GTSRB - German Traffic Sign Recognition

ZHU Zheyu 20076219d

Introduction

a. Background and motivation

In recent years, the advancement of autonomous vehicles and intelligent transportation systems has gained significant attention. One of the critical components in ensuring safe and efficient navigation is the ability to accurately recognize traffic signs. Traffic sign recognition plays a vital role in assisting both human drivers and autonomous vehicles in adhering to traffic rules and making informed decisions. Machine learning and deep learning techniques have shown promising results in image recognition tasks, and their application to traffic sign recognition is a natural extension.

b. Problem statement

The goal of this project is to develop and compare various machine learning and deep learning models for traffic sign recognition. We aim to accurately classify traffic signs from the GTSRB (German Traffic Sign Recognition Benchmark) dataset using different techniques, including logistic regression, support vector machines, multi-layer perceptrons, PCA[1], convolutional neural networks[2], and ResNet[3]. The performance of these models will be evaluated and analyzed to determine the most effective approach for traffic sign recognition.

c. Overview of the methodologies

In this project, we will experiment with six different methodologies for traffic sign recognition. We will start with traditional machine learning techniques, such as logistic regression, support vector machines, and multi-layer perceptrons, which have shown success in various classification tasks. To reduce the dimensionality of the input data, we will employ PCA (Principal Component Analysis) [1]and analyze its impact on model performance. Following that, we will explore deep learning techniques, specifically convolutional neural networks (CNN)[2] and ResNet[3], which have demonstrated state-of-the-art performance in various image recognition tasks. By comparing the performance of these models, we aim to identify the most effective approach for traffic sign recognition.

Dataset Description and Preprocessin

a. GTSRB - German Traffic Sign Recognition Benchmark

i. Source and collection process

The GTSRB dataset was created as part of the German Traffic Sign Recognition Benchmark competition hosted by the Institute for Neural Computation at Ruhr-Universität Bochum. The dataset comprises over 50,000 images of 43 different traffic signs, collected from various sources in real-world driving conditions. Images were captured using a camera mounted on a vehicle and then manually annotated with a bounding box enclosing the traffic sign and its corresponding class label.

ii. Structure and content

The GTSRB dataset contains 43 classes of traffic signs, including speed limits, prohibitory signs, mandatory signs, and other informative signs. Each class has a varying number of images, ranging from 210 to 2250, with a total of 51,630 images. The dataset is divided into a training set, which consists of 39,209 images, and a test set, containing 12,630 images. Each image is a 3-channel color image (RGB) and comes with varying resolutions.

Consists of 4 files:

1. Train folder
2. Test folder
3. Train.csv
4. Test.csv

The test folder and train folder contains

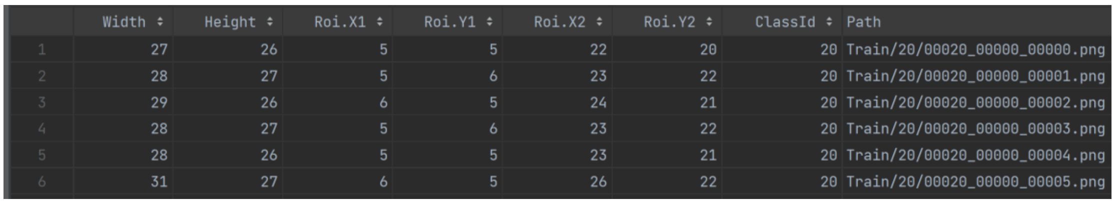
43 sub-folders

each folder is one class of traffic signs

The name of the folder is the classId in csv

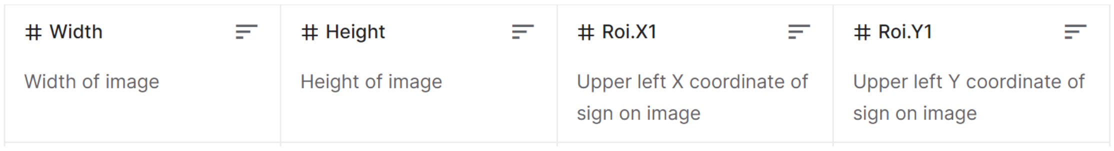
Each class folder contents all images of that class traffic sign

