German Road Sign Recognition

A multi-classification problem

Jonathan Greve

A competition in 2011

The German Traffic Sign Benchmark was a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011.

Traffic sign recognition is a multi-class classification problem with unbalanced class frequencies

The classifier has to cope with large variations in visual appearances due to illumination changes, partial occlusions, rotations, weather conditions, etc.

The problem: classifying 43 different road signs

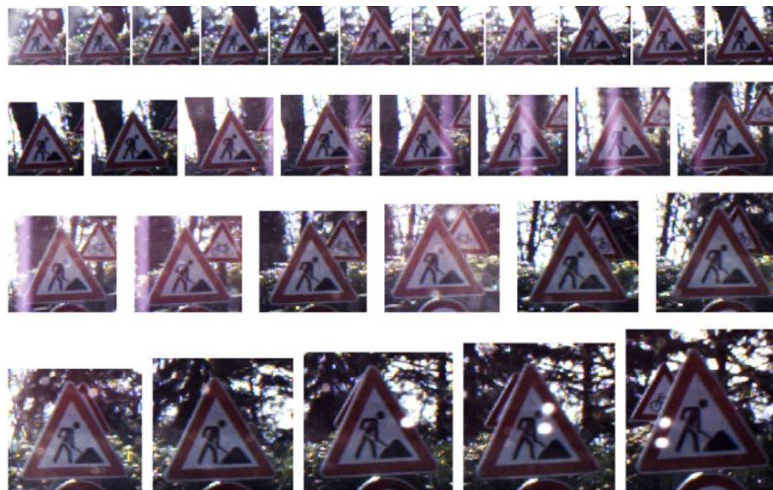
Datasets

- 39209 Train images /30 = 1306 tracks
- 12630 Test images, /30 = 421 tracks

A track is 30 images taken of the same sign. Depending on the speed of the car there might be 5 to 250 images of a sign. Only signs with at least 30 images are used in the dataset. If there are more than 30 images of a sign then 30 images are chosen equidistantly.

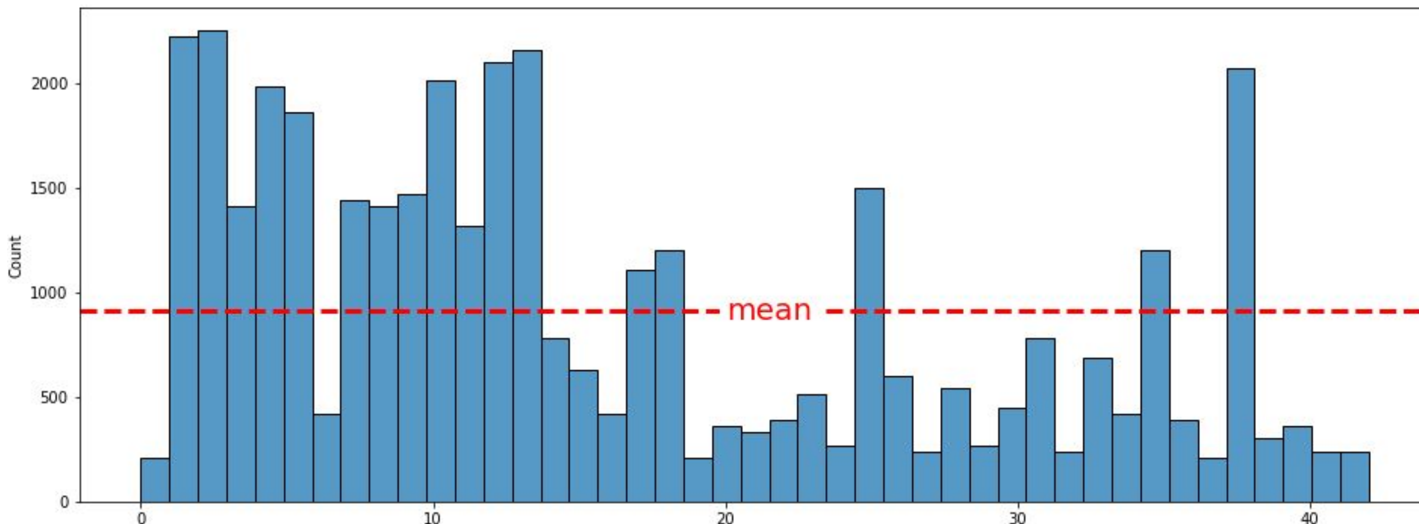
For example if there is 60 images of a sign then every other image is chosen for a total of 30.

