Group Assessment: Traffic Sign Classifier
COSC2673 Machine Learning
RMIT University Melbourne, Semester 2024A

Nguyen Minh Quan - s3979391
Do Le Trang Hanh - s3977955
May 15th 2024

I. Problem Definition

1.1. Project overviews

As prevalent as ever, Machine Learning modeling has become a crucial tool for various purposes in life. From simple tasks such as making suggestive predictions for some data with a given pattern to highly accuracy-centric and pivotal tasks such as medical diagnoses and movement recognition. Among its diverse applications, one pivotal task is traffic sign classification—a domain where precision is paramount for ensuring road safety and efficient traffic management.

In this report, we explore the nuances of machine learning modeling in the context of traffic sign classification, delving into its methodologies, challenges, and significant real-world applications. Leveraging the Belgium Traffic Sign Classification Benchmark, comprising nearly 4000 images categorized into 5 Shapes and 16 Types, we employed a blend of pipelining methods, preprocessing techniques, and model comparisons. Through iterative enhancements to model design, our primary objective was to delve into and comprehend the foundational principles of Neural Networks and analogous classification architectures.

1.2. Problem Statement

Goal: *Machine learning model capable of classifying* **Traffic Signs** (***5 Shapes and 16 Types)*** *based on the given image dataset.*

Strategy:

1. **Exploratory data analysis.**
2. **Dataset preprocessing and calibration.**
3. **Baseline Model Configuration and Testing.**
4. **Model optimization and Insights.**

1.3. Metrics

- **Sparse Categorical Accuracy:** A measure of how accurately a model predicts the correct category among multiple categories, calculates the rate of correct predictions of the model.

$$Sparse\ Categorical\ Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

Reasoning: We selected sparse categorical accuracy as the primary metric to evaluate our model's performance due to its suitability for handling sparse categorical data. Unlike other accuracy metrics, it appropriately accounts for the imbalance in the dataset by considering only the correct predictions made for each category. This metric is robust and provides a comprehensive understanding of how well the model performs across different categories, making it suitable for evaluating classification models dealing with sparse categorical data.

- **Sparse Categorical Cross Entropy:** A measure of the difference between the predicted probability distribution and the true probability distribution of the categorical labels. It quantifies the "distance" between these two distributions.

$$Sparse\ Categorical\ Cross\ Entropy = -\frac{1}{N} - \sum_{i=1}^{N} y_i \log\left(p_i\right)$$

Reasoning: One of the most common model scores for classification via Neural Network problems, we chose this as an additional metric to compare the models. This metric penalizes the model more severely for predicting probabilities far from the true distribution, thus encouraging the model to learn accurate representations of the categorical labels. Additionally, it provides insights into how well the model's predictions align with the ground truth across all categories, making it a valuable metric for assessing classification models dealing with sparse categorical data.

- **Visual coordination of Train-Validation Accuracy Curve in the Learning Curve Plot:** An additional and not quite conventional metric but visual representation of each model's learning process bears significant importance to knowing the progression of each model, as well as performing critical analysis on model performance with (un)anticipated issues that the models might encounter.

II. Exploratory Data Analysis (EDA)

2.1. Initial Dataset Design

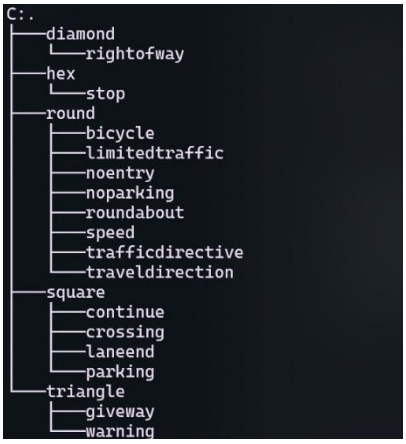

Figure 1. Dataset Directory Tree

The datasets provided are relatively small compared to conventional Machine Learning image classification datasets. With 3699 images divided into 5 main directories representing the traffic signs' Shape, as well as 16 total leaf directories which correspond to their Type as well. (As presented in Figure 1)

Insights:

All images are generally preprocessed beforehand, with all images from the directory rescaled to 28x28 (Height x Width) and Gray Scaled. This is very handy to process because we could directly move on to see the distribution of data without initial processing and data exploration.

