UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Artificial Intelligence and Machine Learning (6CS012)

Traffic Sign Classification Using Deep Learning

Suren Tamang	: 2330715
Group	: L6CG16
Tutor	: Sunita Parajuli
Lecturer	: Simon Giri
Submission Date	: 20 May 2025

Abstract

In the day of this modern time, there is developing the technologies in the field of autonomous vehicles, actions without the support of drivers. One of the most demanded technologies in AV is traffic sign detection and classification models. So based on the inspiration of solving the classification problems, here it is focused on classifying the huge number of traffic sign images using deep learning methods which is the main objective of this project.

To solve this problem, CNN is the most popular and effective deep learning model more focused in the field of image classification. The model is prepared from scratch and transfer learning technologies where pre-trained models are adapted based on the dataset.

Durin the scratch, the trained model gets overfitting that is monitored by the validation set, to minimize the overfitting problem, there has been used of regularization techniques such as dropout, L2 and early stopping when helped in the improvement of model performance with the accuracy of more than 90 percent in both training and testing. Moreover, transfer learning is implemented but got some fluctuation in accuracy, so got the idea of choosing the best pre-trained model based on the project and problem statement which only then performs the best result.

Table of Contents

1. Introduction 1
2. Dataset 1
3. Methodology 2
   3.1. Baseline model 2
   3.2. Deeper model 4
   3.3. Loss function 4
   3.4. Optimizer 5
   3.5. Hyperparameters 6
4. Experiments and Results 6
   4.1 Baseline vs. Deeper Architecture 6
   4.2 Computational Efficiency 7
   4.3 Training with Different Optimizers 7
   4.4 Challenges in Training 8
5. Fine-Tuning or Transfer Learning 9
6. Conclusion and Future Work 10

Table of Figure

Figure 1 Dataset structure 1
Figure 2 Categorical cross entropy 5
Figure 3 SGD optimizer 5
Figure 4 Adam optimizer 6
Figure 5 Model comparison table 7
Figure 6 Optimizer comparison table 8
Figure 7 Base vs Deep vs Pretrained model 9

1. Introduction

In the world, the necessity of transportation to travel from one place to another is the challenging part without a vehicle, then. There are many rules and regulations that need to be followed while driving the vehicle, but without the driver in the vehicle, it is a more challenging and problematic situation which can cause accidents. Hence there is an autonomous vehicle that runs without any drivers but needs to follow the same rules and regulations of the road.

So, the main problem statement is that " Can autonomous vehicle run in the road by following the rules and regulation of traffic sign ?". To solve this problem, there is the need of classification model after the detection model of traffic sign. Hence, this project is based on the traffic sign image classification model using the CNN deep learning model which can perform the best performance in the image classification task.

And the goal of this project is to get and work on GTSRB dataset so that the model will be trained well and able to classify the traffic sign with good accuracy. Also, can be deployed for AV as well in the future reference.

2. Dataset

To work on the project of image classification tasks, we team members decided to choose the GTSRB dataset which is widely used for the traffic sign classification task. The dataset is found from the Kaggle source since this is the publicly available dataset, which was first released in 2011, to introduce the traffic sign recognition and classification competition in the field of neural networks.

The dataset is managed in the folder and CSV format in Kaggle with large number of sample images with 43 different classes. Here, the dataset is organized like this,

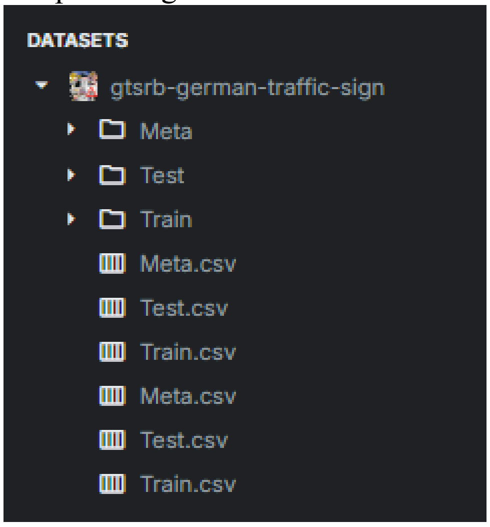
```
DATASETS
▾ gtsrb-german-traffic-sign
    ▸ Meta
    ▸ Test
    ▸ Train
      Meta.csv
      Test.csv
      Train.csv
      Meta.csv
      Test.csv
      Train.csv
```