A single track



Exploring the data

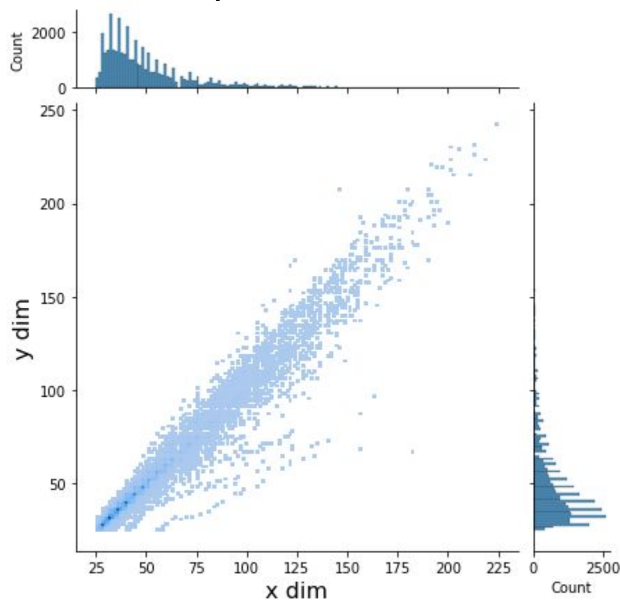
Class frequencies

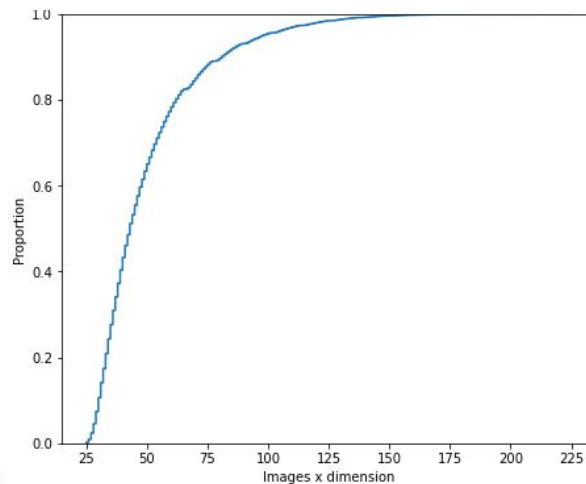


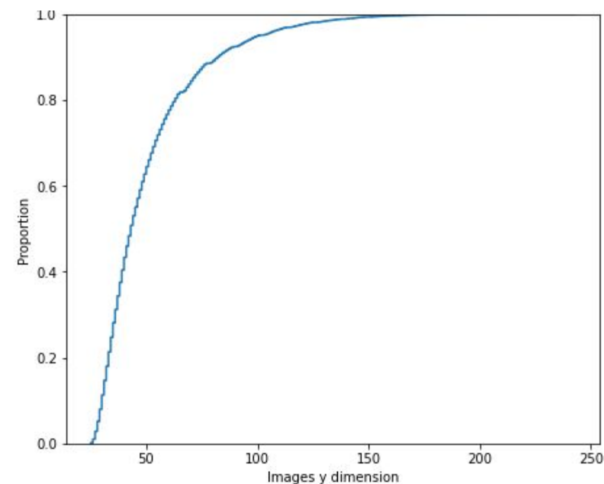
Unbalanced: care must be taken when splitting into train and validation.
Idea: Augment images to increase the size of the dataset.

Image dimensions

- Largest image in the training dataset: **(225, 243)**
- Smallest image in the training dataset: **(25, 25)**
- Average image dimensions in training dataset: **(50.3, 50.8)**







My solutions

1. CNN(s) with many image augmentations.
2. A LightGBM model with no augmentations.

Implementation of CNN

CNN - Preparing the data for training (splititng)

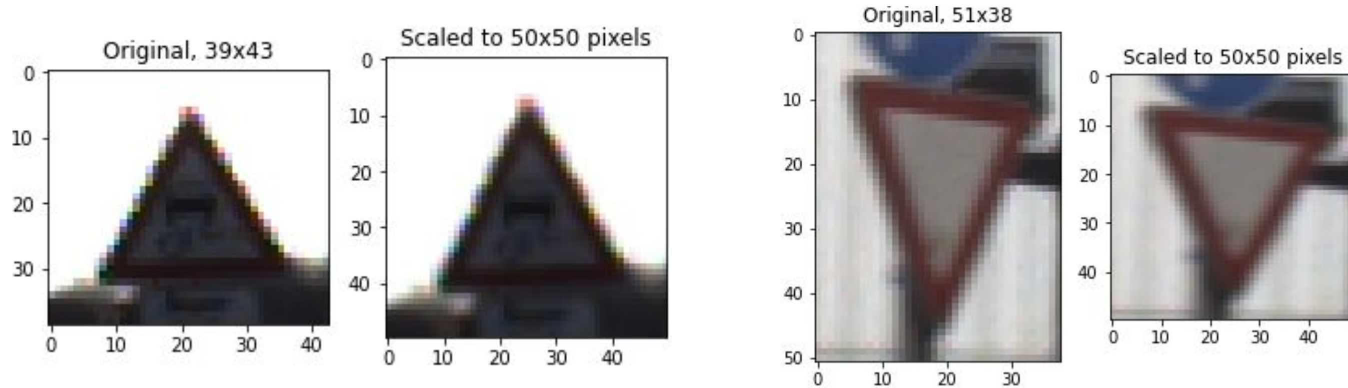
- For model selection using grid search I used a 80/20 train/validation split.

