Traffic Signs Image Classification Using Transfer Learning with Xception CNN Model

Matthew Cohen
Electrical and Computer Engineering
cohen.matthew@ufl.edu

Gloria Katuka
Computer & Information Science and Engineering
gkatuka@ufl.edu

Alex Perez
Electrical and Computer Engineering
alex.perez@ufl.edu

Ruijie Wang
Electrical and Computer Engineering
ruijiewang@ufl.edu

***Abstract*—Convolutional neural network (CNN) models have been widely applied to image classification tasks such as image recognition, object or obstacle detection and other practical applications of computer vision. CNN models have achieved impressive performance in several image classification tasks, which makes it suitable for the specific task of classifying traffic sign images. In summer 2022, the final project for the Fundamental of Machine Learning (FML) course at the University of Florida requires students to create and classify a dataset of traffic signs images. In this report, we describe the experimental process of classifying the dataset of traffic sign images with 10 classes. The goal was to achieve a performance accuracy of 90% or higher. After experimenting with several machine learning classifiers such as k-Nearest Neighbor(kNN), Support Vector Machines(SVM) and CNN, we achieved the best performance of 93% using transfer learning with the Xception convolutional neural network model. Additionally, we added a probability threshold to the model to classify unknown classes and achieved a performance accuracy of 83%. The results further reveals the wide-ranging applications of CNN to image classification tasks, particularly using transfer learning.**

***Index Terms*—convolutional neural networks, machine learning, classification, Xception model**

I. Introduction

In recent years, the popularity of image classifications tasks, such as image recognition, object or obstacle detection and other practical applications of computer vision had lead to the increasing applications of machine learning techniques and models, particularly deep learning techniques and models [1]. Convolutional neural networks (CNN or ConvNet), a special architecture for deep learning, has been found to achieve impressive performance in image classification tasks. CNN has been implemented on common image datasets such as MNIST [2] and Fashion-mnist [3]. Recently, Kayed et al. experimented on classifying Fashion-mnist dataset and found that CNNs outperformed other classifiers with the CNN LeNet-5 model achieving over 98% accuracy [4]. In class, we also practiced applying CNN to the Fashion-mnist dataset, a commonly used image dataset consisting of images of different clothes and their labels. This makes CNNs most suitable for our image classification task.

In summer 2022, the final project for the Fundamental of Machine Learning (FML) course at the University of Florida requires students to create and classify a dataset of traffic signs images. All students individually collected 10 images of 10 traffic signs images 1. Students were provided a subset of the dataset to train a machine learning model with the goal of achieving an accuracy score of 90% or higher. In this report, we discuss the experimental process of training a machine learning model for traffic sign image classification. We experimented the training dataset on several models. For this report, we will analyzed the results from the models generated using kNN, SVM and CNN. With the tremendous success of CNNs in image classifications, we decided to to apply transfer learning using a CNN with Xception model for the final project. Transfer learning is a machine learning technique whereby an existing neural network trained for one task is used to accomplish a similar task [5]. Particularly, if you have a small training set or limited computational resources. Keeping that in mind, we chose to implement the Xception pre-trained model on ImageNet, available with Keras, to the traffic sign image classification tasks. The results shows that the Xception model outperforms other CNN models as well as traditional machine approaches.

II. Implementation

A CNN consists of an input layer, multiple hidden layers (convolutional layers), and an output layer. Each layer consists of some features or characteristics of the image, partitioning the classification process. As such, the CNN extracts and attempts to match some features and not all, and used the features extracted from one layer as the input for the next layer. Backpropagation is used to obtain the weights and values using the error in the output layers to determine how the learning algorithm is doing and what changes are needed. For this project, we used a pre-trained model, Xception, trained by a large dataset to classify our traffic signs dataset. By doing so, we reduce the resource and computational cost for our classification task. Xception, which stands for Extreme inception, consists of a linear stack of depth-wise separable

convolution layers with the residual connection, which reduces the convolution computation cost.

