds.com/learning-imbalanced-classes/
24. http://www.deepideas.net/unbalanced-classes-machine-learning/
25. https://www.kaggle.com/eikedehling/exploring-class-imbalance-resampling-and-weights
26. https://www.jeremyjordan.me/imbalanced-data/
27. https://datascience.stackexchange.com/questions/13490/how-to-set-class-weights-for-imbalanced-classes-in-keras
28. http://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html
29. https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/
30. https://datascience.stackexchange.com/questions/23213/why-does-decreasing-the-sgd-learning-rate-cause-a-massive-increase-in-accuracy

31. https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10
32. https://github.com/scikit-image/scikit-image/issues/543
33. https://www.quora.com/How-can-I-calculate-the-size-of-output-of-convolutional-layer
34. https://matplotlib.org/
35. https://seaborn.pydata.org/
36. https://keras.io/preprocessing/image/#imagedatagenerator-class
37. https://en.wikipedia.org/wiki/Confusion_matrix
38. https://stackoverflow.com/questions/44193270/how-to-calculate-the-output-size-after-convolving-and-pooling-to-the-input-image
39. https://stackoverflow.com/questions/45538980/how-does-the-size-of-the-patch-kernel-impact-the-result-of-a-convnet
40. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
41. https://en.wikipedia.org/wiki/Precision_and_recall
42. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.fbeta_score.html
43. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/
44. https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8
45. http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf.
46. https://towardsdatascience.com/accuracy-paradox-897a69e2dd9b
47. https://stats.stackexchange.com/questions/164876/tradeoff-batch-size-vs-number-of-iterations-to-train-a-neural-network
48. https://www.quora.com/What-are-the-usual-batch-sizes-people-use-to-train-neural-nets
49. https://stackoverflow.com/questions/49922252/choosing-number-of-steps-per-epoch