Evaluation of CNNs on Graffitied Traffic Signs Classification

Yanqin Wu (yw4359)
Kayan Shih (ks5250)

Executive Summary

Goal:

The objective of this project is to test and compare different CNN model architectures' accuracy and robustness in identifying different types of **Graffitied** traffic signs.

Solution Approach:

LeNet5, ResNet50, VGG16, inceptionV3

Benefit:

In real life situations, some traffic signs that we see on the streets are not in perfect conditions: they are sometimes damaged or scribbled on. In this case, finding the most accurate model is just as vital as finding the most robust one.

Problem Motivation

Within the skyrocketing industry of self-driving smart car, safety has always been an essential concern that all car manufacturer and innovator need to address while updating vehicle's other attributes such as speed and energy consumption. On this topic, abiding to the traffic signs is a significant component. However, in real life, some of the traffic signs are often damaged or doodled which may make them unrecognizable or wrongly identified on traffic sign classification model that were trained on only the perfect-condition signs. Therefore, it's vital for all self-driving vehicles to be able to identify traffic signs robustly and consistently.

Background Works

https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Zhu_Traffic-Sign_Detection_and_CVPR_2016_paper.pdf

1. Traffic-Sign Detection and Classification in the Wild

https://www.researchgate.net/publication/352393724_Traffic_Sign_Classification_and_Detection_of_Indian_Traffic_Signs_using_Deep_Learning

2. Traffic Sign Classification and Detection of Indian Traffic Signs using Deep Learning

https://ieeexplore.ieee.org/document/9478172

3. Traffic Signs Classification using Convolutional Neural Networks: A review

https://arxiv.org/abs/2209.15251

4. Traffic Sign Classification Using Deep and Quantum Neural Networks

Technical Challenges

- **The first challenge of this project is that there are many variations in the training and testing traffic image data**. The images of the same type of traffic sign may have drastically different colors, dimensions, and resolutions. Furthermore, the images could also be taken from many different angles, which may lead to a certain level of distortion of the object. Issues like these may pose significant challenges in consistently classifying images.
- **The second challenge of this project is that not all classes have the same number of images.** Some of the traffic signs are under-represented. For those classes, we can find additional images and add them to the dataset
- **The third challenge is to manually graffiti images.** During the process, we can also use different techniques to modify the image such as horizontal flip and color change.

Approach:

- Compare LeNet-5, ResNet-50, VGG-16, and InceptionV3's performance in classifying various traffic signs on the traffic Sign Dataset to select the best model.
- Add noise to the image to resemble real-life situation where the signs are sometimes scribbled or damaged. Then, we measure the performance of previous trained models to see which model has the best robustness.
- Use Transfer Learning to see how much performance can be boosted if model had been trained with real-life image data

Dataset and Sample images

The dataset that we used contains 58 different types of traffic signs (classes) , and there is an average of 71 images per class (total of 4170 samples). The test and validation split is 80% train, 20% validation.






Solution Diagram/Architecture

LeNet-5

Model: "sequential_14"

Layer (type)	Output Shape	Param #
sequential_13 (Sequential)	(None, 224, 224, 3)	0
rescaling_11 (Rescaling)	(None, 224, 224, 3)	0
conv2d_196 (Conv2D)	(None, 222, 222, 6)	168
average_pooling2d_26 (AveragePooling2D)	(None, 111, 111, 6)	0
conv2d_197 (Conv2D)	(None, 109, 109, 16)	880
average_pooling2d_27 (AveragePooling2D)	(None, 54, 54, 16)	0
flatten_4 (Flatten)	(None, 46656)	0
dense_24 (Dense)	(None, 120)	5598840
dense_25 (Dense)	(None, 84)	10164
dense_26 (Dense)	(None, 58)	4930

Total params: 5,614,982
Trainable params: 5,614,982
Non-trainable params: 0

ResNet50

Model: "sequential_15"