A. Data Preprocessing

[] The dataset of traffic sign images were collected by students enrolled in the course. Each student was tasked with collecting 10 images for each traffic sign. After the images were collected, each project group received the same training dataset of 6195 images. Each label and sample image is shown in 1. The training dataset provided consisted of two numpy files, one for the images and the other for the labels. The first thing we did was to upload the dataset using $np.load('data\_train.npy')$ and $np.load('labels\_train.npy')$, respectively. Initial review of data showed that some of the images were mislabelled and contained noise. We manually cleaned some of the data by fixing mislabelled images, then we experimented on several pre-processing techniques.

Fig. 1. Sample Image from Traffic Sign dataset

To implement the Xception CNN model, the pre-processing steps we applied included: First, we used casting to convert the numpy arrays for the images into tensorflow tensors, using

```
data_train =data_train.reshape((data_train.shape
    [0],300,300,3)
data_train_rs = tf.constant(data_train),dtype=tf.
    float32).
```

Listing 1. Reshaping the training datasetS

We performed data augmentation on the dataset, which included resizing, rescaling, random flipping, and random rotations. This would allow us to push the model to truly detect the unique features describing the signs as well as increased the size of our dataset. We kept the image sizes as 300 X 300 to preserve all the pixels(information). Although keeping the image at 300 X 300 might cause overfitting and increase training time, it can also increase the accuracy of prediction.

```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.Resizing(IMG_SIZE, IMG_SIZE),
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.RandomFlip('
    horizontal_and_vertical'),
    tf.keras.layers.RandomRotation(0.2)
])
```

Listing 2. Reshaping the training datasetS

We used additional pre-processing techniques for experiementing with other models. For instance, we used Min-Max Scaling, StandardScaler, and other features extraction techniques like PCA, LDA for SVM and KNN models. CNN models automatically extract features through their convlutional layers.

B. Model Selection

After splitting the data into training, validation, and testing sets, we implemented several machine learning models before arriving at the final model with the best performance for the training dataset. Some of models we attempted include: a Naive Bayes classifier, a Support Vector Machine (SVM) classifier, a k-Nearest Neighbor classifier and a Linear Perceptron classifier. After going through these traditional machine learning approaches and analyzing the results, we were certain that we needed to implement a neural network to achieve our goal performance. With that in mind, we implemented various types of convolutional neural networks before deciding on the pre-trained Xception CNN model as the best model for this image classification task. We used the Xception model from Keras as the base model, then used global average pooling and added an output layer containing 10 neurons with a soft max activation function. The final model was compiled and fit using 10 epochs. The model describing the epoch that performed the best in validation was then chosen to predict the training and test sets. Fig 2 shows the model summary for the final version of Xception model.

```
base_model = keras.applications.Xception(
                weights='imagenet',
                input_shape=(300,300,3),
                include_top= False)
#Functional API
IMG_SIZE=300

inputs = keras.layers.Input(shape=(300,300,3))
data_augmented  = data_augmentation(inputs)
x = base_model(data_augmented, training=False)
avg_pool = keras.layers.GlobalAveragePooling2D()(x)
outputs = keras.layers.Dense(10, activation='softmax
    ')(avg_pool)
model = keras.Model(inputs, outputs)
base_model.trainable = False
```

Listing 3. Initializing the Xception CNN model

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 300, 300, 3)]     0

 sequential_2 (Sequential)   (None, 300, 300, 3)       0

 xception (Functional)       (None, 10, 10, 2048)      20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 10)                20490