- Custom split function required due to the nature of the data

- I implemented a stratified split function because of the unbalanced data

CNN - Preparing the data for training (preprocessing)

- First I resize all images to the average image dimensions: **50x50 pixels**



- Then the images are converted to a Tensorflow dataset which will make augmenting images easier and faster during training.
- Pixel values are converted from integers to floats.

CNN - Augmenting the images

For every batch every image is augmented in the following order:

1) Randomly cropped to 48x48 pixels (i.e. 2 pixels in each dimensions).
2) If the image remains symmetric across an axis it is randomly flipped on that axis.
3) The brightness is randomly adjusted.
4) The contrast is randomly adjusted.
5) The saturation is randomly adjusted.
6) The hue is randomly adjusted.
7) Randomly rotated between -27 to 27 degrees
8) Randomly translated 0-3 pixels up and down on each axis
9) Randomly zoomed between 0 - 20%

Only for the training images of course.

CNN - Augmenting examples

Before

After

CNN - Model selection using grid search

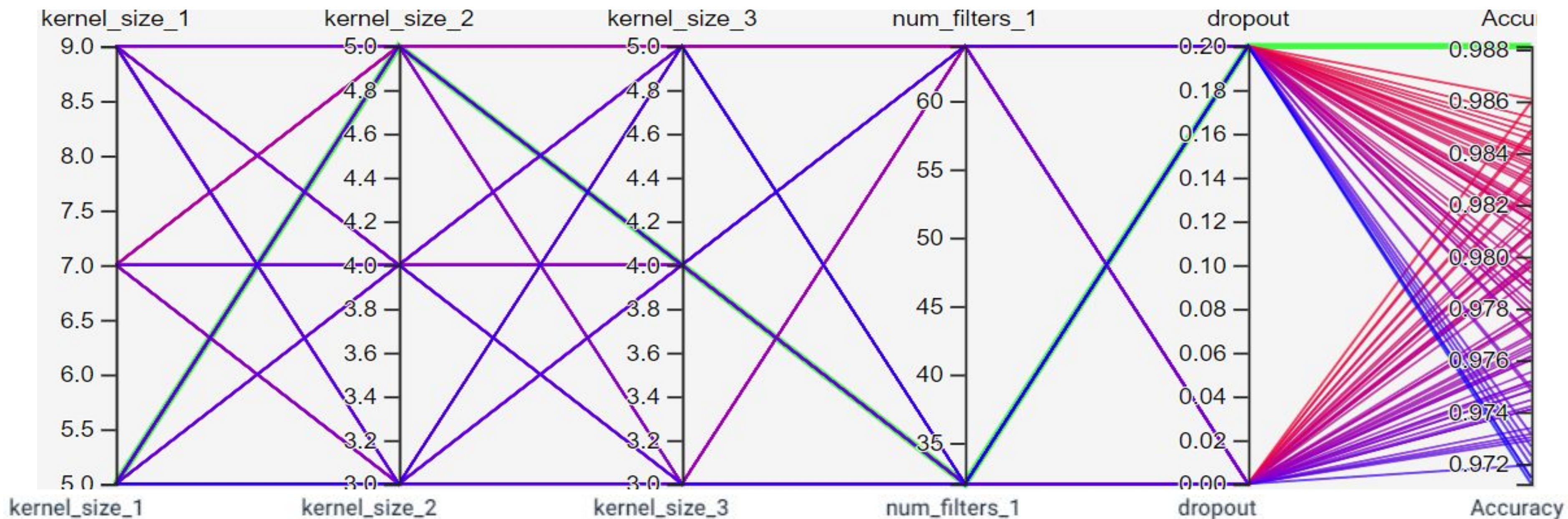
I optimized for the following parameters:

1. The kernel size of the three convolution layers. **3*3*3**
2. The number of filters for each convolution layer. **2**
3. With and without dropout. **2**

3*3*3*2*2 = 108 models in total.

It took about 12 hours for the grid search to complete.