Layer (type)	Output Shape	Param #
sequential_13 (Sequential)	(None, 224, 224, 3)	0
rescaling_12 (Rescaling)	(None, 224, 224, 3)	0
resnet50 (Functional)	(None, 7, 7, 2048)	23587712
global_average_pooling2d_6 (GlobalAveragePooling2D)	(None, 2048)	0
dropout_6 (Dropout)	(None, 2048)	0
dense_27 (Dense)	(None, 1024)	2098176
dense_28 (Dense)	(None, 58)	59450

Total params: 25,745,338
Trainable params: 2,157,626
Non-trainable params: 23,587,712

Solution Diagram/Architecture

VGG-16

Model: "sequential_16"

Layer (type)	Output Shape	Param #
sequential_13 (Sequential)	(None, 224, 224, 3)	0
rescaling_13 (Rescaling)	(None, 224, 224, 3)	0
vgg16 (Functional)	(None, 7, 7, 512)	14714688
global_average_pooling2d_7 (GlobalAveragePooling2D)	(None, 512)	0
dropout_7 (Dropout)	(None, 512)	0
dense_29 (Dense)	(None, 1024)	525312
dense_30 (Dense)	(None, 58)	59450

Total params: 15,299,450
Trainable params: 584,762
Non-trainable params: 14,714,688

InceptionV3

Model: "sequential_17"

Layer (type)	Output Shape	Param #
sequential_13 (Sequential)	(None, 224, 224, 3)	0
rescaling_14 (Rescaling)	(None, 224, 224, 3)	0
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
global_average_pooling2d_8 (GlobalAveragePooling2D)	(None, 2048)	0
dropout_8 (Dropout)	(None, 2048)	0
dense_31 (Dense)	(None, 1024)	2098176
dense_32 (Dense)	(None, 58)	59450

Total params: 23,960,410
Trainable params: 2,157,626
Non-trainable params: 21,802,784

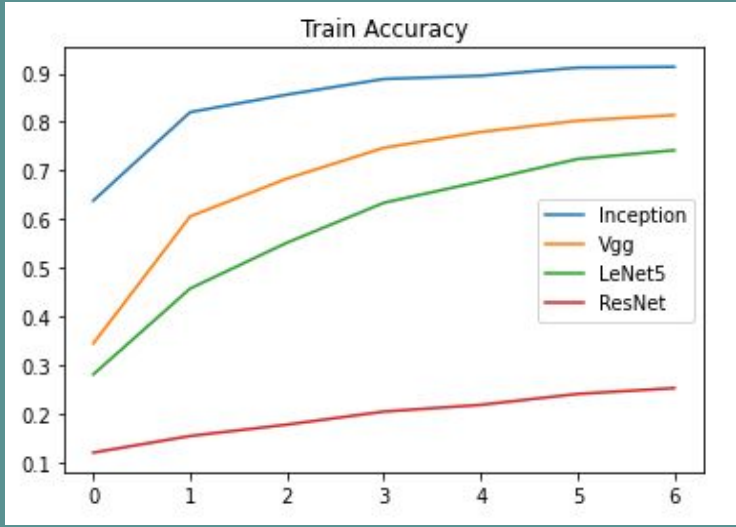
Demo/Experiment Design Flow

For each of the model:

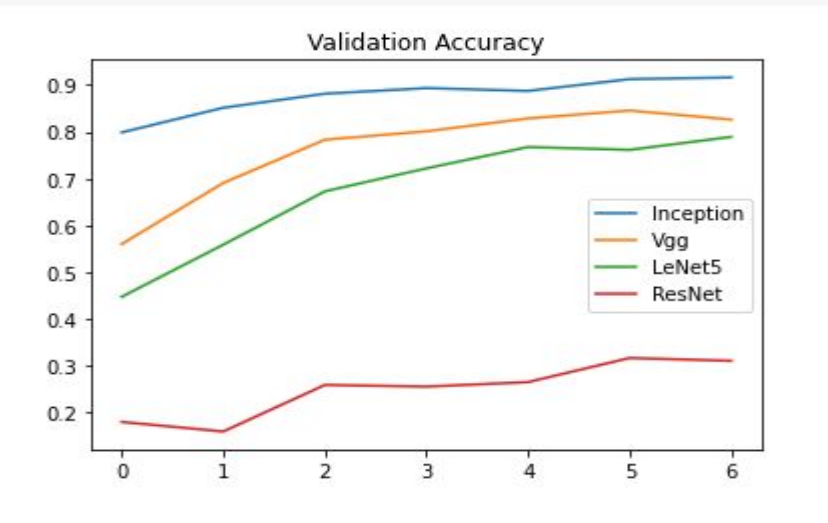
1. Data augmentation: Random horizontal and vertical flip, Random Zoom, and Random Rotation
2. Adjust input shape to (224,224,3)
3. Rescaling: 1/255
4. For different model add their according layers
5. Softmax function for 58 classes
6. Calculate loss using Sparse Categorical Cross entropy and Adam optimizer
7. Compile
8. FIt
9. Evaluate

Experiment Evaluation

Train Accuracy



Validation Accuracy



Modified Dataset

```
for dir in os.listdir(path):
    if dir != ".DS_Store" and dir !="image_change.ipynb":
        for image in os.listdir(path+dir):
                if image.endswith(".png"):
                    im = Image.open(path+dir+"/"+image)
                    img_arr = np.asarray(im)
                    count = np.random.randint(10)
                    width = np.random.randint(len(img_arr[0])/5)
                    length = np.random.randint(len(img_arr))
                    for run in range(count):
                        x = np.random.randint(len(img_arr[0]))
                        y = np.random.randint(len(img_arr))
                        r = np.random.randint(255)
                        g = np.random.randint(255)
                        b = np.random.randint(255)
                        for i in range(width):
                            if x+i >= len(img_arr[0]):
                                break
                            for j in range(length):
                                if y+j >= len(img_arr):
                                    break
                                img_arr[y+j][x+i]=[r,g,b]
                    im = Image.fromarray(img_arr)
                    im.save(path+dir+"/"+image)
                    print("Image Changed")
```

Randomly Changed the color of specific pixels in random regions

Experiment Evaluation for modified dataset

LeNet5:

```
[166] LeNet5_eval_original = LeNet_model.evaluate(val_ds)
27/27 [==============================] - 0s 11ms/step - loss: 0.9827 - accuracy: 0.6847

LeNet5_eval_graf = LeNet_model.evaluate(m_val_ds)
27/27 [==============================] - 0s 11ms/step - loss: 3.3008 - accuracy: 0.5156
```

ResNet50:

```
[168] res_model_original = res_model.evaluate(val_ds)
27/27 [==============================] - 1s 25ms/step - loss: 2.8376 - accuracy: 0.2494

[169] res_model_graf = res_model.evaluate(m_val_ds)
27/27 [==============================] - 1s 26ms/step - loss: 2.9486 - accuracy: 0.2218
```

Experiment Evaluation for modified dataset

VGG16:

```
[170] vgg_model_original = vgg_model.evaluate(val_ds)

27/27 [==============================] - 1s 29ms/step - loss: 0.6042 - accuracy: 0.8165

vgg_model_graf = vgg_model.evaluate(m_val_ds)

27/27 [==============================] - 1s 28ms/step - loss: 1.9862 - accuracy: 0.5396
```

InceptionV3:

```
[172] inception_model_original = inception_model.evaluate(val_ds)

27/27 [==============================] - 1s 24ms/step - loss: 0.3440 - accuracy: 0.8633

inception_model_graf = inception_model.evaluate(m_val_ds)

27/27 [==============================] - 1s 24ms/step - loss: 1.5873 - accuracy: 0.6187
```