Both 2 csv files have 8 columns

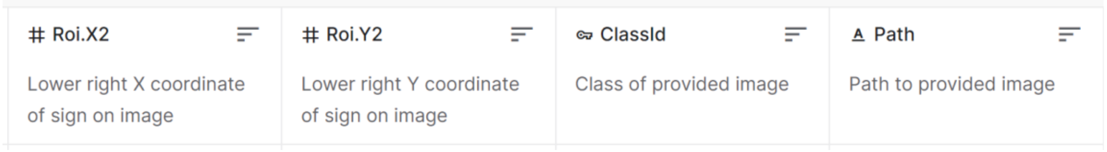


	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
1	27	26	5	5	22	20	20	Train/20/00020_00000_00000.png
2	28	27	5	6	23	22	20	Train/20/00020_00000_00001.png
3	29	26	6	5	24	21	20	Train/20/00020_00000_00002.png
4	28	27	5	6	23	22	20	Train/20/00020_00000_00003.png
5	28	26	5	5	23	21	20	Train/20/00020_00000_00004.png
6	31	27	6	5	26	22	20	Train/20/00020_00000_00005.png

The meaning of column is followed:



# Width	# Height	# Roi.X1	# Roi.Y1
Width of image	Height of image	Upper left X coordinate of sign on image	Upper left Y coordinate of sign on image

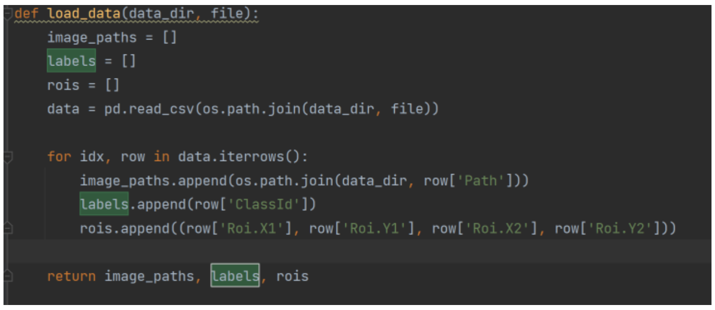


# Roi.X2	# Roi.Y2	ClassId	Path
Lower right X coordinate of sign on image	Lower right Y coordinate of sign on image	Class of provided image	Path to provided image

b. Data preprocessing

i.Data reading

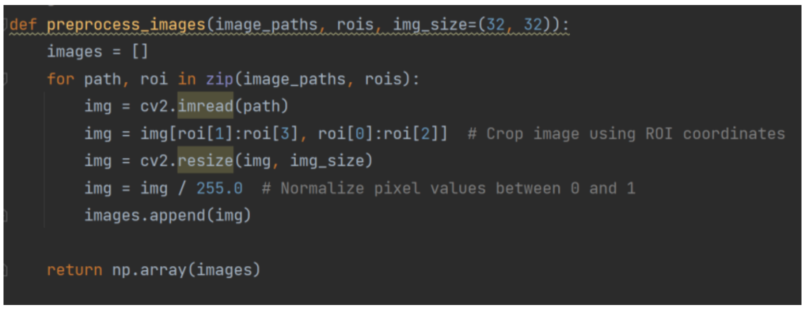
use pandas.read_csv() to read the csv as panda array



```
def load_data(data_dir, file):
    image_paths = []
    labels = []
    rois = []
    data = pd.read_csv(os.path.join(data_dir, file))

    for idx, row in data.iterrows():
        image_paths.append(os.path.join(data_dir, row['Path']))
        labels.append(row['ClassId'])
        rois.append((row['Roi.X1'], row['Roi.Y1'], row['Roi.X2'], row['Roi.Y2']))

    return image_paths, labels, rois
```

Use cv2.imread to convert png to numpy arrary

image = cv2.imread("path/to/image.png")



```
def preprocess_images(image_paths, rois, img_size=(32, 32)):
    images = []
    for path, roi in zip(image_paths, rois):
        img = cv2.imread(path)
        img = img[roi[1]:roi[3], roi[0]:roi[2]]  # Crop image using ROI coordinates
        img = cv2.resize(img, img_size)
        img = img / 255.0  # Normalize pixel values between 0 and 1
        images.append(img)

    return np.array(images)
```

ii. Image resizing and normalization

use the coordinates of the edge of the sign provided by the csv to crop the images.

To ensure consistency and reduce computational complexity, all images will be resized to a fixed dimension, we using, 32x32 pixels.