CNN - Model selection using grid search



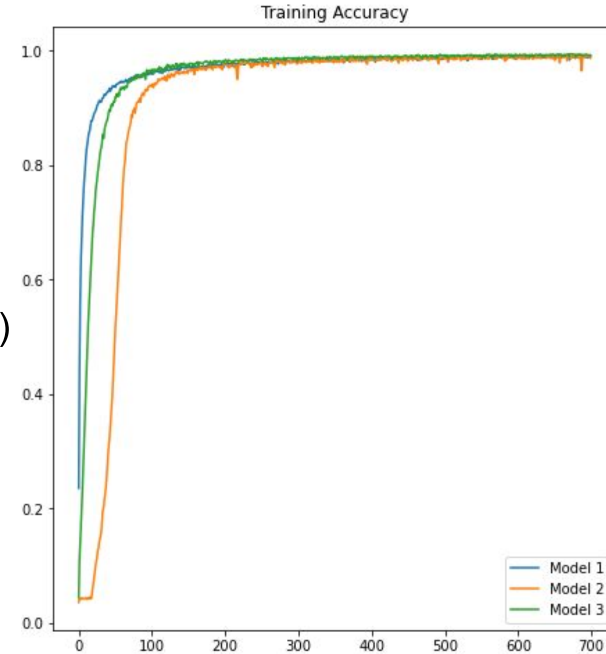
kernel_size_1	kernel_size_2	kernel_size_3	num_filters_1	dropout	Accuracy
5.0000	5.0000	4.0000	32.000	0.20000	0.98813
9.0000	4.0000	4.0000	64.000	0.20000	0.98609
5.0000	4.0000	5.0000	64.000	0.0000	0.98597

Committee CNN - The top 3 models are used

- The 3 models have the following layout:
  1. Conv2D layer filters=(32, 64, 64), kernel=(5x5, 9x9, 5x5)
  2. ELU activation
  3. MaxPooling2D pool_size=2x2
  4. Conv2D layer filters=(64, 128, 128), kernel=(5x5, 9x9, 5x5)
  5. ELU activation
  6. MaxPooling2D pool_size=2x2
  7. Conv2D layer filters=(128, 256, 256), kernel=(5x5, 9x9, 5x5)
  8. ELU activation
  9. Flatten
  10. Dropout
  11. Dense (softmax)

  For a total of 383083, 946603 and 1230763 trainable parameters respectively for each CNN.

- Each model is trained seperately and then the predictions for each model are averaged before using argmax to get the final predictions for each image.

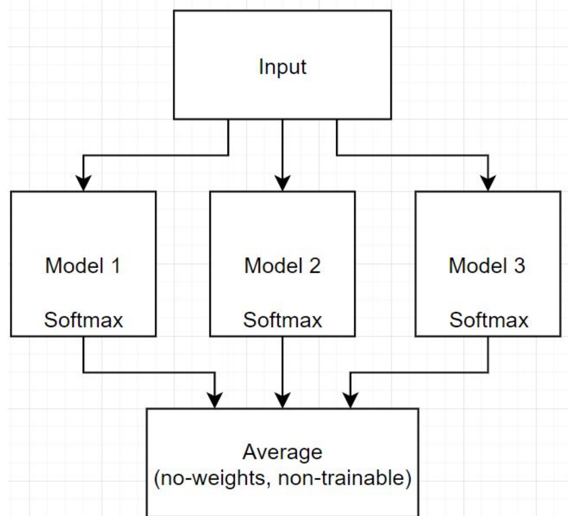


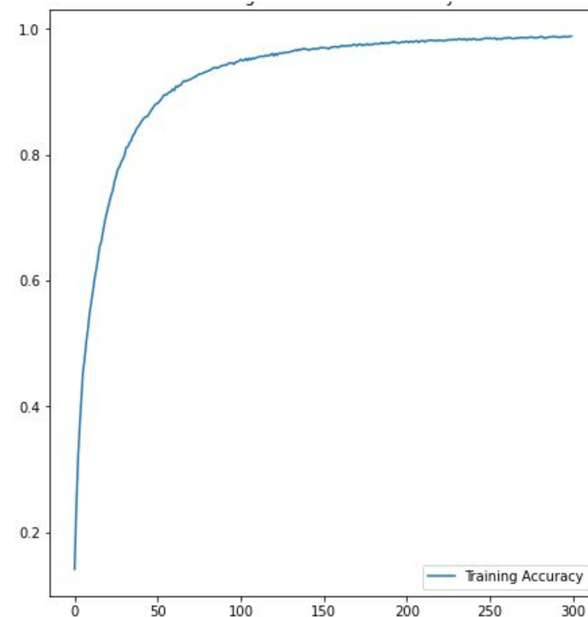
Ensemble CNN - The top 3 models are used

- The models are the same as the CNNs I just showed.
- Instead of training each model seperately we train them together and for each prediction during training we do the averaging.
- Thus we aren’t training 3 models but a single model based on 3 models.