Experiment Evaluation for modified dataset

Model	Val Acc (original dataset)	Val Acc (modified dataset)
LeNet-5	0.6847	0.5156
ResNet-50	0.2494	0.2218
VGG-16	0.8165	0.5396
Inception-V3	0.8633	0.6187

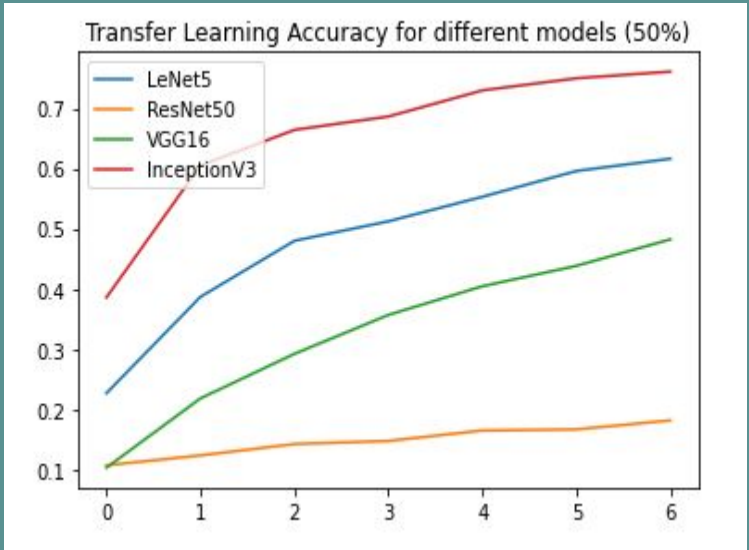
We can see that InceptionV3 is still the best-performing model

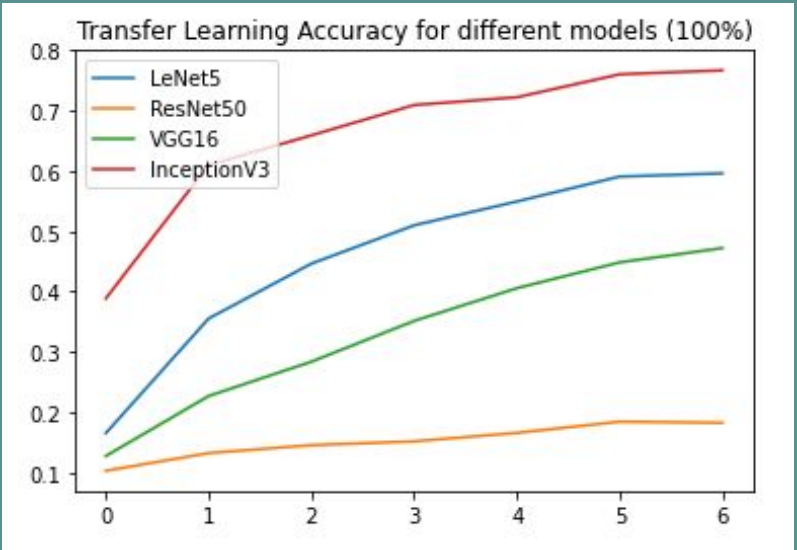
Transfer Learning Implementation

- Transfer learning is a machine learning technique for adapting pre-trained models to new tasks and can significantly improve the performance of the model on the new task with minimal additional training.
- Implementation on our code:

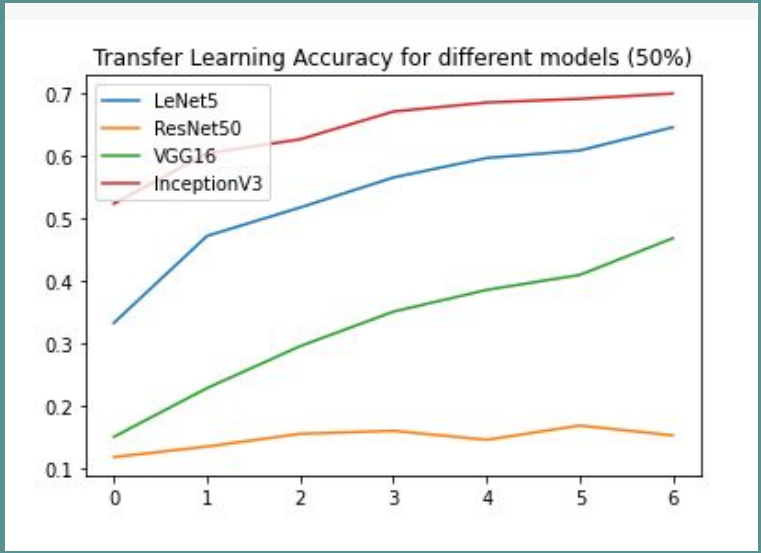
1. Create a new model and add the pretrained model layers up to the last convolutional block.
2. Freeze the first half of the layers / all of the layers.
3. Add a new dense layer with 58 units and a softmax activation function.
4. Compile the model with a loss function of Sparse Categorical Cross entropy and an Adam optimizer.
5. Set callback functions to early stop training.
6. Train the model using the train_modified and val_modified datasets.

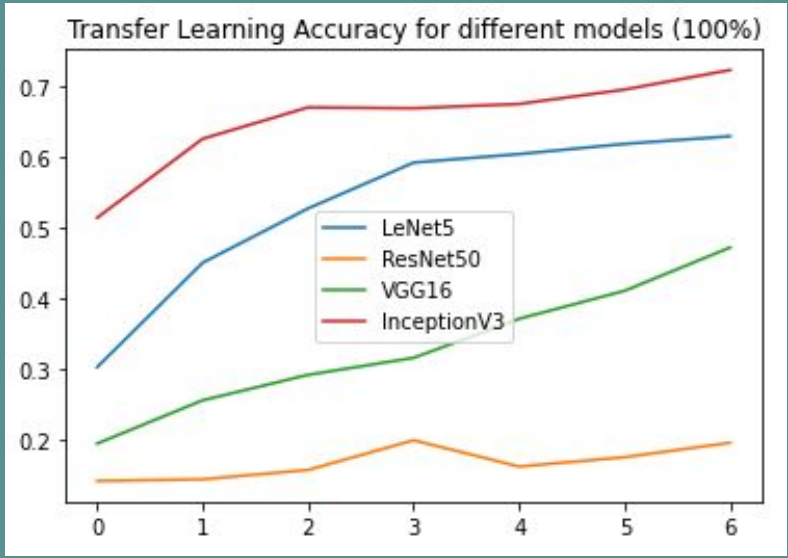
Transfer Learning Train Accuracy





Transfer Learning Validation Accuracy





Experiment Evaluation for modified dataset

Model	Before transfer learning on modified dataset	After transfer learning on modified dataset (freeze 50%)	After transfer learning on modified dataset (freeze 100%)
	Val Acc (modified dataset)	Val Acc (modified dataset)	Val Acc (modified dataset)
LeNet-5	0.5156	0.6463	0.6283
ResNet-50	0.2218	0.1535	0.1954
VGG-16	0.5396	0.4688	0.4712
Inception-V3	0.6187	0.7002	0.7218

We can see that InceptionV3 is still the best-performing model

Conclusion

1. **Accuracy:**

   InceptionV3 > VGG16 > LeNet5 > ResNet50

2. **Robustness Against Graffitied Traffic Signs:**

   InceptionV3 > VGG16 > LeNet5 > ResNet50

3. **Change of Validation Accuracy between original data and modified data:**

Model	LeNet-5	ResNet-50	VGG-16	Inception-V3
Change of Val Acc	-0.1691	-0.0276	-0.2769	-0.2446

4. **Transferability**:

   · LeNet5 and Inception-V3: ↑
   · ResNet50 and VGG-16: ↓

GitHub Repo

https://github.com/JefferеyChasing/Traffic-Sign-Classification