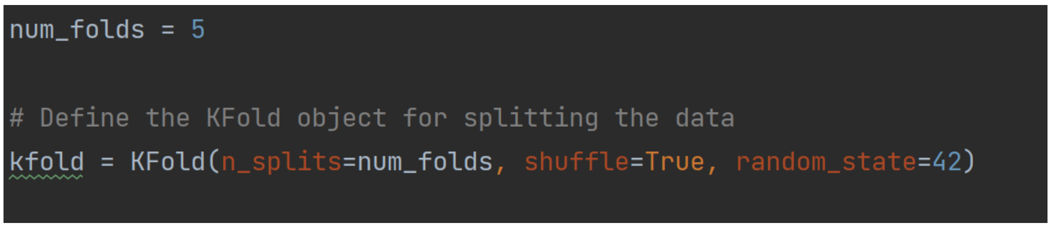
Additionally, images will be normalized to have pixel values ranging from 0 to 1, instead of the original 0 to 255. This normalization process helps improve the convergence and generalization of the models.

iii. Train-test split, training -validation split.

The GTSRB dataset is already divided into a training set and a test set.

However, to evaluate the models' performance during the training process and avoid overfitting, we will further split the training set into a training set and a validation set.

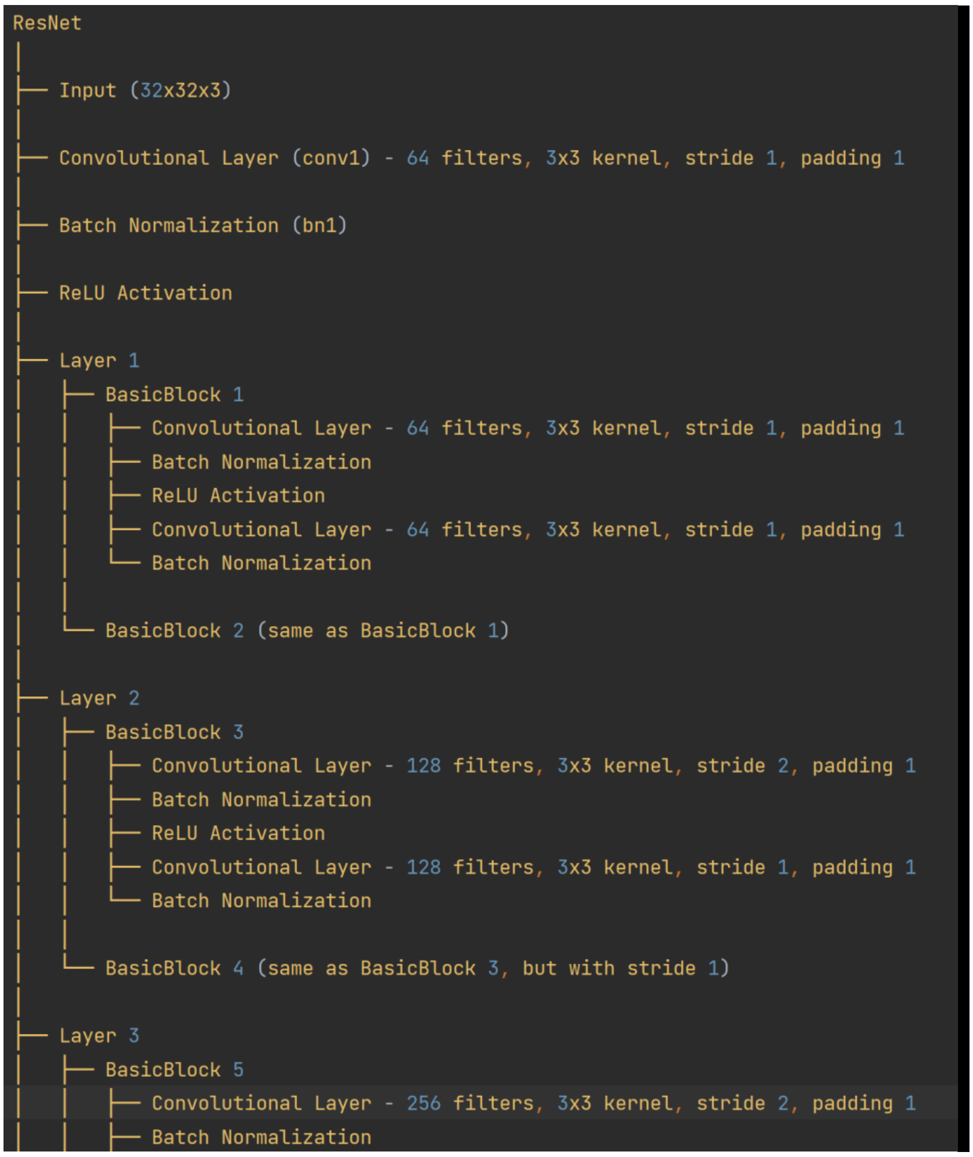
We use k-fold,with k=5, shuffle=True, random_state=42,which is to allocate 80% of the training set for training and the remaining 20% for validation and shuffle the data before splitting into batches.