=================================================================
```

Fig. 2. Xception Model Summary

C. Performance Evaluation

To determine the best model for this classification task, we performed several experiments and evaluated the model

performance based on accuracy. Model accuracy is defined as the ratio of correct predictions to the total number of samples. We also analyzed the precision, recall and F1-scores of the Xception model for each class. The precision is performance of detection, also known as the positive preodictive value (PPV). The recall is the probability of detection or the true positive rate (TPR). The F1-score is a measure of accuracy based on the model's precision and recall. This means that the F1-score will be high, if the precision and recall are both high. In the next section, we analyze the first four classifiers: NB, SVM, kNN and MLP, three CNNs models including the final model chosen for the traffic sign dataset.

III. Experiments

For the experiments, we used two versions of the dataset: the raw data with some misclassified labels and some noise and a cleaner version, which we first manually cleaned by updating the appropraite labels, and automatically processed by resizing and data enhancements. We used Jupiter Notebook to implement the project. We first imported all libraries required from Numpy, Matplotlib, Scikit-learn, Tensorflow, and Keras. Then we pro-processed the data as described in SectionII-A. Next, we split the training dataset into training, validation and test sets. We applied data enhancement techniques such as reshaping, rescaling and transformations , and used feature extraction and dimensional reduction techniques such as principal component analysis (PCA) Then, we performed the following experiments and analyzed the results.

A. Guassian Naive Bayes

The Guassian Naive Bayes classifier is one of the simplest classification models to create and one of the easiest to understand. For this experiment, Principal Component Analysis (PCA) was used to find the minimum number of principal components that could describe majority of the training data. We used GridSearchCV to test various "var smoothing" hyperparameters, and the number of PCA components that that needed to explain 90% of variance in the dataset. To run the PCA, we first used MinMax Scaler to scale the data, then we determine the number of PCA components, reducing the dimension. Then we used the number of components derived as inputs for the Guassian Naive Bayes classifier.

B. Support vector machine (SVM)

For the Support Vector Machine classifier, we also used GridSearchCV fine-tune the hyperparameters "C" and "kernel", where "C" is the regularization parameter, and "kernel" is the type of kernel that will be used for the algorithm ("RBF" kernel is set by default). After conducting the grid search to find the best parameters, we found that "C" = 1 and kernel = 'linear' yielded the best accuracy. The best model was then taken from the GridSearchCV and used on a test set.

C. k-Nearest Neighbor (KNN)

The k-Nearest Neighbor, kNN is another fairly straightforward classifier that simply takes the number of neighbors, finds the k nearest neighbors and assign the class based on majority vote. The kNN model uses the Euclidean distance metric because it normalizes the distances. We also applied the PCA using the pipeline method, and experimented with n_neighbors of 2,3,5 and 10. and found n_neighbors = 5 to be the best estimator. kNN is unlike other classifier because a trained model is not produced. As such, kNN classification will require greater computational resources for large datasets.

D. Linear Perceptron (LP)

For the Linear Perception classifier, similar to the previous two, we used GridSearchCV to test various combinations of the "penalty" and "alpha" hyperparameters. The "penalty" hyperparameter refers to the type of regularization being used on the classifier (l1, l2, etc.), and the "alpha" hyperparameter refers to the constant that multiplies the regularization term. Like before, the best set of hyperparameters were chosen for the test set.

E. Convolutional Neural Network (CNN)

We experimented on a CNN model to establish a base model for the image classification. The input image consisted of the images with "input_size" of (300,300,3). The first hidden layers takes the input and we apply batch normalization with axis = -1 and a "ReLu" activation function to each layer. MaxPooling is added after tow layers with a pool_size of (2,2), then, flattening is introduced. Next Dropout is introduced and the dense later. Lastly, softmax is applied to present teh result in terms of the 10 classes. We used a batch size of 32, 15 epochs, Adam optimizer. The summary of the CNN architecture is as follows:

$$Input \longrightarrow Conv\_layer1 \longrightarrow BatchNorm \longrightarrow ReLu \longrightarrow Conv\_layer2 \longrightarrow BatchNorm \longrightarrow ReLu \longrightarrow MaxPooling2D \longrightarrow Conv\_layer3 \longrightarrow BatchNorm \longrightarrow ReLu \longrightarrow Conv\_layer4 \longrightarrow BatchNorm \longrightarrow ReLu \longrightarrow MaxPooling2D \longrightarrow Flatten \longrightarrow Dense \longrightarrow BatchNorm \longrightarrow ReLu \longrightarrow Dropout \longrightarrow Dense \longrightarrow Softmax \longrightarrow Output$$

F. CNNs + AlexNet