2.2. Distribution Disparity

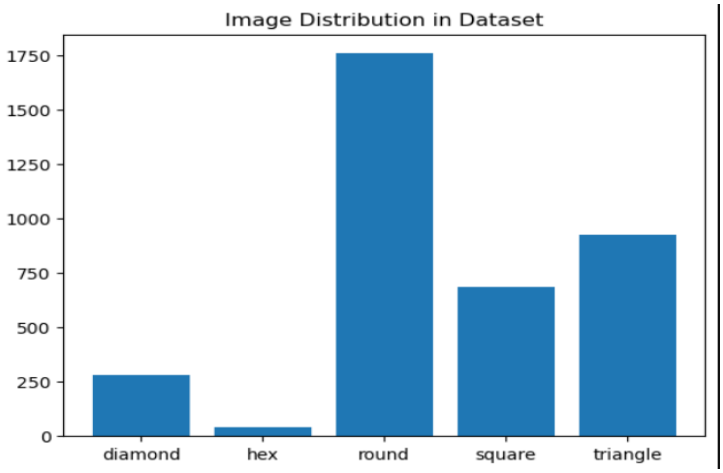

Figure 2. Dataset Imbalance

Insights:

Figure 2 illustrates a notable disparity in the distribution of images among classes, with the 'round' class containing a substantially higher number of images compared to the 'hex' class, which is significantly lower in comparison to the other classes. This imbalance could potentially bias future models towards predicting images from the 'round' class more frequently than other classes, particularly the 'hex' class.

III. Dataset preprocessing and calibration.

3.1. Data Augmentation: Classify by Shape and Classify by Type

Initially, we implemented a classic MLP structure to create a baseline model. The goal then was to know if the model's interpretation was indeed biased based on the data distribution of the dataset. After confirming that hypothesis, we decided to modify the original dataset into 2 separate datasets for separate training protocols. Essentially, the logic is: Using ImageDataGenerator from Tensorflow to create a different modification image one by one from each class until reaching a certain limit. Due to the high discrepancy in the number of images of each class after calibration for Classification by Shape, with their size ranging from 3800 up to 4600 images, we decided to cut down that dataset to the number of the directory with the least number of images, the result can be seen in Figure 3. For Classification by Type, we did not change anything since the discrepancy in images of each category was not too much, as presented in Figure 4.