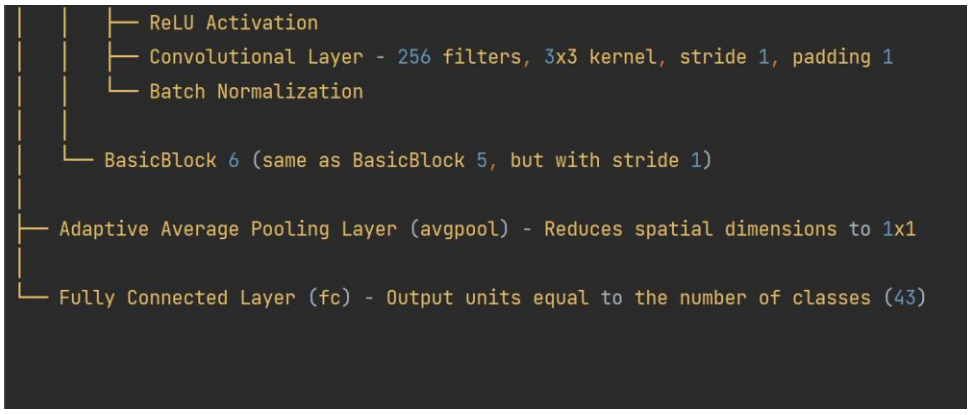
```
num_folds = 5

# Define the KFold object for splitting the data
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)
```

Methodologies

. ResNet



```
ResNet
│
├── Input (32x32x3)
│
├── Convolutional Layer (conv1) - 64 filters, 3x3 kernel, stride 1, padding 1
│
├── Batch Normalization (bn1)
│
├── ReLU Activation
│
├── Layer 1
│   ├── BasicBlock 1
│   │   ├── Convolutional Layer - 64 filters, 3x3 kernel, stride 1, padding 1
│   │   ├── Batch Normalization
│   │   ├── ReLU Activation
│   │   ├── Convolutional Layer - 64 filters, 3x3 kernel, stride 1, padding 1
│   │   └── Batch Normalization
│   │
│   └── BasicBlock 2 (same as BasicBlock 1)
│
├── Layer 2
│   ├── BasicBlock 3
│   │   ├── Convolutional Layer - 128 filters, 3x3 kernel, stride 2, padding 1
│   │   ├── Batch Normalization
│   │   ├── ReLU Activation
│   │   ├── Convolutional Layer - 128 filters, 3x3 kernel, stride 1, padding 1
│   │   └── Batch Normalization
│   │
│   └── BasicBlock 4 (same as BasicBlock 3, but with stride 1)
│
├── Layer 3
│   ├── BasicBlock 5
│   │   ├── Convolutional Layer - 256 filters, 3x3 kernel, stride 2, padding 1
│   │   ├── Batch Normalization
```



```
│   │   ├── ReLU Activation
│   │   ├── Convolutional Layer - 256 filters, 3x3 kernel, stride 1, padding 1
│   │   └── Batch Normalization
│   │
│   └── BasicBlock 6 (same as BasicBlock 5, but with stride 1)
│
├── Adaptive Average Pooling Layer (avgpool) - Reduces spatial dimensions to 1x1
│
└── Fully Connected Layer (fc) - Output units equal to the number of classes (43)
```

