Traffic Sign Classification

Everett Periman
University of Florida
Introduction to Machine Learning EEL5840
Gainesvillle FL ,USA
everettperiman@ufl.edu

Michelle Dupuis
University of Florida
Introduction to Machine Learning EEL5840
Gainesvillle FL ,USA
mdupuis@ufl.edu

***Abstract*—Traffic sign classification is critical for advanced driver assistant systems in vehicles. In this study, machine learning techniques were explored to determine the best accuracy on training and blind test datasets. In this experiment, ten classes of traffic signs were collected in real life environments. After several initial experiments were conducted, it was determined that Convolutional Neural Networks resulted in the best recognition performance for both training and test datasets.**

INTRODUCTION

Traffic sign recognition and classification are essential for advanced driver assist functionality as the vehicle industry strives for autonomous driving. In this paper, deep learning especially the transfer learning method is explored to identify the best algorithm for accurate and robust traffic sign classification. Transfer Learning is a machine learning method that takes features that are learned on one problem and applies it on a new or similar problem. It is considered an optimization that saves time and can yield better performance than designing a model from scratch. The normal workflow consists of taking the layers from a previous trained model and freezing them. Some new trainable layers are added to the top of the frozen layers and then the new layers are trained on the new dataset. There is an additional optional step that consists of unfreezing all or partial layers of the model to retrain on the new data at a low learning rate. This is called fine tuning [4].

Convolutional Neural Networks (CNN) are multi-stage deep architectures that alternates convolutional layers with pooling layers followed by fully connected layers. Convolutional operation produces a weighted sum of inputs by sliding a weighted window across the entire image. The pooling operation is applied to capture the more significant feature representations and reduce computational time. CNN facilities the effective feature extraction while reducing the parameter and neuron connections. CNN is a state-of-the-art technique to use for image recognition and classification. For the classification described in this paper, it is common to use a deep learning CNN model that is already trained for image classification such as ImageNet. There are multiple pretrained models available with the top three being VGG(VGG19), GooLeNet (InceptionV3) and Residual Network (ResNet50) [5]. For this study, the InceptionV3 model was selected.

IMPLEMENTATION

Data Collection

Ten samples of ten classes of traffic signs were collected under real environment conditions by each individual student. These totaled 100 images per student. The class label names were ‘Stop’, ’Yield’, ’Red Light’, Green Light’, ’Roundabout’, Right Turn Only’, Do Not Enter’, ’Crosswalk’, ’Handicap Parking’, ’No Parking’. They were labeled numerically from 0-9 respectively. The total collected data set from each student was merged and randomly partitioned into a training set (approx. 70%) and an easy test set (approx. 30%). The training set was used to fit the model created. The easy test set is considered a blind set and will used to test the model. There is an additional hard test set that will contain part of the collected dataset and some random non traffic sign images. This will be used to validate the robustness of the model.

Data Prepartion

The data test set along with the label test set was imported into the algorithm. The data was reshaped into 300x300x3 for RGB images. After several test runs of the algorithm it was deemed necessary to resize the images to 150x150. The data was then processed so that it would be in the form for the model.

```
tf.keras.applications.inception_v3.preprocess_input(data_train, data_format=None)
```

The data was then split into training and test sets with the sklearn function train_test_split with an 80% to 20% division.

There was an additional step of using one-hot encoding technique on the labels. This is used to force the output to be an integer. This will help the model when confused between two classes, it will pick the class that returns a smaller error. The use of one hot encoding as opposed to integer encoding is that there are more than two classes in the dataset.

Lastly, the train data images were randomly augmented in real time during the training using the image data generator. The transformations that were chosen were horizontal flip, vertical flip, 90 degree and a reduction of brightness up to 25%.

```
train_datagen = ImageDataGenerator(horizontal_flip=True,
                                   vertical_flip=True,
                                   rotation_range=90,
                                   brightness_range=(.75, 1))
```

This helps with the robustness of the model and saves on overhead memory. The resulting data is then put into a predefined batch size. The training data is now ready to be used in the model.

InceptionV3 Model Architecture

The Inception-v3 model was released in 2015 and was shown to have a lower error rate than its predecessors. It has three parts: the basic convolutional block, Inception module and the classifier. The convolutional block alternates convolutional with max-pooling layers and is used for feature extraction. Max-pooling produces the maximum pixel value over the non-lapping region of the weighted window. The Inception module is designed as a Network-In-Network where multi-scale convolutions are conducted in parallel and the results of each branch are concatenated. Finally, the classifier adds more training stability and gradient convergence. As a result, the issues of overfitting and vanishing gradient are alleviated [7]. The model consists of 42 layers which are shown in Fig 1. The convolutional kernel is widely used to reduce the number of feature channels and accelerate the training speed. By having the large convolution decomposed into smaller convolutions this reduces the number of parameters and computation expense.