Figure 1 Dataset structure

There are 43 classes in the dataset which represents the unique traffic sign images having different pixel sizes. Some of the classes images are speed limit, warnings, prohibitions and more. For these classes, there are many sample images that are associated with different sizes of pixels such as 32 by 42, 64 by 46 and many more. Since the images size are in different pixels, the images are preprocessed to keep them in 32 by 32 pixel size so that they can be used later for the input in classification model. Cv2, NumPy and pandas are the top libraries that have been used to load, read and preprocess the dataset to prepare for the model.

Moreover, to minimize the computational power, the images are normalized into the range of 0 and 1, which is also known as the min max normalization that directly helps the model converge faster and perform better. Hence before normalization, the images are transformed into the fixed 32 by 32 pixel sizes and NumPy array, only then possible to rescaling the sample images in the range of 0 and 1.

The following are done for the preprocessing of the dataset.

a. Filtered out the classes(19 classes) that have less than 500 hundred sample images
b. Check for the corrupted images so that the dataset must be cleaned.
c. Prepared the images and labels by reading and resizing the pixels of images and stored them in the corresponding list.
d. Converted the images and labels in NumPy array to make the training faster.
e. Sample images proceed for the min max normalization
f. Label data is proceeded for label and one hot encoding in 24 classes.
g. It is ready to split the dataset into the train test split to train the model with validation dataset and monitoring the overfitting.

The dataset is split into training and validation set and then again train set is taken of first 2600 sample images so that can train in small set and validate in large set.

Original training set:-39209 sample images, 43 classes

Filtered training set:- 33239 sample images, 24 classes

Split training set:- 26591 samples images, 24 classes

Split validation set:- 6648 sample images, 24 classes

Training set:-2660 sample images, 24 classes

Testing set:- 12630 sample images, 24 classes

3. Methodology

3.1. Baseline model

Working on the image classification task, here the required flow and steps need to follow to get the proper learned and trained model.

Building the model:- Sequential
Compile the model:- Includes optimizer, loss function and metrics
Train the model:- Fit the model with certain epoch and batch size: 32

Evaluate the model: Evaluating the trained model in an unseen train set of datasets

Building the model
First this is the model that needs to be prepared with the normal architecture of the CNN and fully connected neural network. So , there are two parts of the whole model that is separated with the flattened layer, before the flattened layer, the architecture can be as the feature learning and after that can be said the classification architecture of fully connected neural network.

Feature learning:- This is made of convolutional layer, Activation layer and pooling layer where first layer is for structuring and learning the low to high level of images edges with feature map, and the second layer is for converting features map into the relu activation function still in feature map matrix, and the third layer is to reduce the dimension of the feature map with preserving the spatial information of the images respectively.

The hyper parameters of this first architecture are:

- Number of filters - 32
- Size of filters – 3 by 3
- Padding – same
- Pooling – 2 by 2 MaxPooling2D
- Stride – 1, 1 for Conv2D, 2,2 for pooling
- Activation function – relu