i. Model architecture and parameters

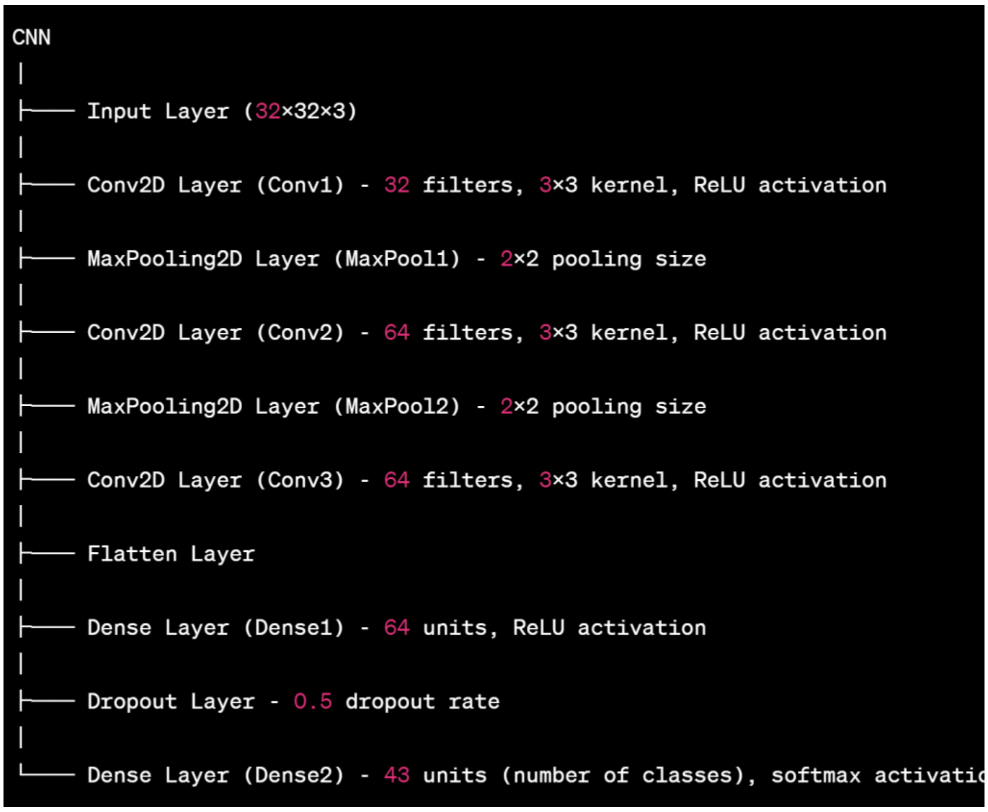
The ResNet architecture used in this project is a modified version of the ResNet-18 model[3], specifically designed for 32x32 images. The model consists of an initial convolutional layer with 64 filters and a kernel size of 3x3, followed by batch normalization and a ReLU activation. Subsequently, three layers of residual blocks are applied, each with different numbers of filters (64, 128, and 256) and strides (1, 2, and 2), respectively. After the final residual block, an adaptive average pooling layer is used to reduce the spatial dimensions to 1x1, and a fully connected layer is added for classification, with the number of output units equal to the number of classes (43).

ii. Hyperparameter tuning and selection

In this project, the Adam optimizer with a learning rate of 0.001 is used for training the model. The batch size for the DataLoader is set to 64, and the training process lasts for 20 epochs. The choice of these hyperparameters is based on their common use in similar deep learning tasks and the observed performance on the given dataset. However, further tuning of these hyperparameters could potentially lead to improved performance.

Convolutional Neural Network (CNN)

i. Model architecture and parameters:



```
CNN
|
├── Input Layer (32×32×3)
|
├── Conv2D Layer (Conv1) - 32 filters, 3×3 kernel, ReLU activation
|
├── MaxPooling2D Layer (MaxPool1) - 2×2 pooling size
|
├── Conv2D Layer (Conv2) - 64 filters, 3×3 kernel, ReLU activation
|
├── MaxPooling2D Layer (MaxPool2) - 2×2 pooling size
|
├── Conv2D Layer (Conv3) - 64 filters, 3×3 kernel, ReLU activation
|
├── Flatten Layer
|
├── Dense Layer (Dense1) - 64 units, ReLU activation
|
├── Dropout Layer - 0.5 dropout rate
|
└── Dense Layer (Dense2) - 43 units (number of classes), softmax activati
```