Pre-Processing the Model

Now all the parameters need to be set for the model prior to training the model. When using the InceptionV3 model the input shape needs to be set so it matches the input data. The training dataset needs to be set at (150, 150, 3) for an 150x150 RGB image.

```
inception = inceptionV3(input_shape=(150,150,3),
                        include_top=False,
                        weights='imagenet')
```

When using the InceptionV3 model the input shape needs to be set so it matches the input data. This training dataset needs to be set at (150, 150, 3) for a 150x150 RGB image. The top layer needs be excluded for feature extraction by setting the top layers to False.

```
for layer in inception.layers:
    layer.trainable = False
```

This command will freeze the base model layers so that they are not updated during the training process.

Next the final output layer of the model needs to be defined

```
x = tf.keras.layers.Flatten()(inception.output)
x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.Dropout(0.15)(x)
x = tf.keras.layers.Dense(10, activation='softmax')(x)
```

As there are many hidden layers within the model there are techniques in which to prune the model to prevent overfitting.

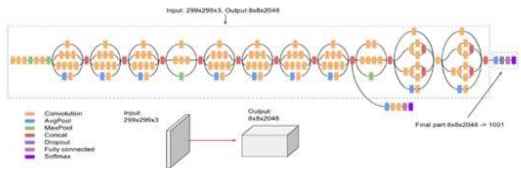

Figure 1: Inception-v3 Model Architecture

Dropout is one of those parameters used to inactivate some of the weight connections and discard them. The selection of hyperparameters helps with the vanishing gradient problem. In this model the activation function used is the Rectified Linear Unit (ReLu). The ReLu is a non-negative piecewise function that obtains the maximum value between zero and one. Since one-hot encoding is used, the softmax activation function is used at the output layers to have confidence for each decision. The softmax normalizes the output and assigns a probability to each output.

Now the model needs to be finetuned with an optimizer and recompiled with the following command.

```
model.compile(optimizer = optimizer,
              loss = 'categorical_crossentropy',
              metrics = ['categorical_accuracy', 'acc','mean_squared_error'])
```

The best optimizer for the model was found to be the Stochastic Gradient Descent (SGD). This optimizer defines a loss function that expresses how well the weights and biases allow the network to fit the training data. The network parameters can then be adjusted to minimize the loss. Once the direction of the greatest loss is determined, then it is known to move in the opposite direction and at a distance determined by the learning rate. The Nesterov Momentum computes the gradient term not from the current position but instead from an intermediate position. This helps because if the momentum term points in the wrong direction or overshoots then the gradient can correct it in the same update step.

```
SGD = SGD(learning_rate=0.01, nesterov=True)
optimizer = SGD
```

Training the Model

The model is trained with the randomly augmented train data. The number of epochs was chosen with a maximum of 150 and PATIENCE of 15 along with early stopping. The batch size can be varied but for the training it was set to 64. The trained model was saved to the H5_OUTPUT Filename. For more information on the how to utilize the trained model refer to the README file.

EXPERIMENTS

Feature Extraction

Feature extraction is needed to help with dimensionality reduction. There are several methods that can be applied such as K-Nearest Neighbor (KNN), Naïve Bayes and Support Vector Machine (SVM). Each of these were explored in the Short Assignment 4. After comparing the results from that assignment, it was determined that designing and optimizing those feature extraction techniques would be time consuming. Table 1 shows that each one would need a lot of optimizing to get higher percentage of accuracy. Given the project completion date along with the loss of a team member, it was deemed prudent to take advantage of the transfer learning method and use Inception-v3 pretrained model for feature extraction.

Table 1: Feature Extraction Accuracy for various classifiers

Feature Extraction Improvements in Accuracy			
	KNN	Naive Bayes	SVM
W/O FE	53.294	48.264	24.219
With FE	64.968	52.529	63.285
% Change	17.979	8.119	61.730

Traditonal Machine Learning vs Transfer Learning

Traditional Machine Learning models need to be developed from scratch while transferred learning can take a pre-trained model and transfer the knowledge to a new dataset. Traditional Machine Learning models are dependent on sufficient labelled data for a specific domain, require feature extraction, optimize through hyperparameter tuning and require lots of training data and time. On the other hand, transfer learning with its pre trained model reduces the training time and requires less data to train to increase performance. In the Inception-v3 model, there is flexibility to "freeze' all the layers to use the pre-trained weights as is or layers can be retrained as needed to increase performance. As shown in Table 2, the model training accuracy for the Inception-v3 is over 90% without optimization while the other non-optimized traditional machine learning models had inferior accuracy results.