The architecture looks like this:

It has 2560466 parameters in Total (just add the parameters in each layer).





Implementation of LightGBM

LightGBM - Preparing the data

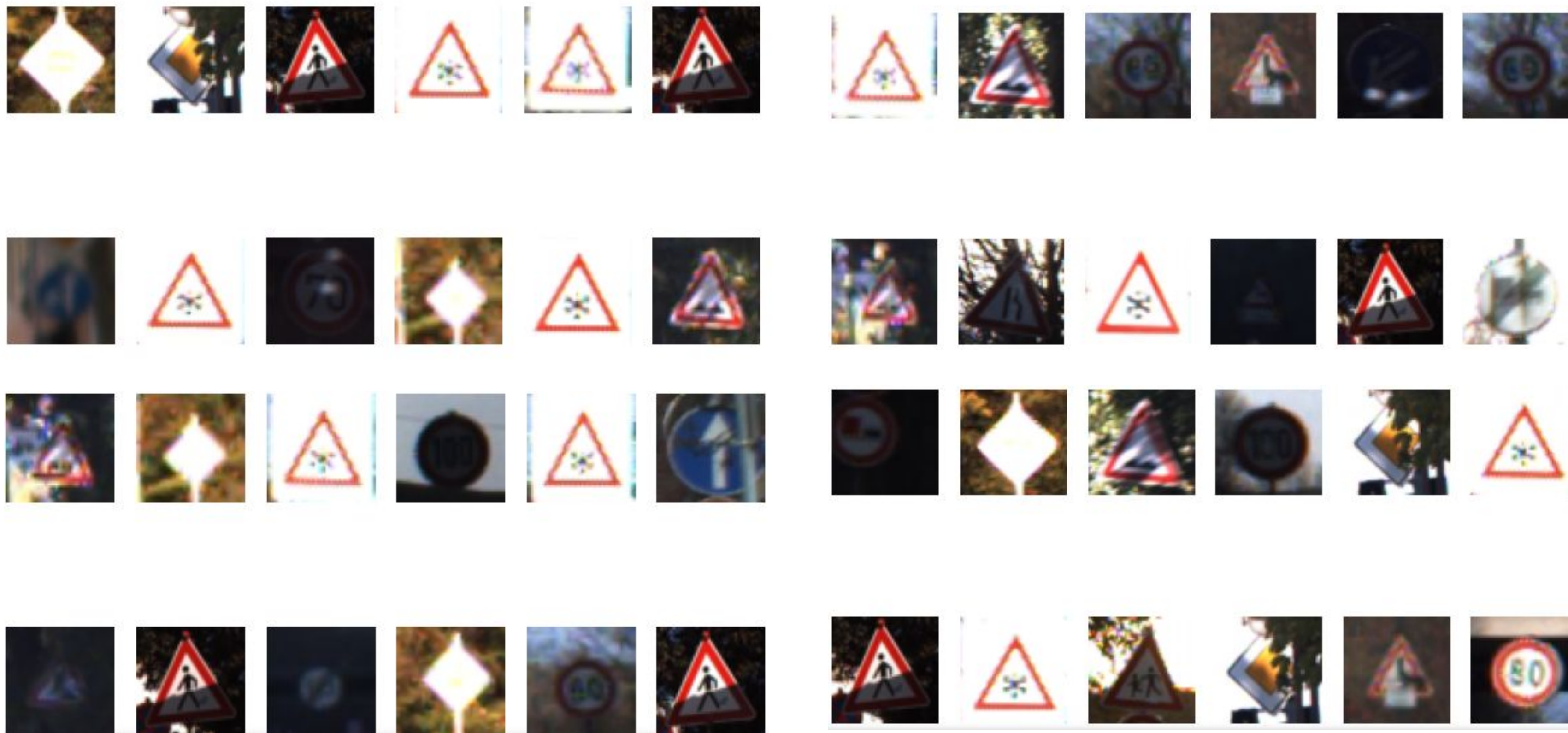
The data is preprocessed in the same way as for the CNN except:

- Images are resized to 40x40
- Unlike a CNN it requires a 1d array. So the images are reshaped to an 1d array of length: **40*40*3 = 4800**

Results

Team name	Algorithm	Accuracy %
ISDIA (1st place)	Committee of CNNs	99.46
Human	Best human	99.22
Human	Average human	98.84
Me	**Committee of CNNs**	**98.80**
Sermanet (2nd place)	CNN	98.31
Me	**Top 2 CNN - Grid search**	**98.08**
Me	**Top 1 CNN - Grid search**	**98.00**
Me	**Top 3 CNN - Grid search**	**97.21**
Me	**Ensemble of CNNs**	**95.50**
Me	**LightGBM**	**80.97**

CNN - Some misclassified images (Committee CNNs)



Discussion (1/2)

- The models were selected based on the validation data and then evaluated on the test data. After evaluating on the test data no more model tuning was done. This is to keep the evaluation as an unbiased estimation of the accuracy on unseen data.

