ECE 552

Fuzzy Systems

Student Names:

- **Muhammad Mustafa**
- **Mohammed Faraz Hussain**

Abstract:

Michigan has a well-established, large infrastructure of roads and pavements that caters to about 8 million vehicles. Unfortunately, one of the most common obstacles faced on Michigan's roads today is the presence of potholes. In Michigan, many potholes are formed due to the extreme freeze-thaw cycles and old infrastructure, which speeds up the wear and tear on the pavement. Potholes have been an ongoing problem, and they only increase more day by day. Most recently, dating to 2024, a large pothole on I-96 near Lansing caused a lot of car accidents early because a lot of drivers had unexpected tire blowouts. This and many other instances demonstrate how unmonitored road flaws can lead to hazardous traffic incidents. One possible solution would be a detection system for potholes that'd not only be reliable but also automated because of the number of potholes in this state. This research builds on that idea and investigates a fuzzy-logic-based pothole classifier built on engineered features and compares its performance with two deep learning transfer learning models: VGG-16 and ResNet-50. In our fuzzy system, the approach processes sliding window image patches to estimate pothole severity based on intensity, local texture variance, edge density, and blob shape image descriptors. On the other hand, CNN models directly learn these discriminative features from the labeled images. Experimental evaluation shows that, while a fuzzy system excels in interpretability and computational efficiency, its accuracy is limited to approximately 85% - 88% under optimal conditions, which decreases significantly if potholes occur at extreme views or outside of the image center. In this paper, we compare our fuzzy system to VGG-16, a neural network, and from evaluation, we've observed that it reaches 96%, and ResNet-50, with 94%, performing much better compared to the generalization and robustness of the fuzzy system over diverse conditions. This work concludes that deep learning provides better performance for large-scale deployment, while

fuzzy logic is useful in lightweight, interpretable, and edge-computing environments. Hybrid models are recommended, along with future improvements.

I. Introduction:

For years, Michigan has struggled with deteriorating road surfaces because of the extreme winters with heavy vehicle loads and inefficient long-term pavement funding. In fact, because of the freeze/thaw process, the tar expands, thus shrinking, which results in cracks that develop into potholes. The annual reports filed by the Michigan Department of Transportation include several thousand reports of damage, with costs typically in the tens of millions of dollars. Early in 2024, there was a massive pothole on I-96, which caused a multi-car accident because several cars experienced tire blowouts. Such incidents are proof that undetected potholes cannot afford to be prevented by anything less than enhanced safety features than mere manual inspection.

Several technology-based solutions have evolved to advance roadway safety, most of which involve the automation of pothole detection. Most cars, mobile phones, and city vehicle inspection cars are equipped with high-resolution cameras, which make image-based detection a real possibility. The existing vision-based solutions use thresholding, edge detection, or geometric features to identify the condition of the road but find it difficult to generalize to the shapes, shadows, and textures involved.

These are mostly represented by two major paradigms: fuzzy logic systems and deep CNNs. Fuzzy systems encode domain knowledge through linguistic rules and are thus highly interpretable and very computationally efficient, which makes them appeal for embedded systems, such as dashcams or mobile apps. Unfortunately, fuzzy models may fail under complex imaging conditions. Deep learning models, on the other hand, automatically learn robust

hierarchical features from data given and have proven highly effective in a range of surface-defect detection problems such as pavement distress, road anomalies, and structural deterioration [1, 2, 3, 4]. Transfer learning using architectures like VGG-16 and ResNet-50 allows achieving high accuracy even on moderately sized training datasets. This research compares the effectiveness of a fuzzy-logic-based pothole detection system to two deep learning CNN architectures. The dataset and experimentation used in both approaches are real-world road conditions from Michigan. The objective is to find not just which method is better but also the strengths, limitations, and realistic deployment cases for each method, hence the real-world dataset.