Table 2: Model Training Accuracy

Model Training Accuracy on Feature Extracted Data			
KNN	**Naive Bayes**	**SVM**	**InceptionV3**
64.968	**52.529**	**63.285**	**93.45**

Choosing the best optimizer for the model

There are several optimizers that can be used in the Inception-v3 model. The ones that were tested were the Adaptive Moment Estimation (ADAM), Nestrov Adaptive Moment Estimation (NADAM) and the SGD. SGD maintains a single learning rate and that rate does not change during training. ADAM maintains a per parameter learning rate and performs adaptive learning on not only the average first moment but also on the second. NADAM is the ADAM with Nestrov's momentum formula. Using GridSearch to determine the best optimizer for the model. It was determined that the SGD with Nestrov momentum was the optimal solution with a learning rate of 0.01.

Image Augmentation

As mentioned earlier, image data augmentation was used in the model through the image generator function This function is designed to provide real-time data augmentation during the training stage. A variety of different parameters were implemented and tested using the Image Data module from TensorFlow. As filters were implemented the CNN was trained against them and a baseline accuracy and confidence rating was assessed. It was discovered that each augmentation inserted slightly improved accuracy but greatly improved the softmax confidence. With exception with the brightness reduction augmentation, this filter initially caused a 23% reduction in accuracy until the parameters were adjusted accordingly. The final implementation of image augmentation changed the accuracy of the CNN from 88.97% to the current 95.72%.

Image Size Reduction

Given the original images are 300x300x3 the CNN was originally trained against these images. It was found that the model trained extremely slowly even with GPU acceleration. This lead to an optimization problem where reducing the amount of data from the photo vs losing spatial resolution which had to be balanced. A grid search across resolution sizes was conducted using 5 epochs to gain an approximation of time per epoch and model accuracy. It was determined that the resolution of 150x150 still contained most of the relevant information for the model and cut training time by a magnitude at ~21s/epoch vs 232s/epoch for the 300x300 images.

Early Stopping

The early stopping parameter is needed to stop the training model when there is no continuous improvement as it chases the loss function by tuning the parameters. This can cause overtraining. After multiple experiments it was found that 15 was the most optimal value to prevent overtraining and to ensure that the model had enough leeway to find the least loss solution.

Dropout

As mentioned before, dropout is a parameter used to inactivate some of the weight connections and discard them when training the model. This aids in preventing overfitting of the model. The dropout ratio is in between 1 and 0. The rate is defined as the parameter which determines the odds of dropping out neurons. To optimize the best dropout rate, several training runs were performed and in Fig 4 it shows that the dropout rate of 0.15 was the best to limit overfitting while 0.05 was prone to overfit.

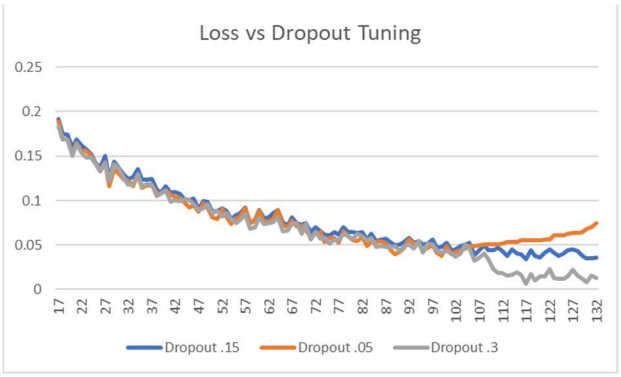

Figure 4: Loss versus Dropout

EXPERIMENTAL RESULTS

Final Model Results

The parameters of the final model are as follows, SGD with Nesterov, with a learning rate of 0.01, a Dropout rate of .15, a patience value of 15. The model was trained using the Early Stopping method while checking for the minimum loss. Fig 5 shows the Normalized Loss, MSE and Accuracy trends over the lifetime of the training session.

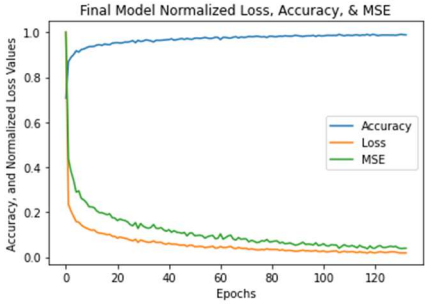

Figure 5: Final Model Statistics

Overfitting is a significant issue with training CNNs, and this was countered in numerous ways. By introducing tuned Dropout and patience rates it can be observed that the model stopped before experiencing signs of overfitting.

Easy Test Evaluation

The final Easy Test evaluation was performed on the leftover 20% of the provided training data. This data had no training performed on it and had no image augmentation except for the resizing to 150x150 and the InceptionV3 preprocessing. Fig 6 demonstrates the Confusion Matrix of the label predictions performed on the Easy Test set.

Some areas of confusion that are noteworthy is the Green Light/Red Light mis-predictions and the Roundabout/ Right Turn Only mis-predictions. The mis-predictions of the lights were expected, and over various iterations was reduced by implementing the brightness data augmentation. The test run accuracy on this Easy Test data set was evaluated to be 98.50%.