- I choose not to optimize my LightGBM model further, such as augmenting images based on my initial results. The CNN was vastly superior, so I focused my efforts on improving it. In the 2011 competition a RF did alright with an accuracy of ~96.14%. So it should be possible to get better results with LightGBM with more feature engineering. But even a simple CNN model is better than the competitions RF model, so a tree based approach might not be optimal.

- The model was selected on the background of the grid search. However each model was only trained for 100 epochs in the grid search due to time limitations. It is possible a different model(s) would be chosen if more epochs were run. The validation error was also not stable and was fluctuating up and down slightly.

Discussion (2/2)

- The ensemble model which includes an averaging layer uses the output from all three CNNs when computing each CNN weight. I.e. if two CNNs predict the wrong class and one the correct class the wrong one is likely chosen and the weights in the correct CNN is affected negatively. It is also difficult to inspect if any of the internal models are underperforming when they are grouped like this.

- The committee model performed better than any of its' CNNs did individually. This is expected. The best model from the competition was also a committee of CNNs.

- Training the ensemble CNN required lowering the Adam optimizers learning rate by a factor of 10 compared to for a single CNN for it to converge. This means it also trained slower

Conclusion

- In this project we looked at a multi-classification problem with real world applications. Out model approached state of the art from 10 year ago. We presented multiple approaches to the learning problem.

- The models were selected based on the validation data and then evaluated on the test data. After evaluating on the test data no more model tuning was done.

- We discussed challenges faced such as how to scale images to be the same dimensions, how to handle the imbalanced data and possible solutions.

- We compared two different approached, LightGBM and CNNs. The CNNs quickly gave good results and our findings match those from the competition, i.e. that a committee of CNNs are very performant.

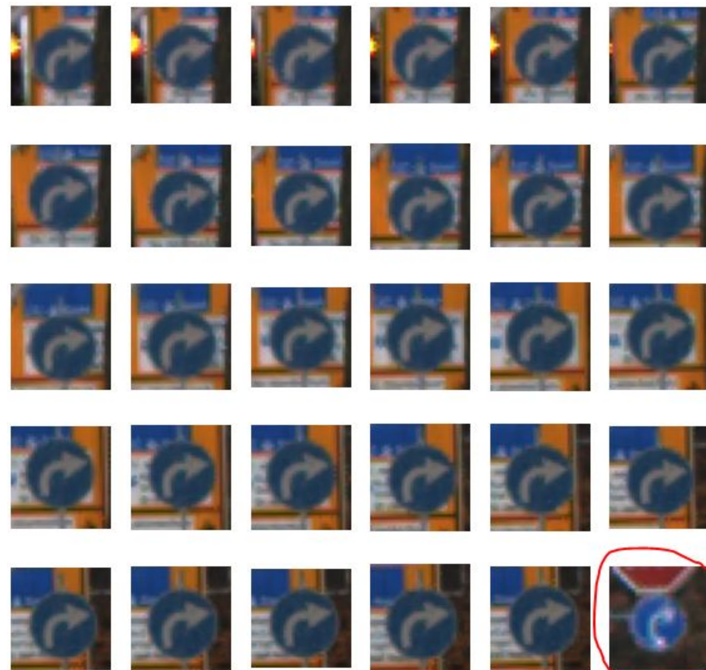
Appendix

Why 39209 images in dataset if each track has 30 images

After some detective work I discovered that the 19th track in class 33 only has 29 images.

The detective work involved printing the start index of each class. Then I found that **num_images mod 30 != 0** for class 33.

Then I printed 30 images at a time until I found that a single track was missing a single image.



The 30th image is missing from this track and instead the next track begins.