Classification:- The architecture is fully connected neural network aims to classify the images based on the number of classes also known as multi class classification. There are three main layers that perform the operation on the classification task. Input, hidden and output layers are responsible so that they are connected each other and names as dense layers as well.
The images are flattened before the input layer and passed into the input layer; hence the total number of image pixels is equal to the number of neurons in the input layer. And the images are fed into the dense hidden layer to recognize the image with the help of activation function relu in each neuron of the hidden layer.
And the last output layer is responsible for classification of the images, so the SoftMax activation function is mandatory to be used in this layer. So, in the dataset after filtering the classes, there are 24 classes, so the SoftMax functions help to classify the image with the highest probability distribution.
The hyperparameter of this architecture are:

- Number of dense layers – 3 assigned
- Size of dense layer – 256, 128, 64
- Size of output layer based on the classes – 24 classes
- Activation function – relu

3.2. Deeper model

Following the above base model architecture, there has been added some of the important performance changeable regularization layers so that minimizing the overfitting is the objective of working in this deeper model.
In this deeper model,
The model is built with double convolutional layers and the regularization technique are implemented. Just the number of filters is updated with 64 and 128 in the 2nd and 3rd block of convolutional layers. And in the dense layer, there has been added one layer with more neurons.
The Overfitting mitigation techniques implemented are:
L2 regularization:-
The is one of the best methods to reduce overfitting which adds the penalty to large weight so that can encounter smaller and more generable weights.

Dropout:-
This is the technique to drop the neurons from the layers during the time of training that can prevent overfitting and improve the performance of models.

Early stopping:-
The models are directly dependent on this technique to avoid overfitting problems by stopping the training when the validation loss stopped improving, so help in preventing the over training and save the time as well.

Batch Normalization:-
To speed up the training, this helps to normalize the activations in each mini batch by making each layer to zero mean and unit variance. It acts like the regularizer that is placed before the relu activation function.

3.3. Loss function

The task is to classify traffic sign images, so to get the error from the actual and predicted results, there needs to be the loss function which measures the difference between actual and predicted probabilistic distribution. Since there is multi class classification of traffic sign images, categorical cross entropy is the best loss function that is used in the project.

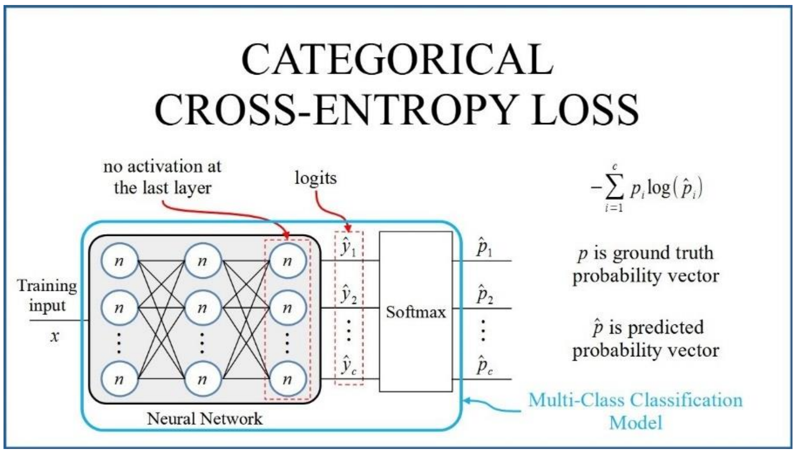
CATEGORICAL CROSS-ENTROPY LOSS
no activation at the last layer
logits
$-\sum_{i=1}^{c} p_i \log(\hat{p}_i)$
Training input
x
n
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_c$
Softmax
$\hat{p}_1$
$\hat{p}_2$
$\hat{p}_c$
p is ground truth probability vector
$\hat{p}$ is predicted probability vector
Neural Network
Multi-Class Classification Model

Figure 2 Categorical cross entropy

To work on this loss function, the actual label is from dataset and the predicted label comes from the image data that should be converted into NumPy array and y label needs to convert into one hot encoding. And it is possible to distribute the probability in each class. 24 classes are managed into 24 columns, and only the corrected label will be 1 in the 24 columns and other columns will be zero.