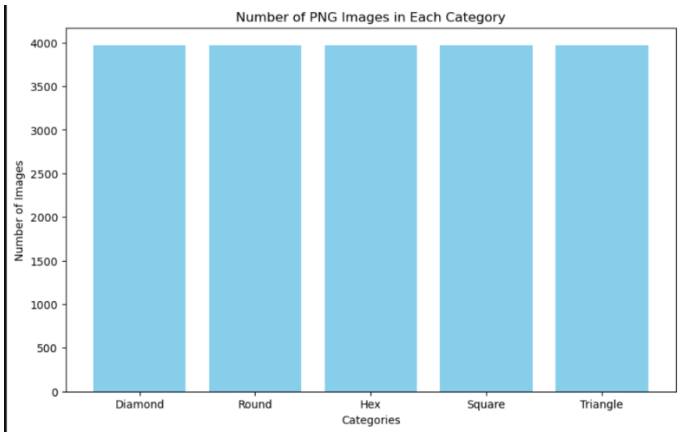

Figure 3. Classify by Shape

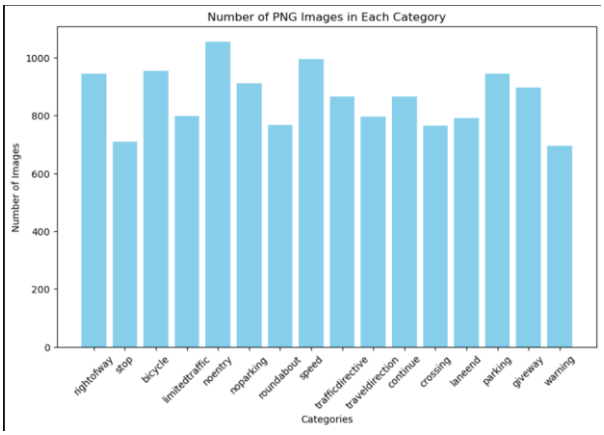

Figure 4. Classify by Type

IV. Baseline Model Configuration and Settings

```
MLP = Sequential([
    keras.Input(shape=(28, 28, 1)),
    Rescaling(1/255),
    Flatten(),
    Dense(512, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dense(5, activation='softmax'), #This is fixed for multiclass classification
])

MLP.compile(
    optimizer=Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['sparse_categorical_accuracy']
)
```

Figure 5. Baseline Model: Multi-Layer Perceptron

We immediately thought of several Convolutional Neural Network designs after choosing this project, so picking a classic MLP design to be our baseline would be the most suitable option. With 1 Input Layer, 2 Hidden Layers with decreasing neurons and a 50% dropout rate. The model seems to have a very stabilized learning curve as well as an astonishingly strong validation accuracy growth, which grows from 28% to 91% just after 10 epoch. It can be seen in the code section that although the model gave excellent predictions on the given dataset for each type of classification, it did not perform very well on the external dataset. This could be a classic overfitting problem of the model. However, it was a desirable outcome, since our initial aim was to reach a certain degree of generalization from the baseline model. It shows that our Data Augmentations are effective.

V. Model optimization and Insights.

We initially came up with a list of possible algorithms for our model besides Convolutional Neural Network, such as K-Nearest Neighbor, Naïve Bayes Classifier, Random Forest Classifier and Support Vector Machine. Due to each characteristic of them being unfit for this task (KNN – High Dimensionality, Naïve Bayes – Feature Independence, Random Forest and SVM – Computationally Expensive), we decided to only utilize and optimize various CNN models.

5.1. Model Design: Convolutional Neural Network

We proposed 7 incrementally improved designs in our work:

Model	Design Characteristic
Enhanced CNN (code: model_1)	4 Conv2D Layers -> Flatten -> 1 Dense Layer -> 0.5 Dropout -> 1 Dense Layer
Enhanced CNN wMP (code: model_2)	4*(Conv2D – Maxpool2D) -> Flatten -> 1 Dense Layer -> 0.5 Dropout -> 1 Dense Layer
Enhanced CNN wMP wBN (code: model_3)	4*(Conv2D – BatchNormalization() – Maxpool2D) -> 2 Dense Layers -> 0.5 Dropout
Semi - Deep Neural Network (code: model_4)	3*(2 Conv2D – Maxpool2D) -> Flatten -> 2 Dense Layers -> 0.5 Dropout -> 1 Dense Layer
Semi - Deep Neural Network wBN (code: model_5)	3*(2 Conv2D – BatchNormalization() – Maxpool2D) -> Flatten -> 2 Dense Layers -> 0.5 Dropout -> 1 Dense Layer
Inception-Net Inspired Net (code: model_6)	Conv2D (1x1) -> Conv2D (3x3) -> Conv2D (5x5) -> Maxpool2D -> Conv2D (3x3) -> Conv2D (5x5) -> Maxpool2D -> Flatten -> 1 Dense Layer
Inception-Net Inspired Net wBn (code: model_7)	Conv2D (1x1) -> Conv2D (3x3) -> Conv2D (5x5) -> BatchNormalization() -> Maxpool2D -> Conv2D (3x3) -> Conv2D (5x5) -> BatchNormalization() -> Maxpool2D -> Flatten -> 1 Dense Layer

Rationale:

- **Model_1 (Enhanced CNN):** This model starts with a classic Convolutional Neural Network (CNN) structure, with gradually increasing depth and complexity with each additional convolutional layers.
- **Model_2 (Enhanced CNN with Max Pooling):** A continuation to Model_1, this model incorporates max-pooling layers after each convolutional layer, reducing the spatial dimensions of the feature maps while retaining important information. This introduces feature capturing and reduces computational complexity for the model.
- **Model_3 (CNN with Max Pooling and Regularization):** Introducing batch normalization layers between convolutional layers helps stabilize and accelerate the training process by normalizing the input to each layer. This could lead to faster convergence and higher degree of generalization for the model.
- **Model_4 (Deep Neural Network with Stacked Convolutional Layers):** Model_4 explores a deeper architecture by stacking multiple convolutional layers before max-pooling, enabling a more advanced feature extraction capability of the model.
- **Model_5 (Deep Neural Network with Batch Normalization and Stacked Convolutional Layers):** An improvement to Model_4, but with the inclusion of batch normalization layers for improved training stability and convergence. This combination aims to enhance the model's ability to learn discriminative features while mitigating overfitting.
- **Model_6 (CNN Inspired by InceptionNet with Varied Architecture):** Inspired by Google's InceptionNet architecture, this model employs a combination of different kernel sizes within convolutional layers to capture features at multiple scales. This strategy can potentially enhance the model's ability to discriminate between different traffic sign patterns.

- **Model_7 (CNN Inspired by InceptionNet with Batch Normalization and Max Pooling):** Essentially an improvement to Model 6 with the addition of batch normalization layers for improved training stability. Max-pooling layers are included to reduce spatial dimensions and extract crucial features.

Compile Method:

```
# Iterate through each model
for target_model in NeuralNetwork_models:
    target_model.compile(
        optimizer=RMSprop(learning_rate=1e-6, momentum=0.9),
        loss='sparse_categorical_crossentropy',
        metrics=['sparse_categorical_accuracy']
    )

    target_model.summary()

    hist = target_model.fit(
        train,
        epochs=20,
        batch_size=200,
        validation_data=val
    )

    # Plot the learning curve
    plot_learning_curve(hist)

    # Evaluate the model
    vResults_tf(target_model, val, data.class_names)
    vResults_raw(target_model, raw_data, data.class_names)
```

Figure 6. Classify by Shape Compiler

```
# Iterate through each model
for target_model in NeuralNetwork_models:
    target_model.compile(
        optimizer=RMSprop(learning_rate = 1e-4),
        loss='sparse_categorical_crossentropy',
        metrics=['sparse_categorical_accuracy']
    )

    target_model.summary()

    hist = target_model.fit(
        train,
        epochs=25,
        batch_size=64,
        validation_data=val
    )

    # Plot the learning curve
    plot_learning_curve(hist)

    # Evaluate the model
    vResults_tf(target_model, val, data.class_names)
    vResults_tf(target_model, test, data.class_names)
    vResults_raw(target_model, raw_data, data.class_names)
```

Figure 7. Classify by Type Compiler

Classify by Shape inherently has a smaller output gate than by Type, which causes differences in computational complexity and models' learning growth. As can be seen in Figure 6 and 7, we regulated that exponential rate of growth by implementing different learning rates for the 2 compilers, and through reiterative testing, declared the most effective number of epochs for each type of classification.

The first 3 models were a testing ground for us to understand how addition of different components affect the performance of the model in general. By stacking additional layers on top of each other, we understood how layers such as Maxpool2D and BatchNormalization enhances the model.

The last 4, as their name suggests, was inspired by such designs. These models aim to leverage proven architectural principles to further enhance performance and robustness in traffic sign classification tasks. This systematic approach enabled us to iteratively refine our model designs, ultimately leading to a more informed selection of architectures for effective image classification.

VI. Conclusion

6.1. Models Evaluation and Comparison

Across both Classification by Shape and by Type, all 7 CNN models produced a very consistent train-validation curve and a very satisfactory and consistent learning growth (as regulated by the learning rate). However, all 7 models, albeit having such a remarkable performance, as most of them reaches 90% accuracy just after 50% number of epochs, did not get to the degree of generalization that we expected, as most of them just get 9-10 images right, per batch of 16 images taken from the external dataset.