Hard Test Evaluation:

The final model depends on the SoftMax confidence levels to determine whether an object is "unknown". The model was trained on what has been deemed a sufficiently difficult dataset. So to prepare the model to evaluate the Hard Test the following tests were conducted. A confidence threshold was analytically determined by analyzing the model's confidence at predicting the Easy Test samples. This confidence threshold is then used to determine if the model should return the label as unknown or as the max prediction. Fig 7 demonstrates the rate at which this threshold would mis-predict. The Confusion matrix also represents the classes in which the model has on average lower confidence/ more prone to misclassification. An example of a low confidence prediction can be seen in Fig 7.

Upon testing the performance of the classification with a threshold an accuracy of 95.72% was obtained. The test dataset did not contain "Unknown" class items, but this test served to demonstrate that little performance was lost and that the model was confident throughout as shown in Fig 8.

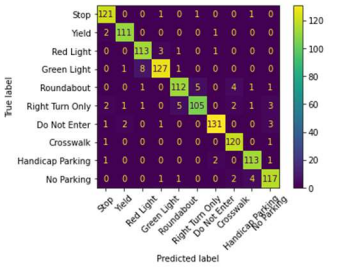

Figure 6: Confusion Matrix Easy Test

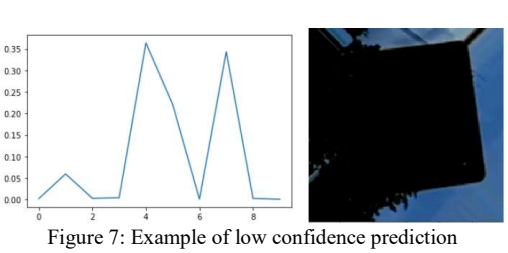

Figure 7: Example of low confidence prediction

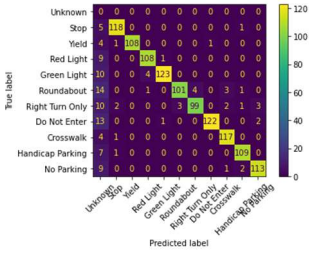

Figure 8: Confusion Matrix with Unknown's

CONCLUSION

During this project it became apparent that the transfer learning method is a very powerful tool for classification. It saves a lot of experimental time and trials. The original approach relied heavily upon feature extraction and traditional ML models but the availability of pretrained CNN's as well as their popularity in image classification influenced the CNN being selected. Analyzing the confidence values of the One-Hot results provided interesting insights into what features ended up being linked to what outputs. An area for future parameter tuning and training would be to unfreeze desirable sections of the InceptionV3 net and to train the model against Unknown data. This final project has enabled the authors to experiment and test their understanding of a wide range of topics from the course.

REFERENCES

[1]. Krithga R, Usha G, Raju N and Narasimhan"Transfer Learning Based Breat Cancer Clasiifcation using One-Hot Encoding" ICAIS , vol. A247, , 2021.

[2]. TensorFlow,"Transfer Learning and fine tuning" https://www.tensorflow.org/tutorials/images/transfer_learning (accessed July 11, 2022).

[3]. Neptune Blog ," Guide: A Practical Tutorial With Examples for Images and Text in Keras'' https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras (accessed July 14, 2022).

[4]. fchollet, "Transfer Learning & fine tuning" . https://keras.io/guides/transfer_learning/ (accessed July 12, 2022).

[5]. J. Brownlee, "Transfer Learning in Keras with Computer Vision Models" https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/ (accessed July 14, 2022).

[6]. liferlisiqi, "Traffic-Sign-Classifier" https://github.com/liferlisiqi/Traffic-Sign-Classifier (accessed July 8, 2022).

[7]. Lin,C, Lin,L,Wang,K,Guo,J,"Transfer Learning Based Traffic Sign Recognition Uisng Ineception-V3 Model " Periodica Polytechnica Transportation Engineering , 2019.

[8].Zhao,C, Zhang,W,"Fast Traffic Sign Recognition Algorithm Based on Mulit-scale Convoltuional Neural Network" Eight Annual Intl Conf on CBD,2020.

[9]. " Inception V3 Model Architecture" Inception V3 Model Architecture (opengenus.org) (accessed July 27 2022)

[10]. S.Sarin, "Exlploring Data Augmentation with Keras and TensorFlow" https://towardsdatascience.com/exploring-image-data-augmentation-with-keras-and-tensorflow-a8162d89b844 (accessed July 13 2022) .

[11]. T.Irla, "Transfer Learning using Inception-v3 for Image Classification" https://medium.com/analytics-vidhya/transfer-learning-using-inception-v3-for-image-classification-86700411251b (accessed July 15,2022).