How the train/validation split function is implemented

```python
3   def make_train_val_split(train_x, train_y, test_size, use_stratified=True, shuffle=True):
4       # Since each track consists of 30 images we must include all 30
5       # in either the test or the train set to avoid contamination.
6       num_images = len(train_x)
7       num_tracks = num_images // 30
8       num_test_tracks = num_tracks * test_size
9
10      classes_start_idx = np.array([0]*43)
11      classes_num_tracks = np.array([0]*43)
12      curr_class = -1
13
14      for i in range(len(train_x)):
15          if (train_y[i] > curr_class):
16  #              print(f'i:{i}. label:{train_y[i]}')
17              classes_start_idx[train_y[i]] = i
18              curr_class = train_y[i]
19          classes_num_tracks[train_y[i]] += 1
20      classes_num_tracks = classes_num_tracks // 30
21      print(classes_num_tracks)
22
23      # Next we must choose if we should choose tracks randomly and risk not representing a class
24      # or choose in a stratified approach
25      train_Xs = []
26      train_ys = []
27      test_Xs = []
28      test_ys = []
29      if (use_stratified):
30          print(f'Using stratified')
31          for i in range(len(classes_num_tracks)):
32              # Insert test tracks and labels for current class
33              num_test_tracks = math.ceil(classes_num_tracks[i] * test_size )
34              class_test_indices = random.sample(range(classes_num_tracks[i]), num_test_tracks)
35              for track_idx in class_test_indices:
36                  for idx in range(30):
37                      curr_idx = classes_start_idx[i] + 30*track_idx + idx
38  #                      print(train_x[curr_idx].shape)
39                      test_Xs.append(train_x[curr_idx])
40                      test_ys.append(train_y[curr_idx])
41
42              # Insert train tracks and labels for current class
43              class_train_indices = np.delete(range(classes_num_tracks[i]), class_test_indices, 0)
44              for track_idx in class_train_indices:
45                  for idx in range(30):
46                      curr_idx = classes_start_idx[i] + 30*track_idx + idx
47  #                      print(f'{classes_start_idx[i]}. {track_idx}. {idx}. {curr_idx}')
48                      train_Xs.append(train_x[curr_idx])
49                      train_ys.append(train_y[curr_idx])
50      else:
51          raise "Not yet implemented"
52
53      train_Xs = np.array(train_Xs)
54      test_Xs = np.array(test_Xs)
55      train_ys = np.array(train_ys)
56      test_ys = np.array(test_ys)
57
58      print(train_x.shape)
59
60      if (shuffle):
61          train_indices = np.arange(train_Xs.shape[0])
62          test_indices = np.arange(test_Xs.shape[0])
63          np.random.shuffle(train_indices)
64          np.random.shuffle(test_indices)
65
66          return train_Xs[train_indices], test_Xs[test_indices], train_ys[train_indices] ,test_ys[test_indices]
67      else:
68          return train_Xs, test_Xs, train_ys ,test_ys
```

How the ensemble model is implemented in code

```
1 inputs = tf.keras.Input(shape=(image_height-pixels_to_crop, image_width-pixels_to_crop, 3,))
2 outputs = tf.keras.layers.average([model1(inputs), model2(inputs), model3(inputs)])
3 ensemble_model = tf.keras.Model(inputs=inputs, outputs=outputs)
4 print(ensemble_model.summary())
```

```
Model: "model_11"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_17 (InputLayer)           [(None, 48, 48, 3)]  0
__________________________________________________________________________________________________
sequential_21 (Sequential)      (None, 43)           383083      input_17[0][0]
__________________________________________________________________________________________________
sequential_22 (Sequential)      (None, 43)           946603      input_17[0][0]
__________________________________________________________________________________________________
sequential_23 (Sequential)      (None, 43)           1230763     input_17[0][0]
__________________________________________________________________________________________________
average_10 (Average)            (None, 43)           0           sequential_21[0][0]
                                                                 sequential_22[0][0]
                                                                 sequential_23[0][0]
==================================================================================================
Total params: 2,560,449
Trainable params: 2,560,449
Non-trainable params: 0
```

Here model1, model2 and model3 are the top 3 CNNs from the grid search. Each model outputs a prediction for each of the 43 classes (i.e. softmax). These outputs are then averaged and used for evaluating the training accuracy for each batch.

Top 1 CNN from grid search

Layer (type)	Output Shape	Param #
sequential_24 (Sequential)	(None, 48, 48, 3)	0
conv2d_54 (Conv2D)	(None, 44, 44, 32)	2432
elu_54 (ELU)	(None, 44, 44, 32)	0
max_pooling2d_36 (MaxPooling	(None, 22, 22, 32)	0
conv2d_55 (Conv2D)	(None, 18, 18, 64)	51264
elu_55 (ELU)	(None, 18, 18, 64)	0
max_pooling2d_37 (MaxPooling	(None, 9, 9, 64)	0
conv2d_56 (Conv2D)	(None, 6, 6, 128)	131200
elu_56 (ELU)	(None, 6, 6, 128)	0
flatten_18 (Flatten)	(None, 4608)	0
dropout_12 (Dropout)	(None, 4608)	0
dense_18 (Dense)	(None, 43)	198187