This might have happened due to several speculated reasons. Firstly, this might have been a classic case of overfitting, due to my overestimation of the models' capability to classify images, let alone perform calculation on a specifically low-resolution image set (28x28). Secondly, the malperformance of the optimized models could have been an 'optimized solution' for a 'not yet optimized problem'. During the Data Augmentation phase, specifically during the augmentation for Classify by Shape dataset, randomly trimming down extra data from dataset with large amount of image could cause internal imbalance between the image types inside a directory. For instance, I have 10 images of one type with label 'A', and 1 image of a

different type with the same label. In such case, the model would be inclined to recognize the image from the 10-image batch rather than a single example.

6.2. Ultimate Judgement

It could be worth having a speculation about the different tuning methodologies and further optimization on the Convolutional Neural Network designs proposed in this paper. Regarding tuning methodologies, 'RMSprop' was used as an optimizer to regulate the learning rate of each model to ensure a stable learning progression for each model and also to avoid vanishing gradients. 'Adam' optimizer was experimented on beforehand, which unexpectedly brought the vanishing gradients back to our models, which leveraged our models' accuracy up to 90%+ even from the very first epoch. Model optimization is also a considerable aspect as well, given that we could have implemented a simpler design, with less layers and possibly lower learning progress, but the progression should be consistent.

Regarding the proposed models, model_4, which implements a classic design of Inception Net has the closest Train-Validation Accuracy Curve, suggesting a steady learning progression. Interestingly, model_1, initially intended as a mere testing model, demonstrates a similarly consistent curve. This unexpected consistency prompts further exploration and experimentation with model_1, potentially unveiling untapped potential and opportunities for refinement.

References

Great Learning Team. (2023, May 30). Hyperparameter tuning with GridSearchCV. Great Learning Blog: Free Resources what Matters to shape your Career!. https://www.mygreatlearning.com/blog/gridsearchcv/

The scikit-yb developers. (n.d.). Alpha selection — Yellowbrick v1.5 documentation. Yellowbrick: Machine Learning Visualization — Yellowbrick v1.5 documentation. https://www.scikit-yb.org/en/latest/api/regressor/alphas.html

Sklearn.model_selection.GridSearchCV. (n.d.). scikit-learn. Retrieved April 7, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV

MYKOLA. (n.d.). GTSRB - German Traffic Sign Recognition Benchmark. Retrieved April 7, 2024, from https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign?select=Test

İlaslan, D. (2023, July 23). Image classification with TensorFlow. Medium. https://medium.com/@ilaslanduzgun/image-classification-with-tensorflow-a361c7b1eb05

Singh, A. (2020, May 8). Demystifying the mathematics behind convolutional neural networks (CNNs). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/02/mathematics-behind-convolutional-neural-network/

Rosebrock, A. (2021, May 14). Convolutional neural networks (CNNs) and layer types. PyImageSearch. https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/

Vasudev, R. (2019, February 11). Understanding and calculating the number of parameters in convolution neural networks (CNNs). Towards Data Science. https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d

Contents

I. Problem Definition 1
1.1. Project overviews 1
1.2. Problem Statement 1
1.3. Metrics 1
II. Exploratory Data Analysis (EDA) 2
III. Dataset preprocessing and calibration. 3
3.1. Data Augmentation: Classify by Shape and Classify by Type 3
IV. Baseline Model Configuration and Settings 3
V. Model optimization and Insights. 4
5.1. Model Design: Convolutional Neural Network 4
VI. Conclusion 5
6.1. Models Evaluation and Comparison 5
6.2. Ultimate Judgement 6
Table of Figures 7

Table of Figures

Figure 1. Dataset Directory Tree 2
Figure 2. Dataset Imbalance 2
Figure 3. Classify by Shape 3
Figure 4. Classify by Type 3
Figure 5. Baseline Model: Multi-Layer Perceptron 3
Figure 6. Classify by Shape Compiler 5
Figure 7. Classify by Type Compiler 5