Input layer: Takes a 32x32x3 image as input.
Conv2D layer (Conv1): 32 filters, kernel size 3x3, activation function ReLU.
MaxPooling2D layer (MaxPool1): Pooling size 2x2.
Conv2D layer (Conv2): 64 filters, kernel size 3x3, activation function ReLU.
MaxPooling2D layer (MaxPool2): Pooling size 2x2.
Conv2D layer (Conv3): 64 filters, kernel size 3x3, activation function ReLU.
Flatten layer: Flattens the input to a 1D array.
Dense layer (Dense1): 64 units, activation function ReLU.
Dropout layer: Dropout rate 0.5.
Dense layer (Dense2): Number of units equal to the number of classes (43), activation function softmax.
ii. Hyperparameter tuning and selection:

Optimizer: Adam optimizer is used with default learning rate (0.001) and other parameters.
Loss function: Categorical crossentropy is used as the loss function, suitable for multi-class classification.
Batch size: 32 samples per batch.
Number of epochs: 10 epochs for training.
Dropout rate: 0.5, used to prevent overfitting by randomly dropping out a fraction of the input units during training.

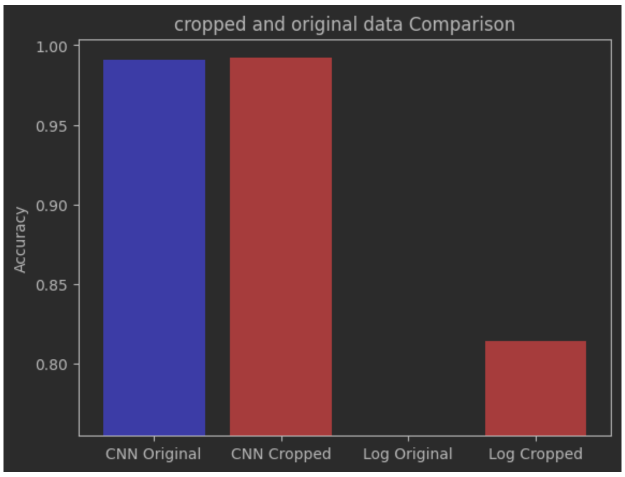
Results and Analysis

In this study, various models were evaluated using different preprocessing techniques, including cropped images and PCA[1] 100 dimensional data.

Cropped vs. Uncropped Data:

Use the coordinates provided by csv to crop the image to avoid background noise.
The performance of the models in train-validation set was compared using cropped and uncropped images. The results are as follows:

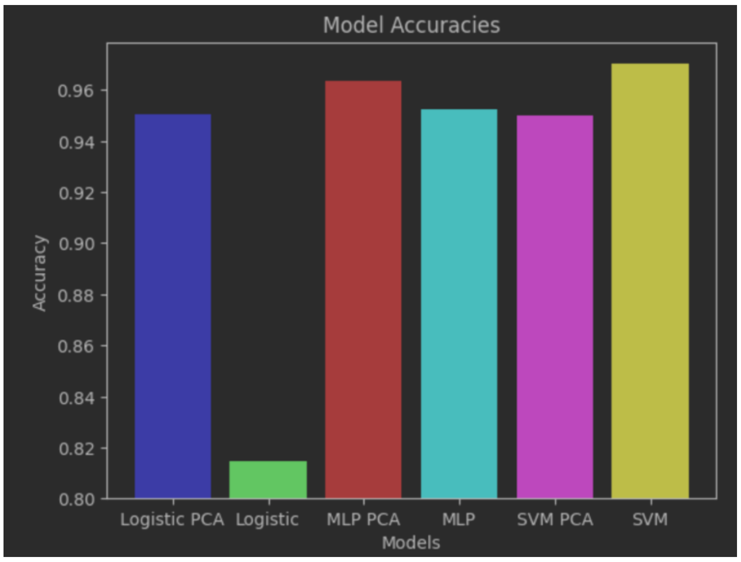
CNN original accuracy (uncropped): 0.9907
CNN cropped accuracy: 0.9920
Logistic Regression original accuracy (uncropped): 0.7549
Logistic Regression cropped accuracy: 0.8146



These results show that using cropped images improves the performance of both the CNN and logistic regression models. Based on this observation, cropped images were used for all subsequent tests.
Both models are improved but CNN improve slightly .it is possible that CNN can extract features from images, ignoring noise of backgrounds.