Source: Google

3.4. Optimizer

During model compilation, the optimizer is defined as the parameter of the model, it is an algorithm which helps in optimizing the weights of the model that directly for reducing the loss function during training the model. There are many types of optimizers in the field of deep learning, two of them that have implement in this classification tasks are:-

SGD

This is the algorithms updating the weight for each sample one by one rather than using the whole batches dataset. This optimizer is very simple and widely used that makes slow in the convergence and may stop in local minima. So , it uses the momentum to accelerate learning.

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Figure 3 SGD optimizer

Adam

This is another optimizer of having the combination of SGD and RMAProp, adjusting the learning rate in each parameter. As compared to the SGD, it has inverse characteristics that have fast convergence, works better with noisy gradients, and the update rule is different as well.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Figure 4 Adam optimizer

3.5. Hyperparameters

The parameters for which the value is defined by the developer are based on the dataset and tuning techniques that is before training the model helps in controlling the learning process. Then it is called hyper parameter. The hyperparameter used for building the model is already explained. So, during model training the following hyperparameter are implemented:-

Learning rate: This helps for updating the model during training, it is unstable if it is set to high, and it is slow learning if it is set to low. For the project the LR is default, and it is showing good performance in default LR of optimizer. It is 0.001 for Adam and 0.01 for SGD.

Batch size: Batch is for assigning the number of training samples in forward and backward pass. 32 is set for the traffic sign image classification task.

Number of epochs: Based on the dataset size and the batch size, and the estimation of training the model, the number of epochs is assigned. For the base model, there are 30 epochs, and for the deep model, there are 40 epochs.

4. Experiments and Results

4.1 Baseline vs. Deeper Architecture

The baseline model is prepared with the normal three convolutional and the three dense layers where the convolutional layers have set the 32, 64 and 128 filters with 3 by 3 sizes.

And the deeper model is built with the double layer of the baseline model and trained with the regularization techniques to avoid overfitting.

The following shows the comparison table of two models.

Models	**Training accuracy**	**Validation accuracy**	**Testing Accuracy**	**Epoch**	**Training Time**

Baseline Model	100%	94.27%	87.29%	30	197.42s
Deep Model	94.10%	97.79%	95.57%	40	823.62s

Figure 5 Model comparison table

From the above table, baseline model is getting the overfitting problem where the model got the 100% accuracy, but the validation accuracy is 94.27% which is bad for unseen data. Not that bad since it is already the 94% accuracy which is great for unseen data. And the total time taken for 30 epoch is about 198 seconds which to too fast since the images are inputted the small pixels. And after evaluating the model with test unseen images, the accuracy is 87.29%, which decreased with 13% from the training accuracy.

After adding the extra dense and convolution layers, and implementing the batch normalization, drop out, L2 regularization, early stopping with the default hyperparameter value, there is the improvement in the model performance, even after the epoch is increased to 40, the training accuracy, validation accuracy, and testing accuracy are 94.10, 97.79, and 95.57% respectively. Hence the overall performance of the deep model is great after taking about 824 seconds of training the model.

4.2 Computational Efficiency

During training, there has been used of normal CPU, the device only supports the CPU processor, even there is available of GPU/TPU, limitation of using that facility was ended, for caring of the storage management from downloading the dataset, the project is done in the root source of the dataset in Kaggle notebook, this leads to no downloading the folders of images.

Training time of the base and deep model has huge difference as shown in the above table, there the time taken to train in the base model was about 198 seconds where, deep model took around 824 seconds. The more layers, epoch and the regularization layers make the deep model time consuming as compared to the normal base model.

4.3 Training with Different Optimizers

To prevent the model from being stuck in local minimum, the SGD model is built with less layers as compared to the Adam. And the hyperparameter is managed accordingly so that the model performs well.