Total params: 383,083
Trainable params: 383,083
Non-trainable params: 0

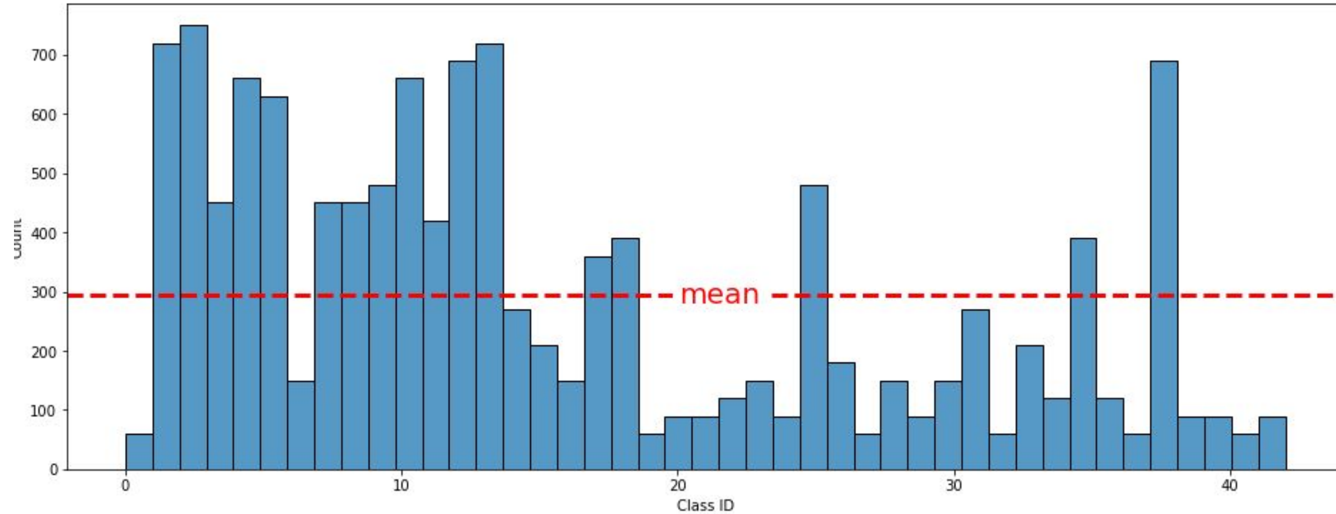
Top 2 CNN from grid search

```
Layer (type)                 Output Shape              Param #
=================================================================
sequential_24 (Sequential)   (None, 48, 48, 3)         0
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 40, 40, 64)        15616
_________________________________________________________________
elu_57 (ELU)                 (None, 40, 40, 64)        0
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 20, 20, 64)        0
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 17, 17, 128)       131200
_________________________________________________________________
elu_58 (ELU)                 (None, 17, 17, 128)       0
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 5, 5, 256)         524544
_________________________________________________________________
elu_59 (ELU)                 (None, 5, 5, 256)         0
_________________________________________________________________
flatten_19 (Flatten)         (None, 6400)              0
_________________________________________________________________
dropout_13 (Dropout)         (None, 6400)              0
_________________________________________________________________
dense_19 (Dense)             (None, 43)                275243
=================================================================
Total params: 946,603
Trainable params: 946,603
Non-trainable params: 0
```

Top 3 CNN from grid search

```
Layer (type)                 Output Shape              Param #
=================================================================
sequential_24 (Sequential)   (None, 48, 48, 3)         0
_________________________________________________________________
conv2d_60 (Conv2D)           (None, 44, 44, 64)        4864
_________________________________________________________________
elu_60 (ELU)                 (None, 44, 44, 64)        0
_________________________________________________________________
max_pooling2d_40 (MaxPooling (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 19, 19, 128)       131200
_________________________________________________________________
elu_61 (ELU)                 (None, 19, 19, 128)       0
_________________________________________________________________
max_pooling2d_41 (MaxPooling (None, 9, 9, 128)         0
_________________________________________________________________
conv2d_62 (Conv2D)           (None, 5, 5, 256)         819456
_________________________________________________________________
elu_62 (ELU)                 (None, 5, 5, 256)         0
_________________________________________________________________
flatten_20 (Flatten)         (None, 6400)              0
_________________________________________________________________
dense_20 (Dense)             (None, 43)                275243
=================================================================
Total params: 1,230,763
Trainable params: 1,230,763
Non-trainable params: 0
```

The test datasets class distribution resemble the trainings datas class distribution (see slide 6 to compare)



This suggests that the training set and the test set is i.i.d which is good because otherwise models trained on the training set might not generalize well.

In each jupyter notebook there is a confusion matrix at the end showing that class 30 is the most problematic. Looking at the images that were misclassified I believe it should be possible to write some custom augmentations to more accurately simulate weather and lighting conditions encountered on the road. I used the basic build in functions for modifying images but they are very coarse. I believe custom augmentation functions customized to simulate road conditions would improve accuracy slightly.

There are lots of cool implementation stuff in the notebooks. But I’ve presented the most important results here.

The end