PCA 100 Dimensional Data vs. Original Data:

The performance of the models was compared using PCA 100 dimensional data and original data in trian-validation set. The results are as follows:



Logistic Regression PCA 100 accuracy: 0.9504
Logistic Regression original accuracy: 0.8146
MLP PCA 100 accuracy: 0.9636
MLP original accuracy: 0.9524
SVM PCA 100 accuracy: 0.9500
SVM original accuracy: 0.9700

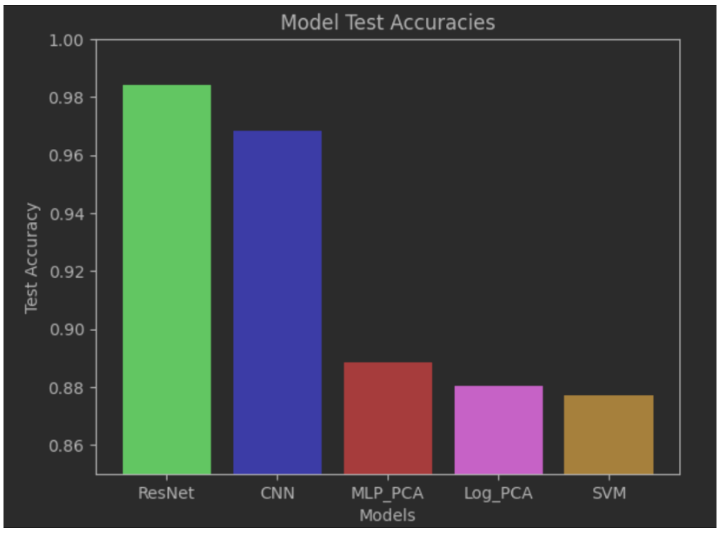
Both MLP and Logistic Regression are improved by pca 100,but SVM decreased.

My assumption:
In high dimensional MLP and Logistic Regression will be overfitting, and pca may avoid overfit to improve the accuracy.
The model with have better performance in high dimensional like SVM ,ResNet,CNN will be worse in pca because in the dimension reducing, the information of data will loss.

So I will choose use cropped data for all the model, MLP and Logistic Regression using the pca 100,CNN and Resnet using original cropped data.

Test

After the validation process, the models were evaluated on the test dataset. The test results are as follows:



CNN Test Accuracy: 0.9684085510688836

ResNet Accuracy: 0.9844

MLP (PCA 100) Test Accuracy: 0.8883610451306413

Logistic Regression (PCA 100) Test Accuracy: 0.8802058590657166

SVM Test Accuracy: 0.876959619952494

When comparing the test results, the ResNet model achieved the highest accuracy (0.9844), followed by the CNN model (0.9684). The MLP and logistic regression models, both using PCA 100 dimensional data, had lower accuracies of 0.8884 and 0.8802, respectively. The SVM model, which did not use PCA 100 dimensional data, had an accuracy of 0.8769.

Discussion and Insights

a. Interpretation of results

The results of this study indicate that different models perform better under different preprocessing conditions. The use of cropped images may improve the performance of all models, while PCA 100 dimensional data improved the performance of model which are not good at high dimensional data like logistic regression and MLP models, but will downgrade in the others models like the SVM model because in the dimensional reducing the information of data will loss. The deep learning architectures like ResNet and CNN models can achieve better performance when using on images .

b. Potential improvements and recommendations

To further improve the performance of the models, additional preprocessing techniques, such as data augmentation, could be explored. Fine-tuning the hyperparameters of each model could lead to better performance as well. Moreover, more advanced architectures like EfficientNet or other state-of-the-art models could be employed.

Conclusion

a. Summary of findings

In this study, various models were evaluated using different preprocessing techniques, including cropped images and PCA 100 dimensional data. The deep learning architectures like ResNet and CNN models demonstrated the better performance on this dataset when using cropped images. The use of PCA with 100-dimensional data provided improvements for the logistic regression and MLP models but was not beneficial for the other model.

b. Final thoughts

Based on these results, it is recommended to use the ResNet or CNN models with cropped images for this task. The SVM model, while not as accurate as the ResNet or CNN models, still achieved a reasonably high accuracy and could be considered for use in cases where computational resources are limited. Further exploration of different models, preprocessing techniques, and fine-tuning strategies could lead to improved performance in future work.

References:
1.Jolliffe, I. T. (2002). Principal Component Analysis. Springer Series in Statistics. Springer New York. doi:10.1007/b98835
2. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems 25 (NIPS 2012), 1097-1105
3.He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778. doi:10.1109/CVPR.2016.90