We made multiple attempts to create and train convolutional neural networks of our own design. The goal of these experiments was to determine how effective a CNN we designed could be. We wanted to see if we could exceed 90% accuracy with a model of this form. The first designs we experimented with were keras models that consisted of a series convolutional layers followed by pooling layers with dense layers at the top of the network. We tested a few different combinations of models similar to this training them on the original unmodified training data for 15-30 epochs. With these models we were only able to achieve at most 90% accuracy on the validation set. Generally, after about 15 epochs these models tended to overfit. We tried adding dropout layers near the top of these models but were unable to find a set of layers that resulted in a model that was able to train and not overfit.

We also attempted to train models using the architecture of AlexNet. These models also suffered from the same issues

as the CNNs above. They were able to achieve about 90% validation accuracy but were unable to improve beyond that without overfitting. Adding the 50% dropout layers that were part of the original AlexNet model caused the model to not improve with training and lower dropout rates did not cause the model to perform any better.

G. Transfer Learning with Xception

For this experiment, we used the pre-trained Xception model, a model that is part of the Keras built in functions that was developed at Cornell University for use in image classification tasks. In this experiment we simply created a sequential Keras model that consisted of an input layer, the Xception model, and an output layer that fit our classification task (a dense layer of 10 neurons). We found that after just a small amount of training (10 epochs) this model was able to outperform any of our other experiments, reaching an accuracy of 93%.

TABLE I
BEST ACCURACY SCORES FOR EACH MODEL

	Performance Measure	
Classifier	***hyperparameters***	***Test accuracy***
GuassianNB + PCA	n_components = 764	31.4%
kNN	n_neighbors = 5	64%
kNN + PCA	n_components = 764, n_neighbors = 5	64%
SVM	C = 1, kernel= 'linear'	72%
LP	α = 0.001, penalty = "l1"	70.7%
CNN	epochs = 15, batch_size = 32	90%
CNN (AlexNet)	epochs = 15, batch_size = 32	92%
CNN (Xception)	epochs = 20, batch_size = 32	92%

The results shows that CNN with Xception and AlexNet models outperforms the other models on the traffic signs dataset. We chose the Xception model for this project. The model also provides a relatively high classification results for individual class labels.

TABLE II
CLASSIFICATION ACCURACY OF EACH TRAFFIC SIGN FOR THE XCEPTION CNN MODEL

Label	**Traffic sign**	***Precision***	***Recall***	***F1-Score***
0	Stop	0.95	0.96	0.95
1	Yield	0.95	0.98	0.96
2	Red Light	0.91	0.91	0.91
3	Green Light	0.90	0.91	0.90
4	Roundabout	0.89	0.90	0.90
5	Right Turn Only	0.87	0.94	0.90
6	Do Not Enter	0.97	0.94	0.96
7	Crosswalk	0.97	0.88	0.92
8	Handicap Parking	0.98	0.94	0.93
9	No Parking	0.92	0.94	0.93

H. Classifying Unknown label - Bonus

We also designed a model to classify unknown labels. Initially, we explored the possibility of adding another class, class "11" to the dataset, then create a layer to collect probabilities.We pivoted and added a conditional model outside the model to judge.Based on the series of output probability values, we finally set our threshold to 0.95. The value of the threshold determine the probability of classifying an image to a class. If it is greater than 0.95, we still use the previous method to determine the class to which the image belongs. For some images, they don't have the same features as our training set, which will cause them not to tend to any one class. Ideally, they should have a uniform probability of corresponding to these 10 classes. The disadvantage of this approach is that if there are some images that are very similar to our training set, it may be judged to belong to that similar group. but in fact it is known class image.

IV. CONCLUSION

From this project we are able to conclude that convolutional neural networks are by far one of the most effective ways to perform an image classification tasks like the one undertaken in this project. Even our less effective or simpler CNN models outperformed our other model attempts, some of which took a lot more work to create and train. Additionally, we have learned that transfer learning is a very quick and effective way to train models. The models we trained via transfer learning were quicker to train and more effective than any of the CNN models that we designed and trained from nothing. In summary we can conclude that by applying transfer learning with the use of preexisting libraries and CNN models it is possible to quickly and easily train a machine learning model that can perform image classification tasks with accuracy greater than 90%.

REFERENCES

[1] P. Druzhkov and V. Kustikova, "A survey of deep learning methods and software tools for image classification and object detection," *Pattern Recognition and Image Analysis*, vol. 26, no. 1, pp. 9–15, 2016.

[2] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[3] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[4] M. Kayed, A. Anter, and H. Mohamed, "Classification of garments from fashion mnist dataset using cnn lenet-5 architecture," in *2020 international conference on innovative trends in communication and computer engineering (ITCE)*. IEEE, 2020, pp. 238–243.

[5] Y. Gao and K. M. Mosalam, "Deep transfer learning for image-based structural damage recognition," *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 9, pp. 748–768, 2018.