The deeper model with SGD has been trained with 2 blocks of convolution layers having four layers and 3 dense layers to get better results. Where the deeper model with the Adam optimizer has a little bit difference in the model architecture of having 6 convolutional layers in 3 blocks, and 4 dense layers has been implemented. The table below shows the comparison of convergence speed and final accuracy of both deep models.

Deep Model	**Training accuracy**	**Validation accuracy**	**Testing Accuracy**	**Epoch**	**Training Time**
Adam Optimizer	94.10%	97.89%	95.57%	32/40	823.62s
SGD optimizer	97.18%	98.08%	95.90%	55/ 100	938.97s

Figure 6 Optimizer comparison table

When the model was built with the same layers and hyperparameters, the model was not converging to the global minima when using the SGD optimizers, so some of the layers and hyperparameters are modified,

All the parameters for Adam were default, but SGD, it is 0.001 learning rate, and dropout rate is 0.2 and 0.3 for dense layer. And the exponential decay in learning rate is applied to the SGD optimizer.

4.4 Challenges in Training

Based on the assignment guidelines, there needs to be built and train 4 models with different optimizers and techniques. Until the implementation of Adam optimizer and train the model, there was less computational cost. But the SGD optimizer problems so many issues. During training the model, the following are the challenges that faced:-

- Proper model architecture layers – Base model and deep model needs to identify the best number of layers and hyperparameters.
- Slow train in SGD – If the hyperparameters are updated, then it needed to train the model again.
- Split the train set with proper sample images – due to large number of sample images, need to plan the perfect sample images for each class.
- Overfitting in base model – the model is trained with less data than the validation data.

- Fitting the regularization layer in deep model – to know the best value for dropout, early stopping and L2.

5. Fine-Tuning or Transfer Learning

To adapt the dataset sample images to pretrained model, there have been options of implementing the two techniques featuring extraction and fine tuning, where the feature extraction and then fine tuning both are implemented for the project.

In the feature extraction technique, the base model of the pretrained model freeze and the top layer that classifies the classes are replaced by the project classes. So, the VGG16 is the pretrained model used to train the classification layer of the dataset after preprocessing the sample images based on the model. To improve performance fine tuning is applied by continuing the epoch of training.

This model is implemented due to its simplicity and uniform architecture, developed by Visual Geometry Group, Oxford. The short description of VGG16 pretrained model is,

- Input size:- 224 by 224
- No of convolutional layers :- 13
- Pooling layers :-5 max with 2 by 2 filter and 2 stride
- FCN:-3
- Activation:- Relu after each FCN and Conv layers

The total learnable layers are 16, so this is called VGG16.

Since the model has too many layers and parameters, it took more time to train the model as compared to the scratch models. This pretrained model has given a good performance with training accuracy of 94.89% and testing accuracy of 90.56%.

The comparison from scratch is shown in the table below.

Models	**Training accuracy**	**Validation accuracy**	**Testing Accuracy**	**Epoch**	**Training Time**
Baseline Model	100%	94.27%	87.29%	30	197.42s
Deep Model	94.10%	97.79%	95.57%	40	823.62s
VGG16	94.89%	89.17%	90.56%	20	3130.34s

Figure 7 Base vs Deep vs Pretrained model

6. Conclusion and Future Work

The project provided important and foundational knowledge in the field of deep learning image classification task. Since the task is performed under the different ways and techniques. So many challenges were faced during model training and got new information in every mistake. There got the ideas of finding the best rules and techniques to make generalize models such as hyper parameter tuning, the importance of number of layers, proper definition of regularization, and more.

The best model was a deep model with regularization layer that shows the best accuracy and proper convergence as compared to the base and pretrained model. Even though the task looked easy there were many challenges in the image preprocessing so that it took longer to complete this project. If there is a chance to work in this project again, there is need to be more management in the dataset and understating the pretrained model which makes good direction in future research.

Since the transfer learning performed overfitting even after too many hyper parameters tuning, there will be further research in this portion to get more improvement and proper completion of this project.

Thank You