II. Background:

Pavement defect detection has been an active area of research over the last two decades. Earlier systems have relied on traditional image-processing operations, such as Otsu thresholding, texture filters, Gabor descriptors, and edge-based segmentation. Koch and Brilakis, for example, proposed an edgebased method to detect concrete pavement cracks by analyzing pixel discontinuities [1]. This faces an immediate disadvantage if shadows or noisy asphalt textures are present.

Fuzzy logic has also played an important role in pavement condition assessment. Several studies have used fuzzy thresholding, fuzzy clustering, and fuzzy rule-based decision systems to identify pavement anomalies from normal texture patterns. Ahmed and Lahouar [2] showed that the fuzzy segmentation technique was able to isolate the boundaries of a pothole for moderate lighting variations; however, its performance degraded drastically on roads with heavy texture. These studies reflect the interpretability advantages shown by fuzzy logic but also expose sensitivity to feature selection and image variance.

Deep learning has revolutionized surface defect detection because it allows for direct learning of high-level representation from raw images. Keeping that mindset, Zhang et al. [3] proposed deep CNNs for road crack recognition. Results showed a significant improvement in performance compared to classical methods. Fan and Zhao [4] approached pavement distress detection using transfer learning with VGG-16, showing robustness to shadows, variable asphalt textures, and variations in camera angle. ResNet-based models were also shown to generalize well across various pavements and environmental conditions [5].

While previous work confirms the accuracy superiority of deep learning, fuzzy systems are still valued for real-time inference, low-power deployment, and interpretable decision-making. To the best of our knowledge, limited research has directly compared fuzzy logic versus CNN-based pothole detection pipelines on Michigan-style roadway imagery. This study contributes by systematically comparing both methods with consistent datasets and evaluation protocols.

III. Implementation:

The implementation of the proposed pothole detection framework consists of two major components: (1) a rule-based fuzzy logic system developed in MATLAB, and (2) two deep learning classifiers based on the VGG-16 and ResNet-50 architectures implemented in Python using PyTorch. Both pipelines are applied to the same dataset of Michigan roadway images to perform a controlled comparison of the interpretability, robustness, and accuracy. In this section, both subsystems are described in full implementation, including feature extraction, fuzzy inference, model training, evaluation, and visualization. Code excerpts are given to show the core algorithm used in the system.

Fuzzy Logic Based Pothole Detection System

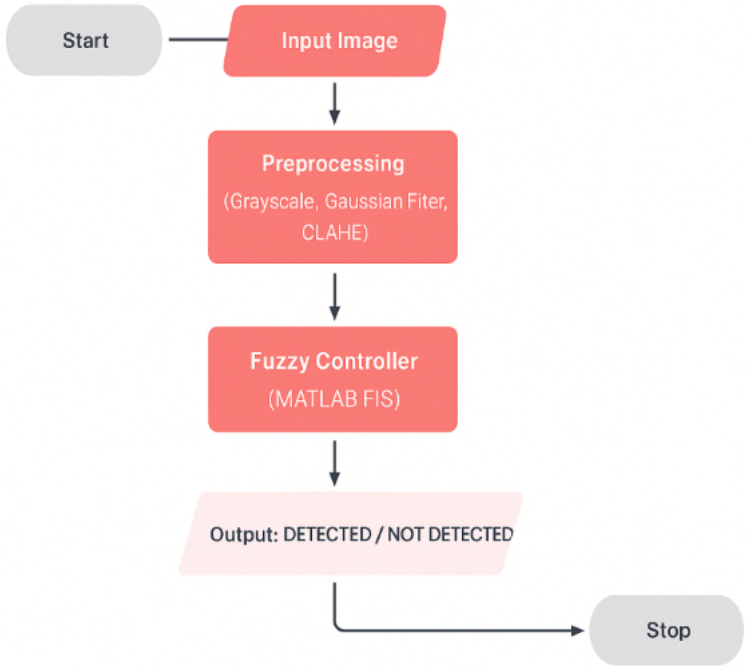


Figure 1 Fuzzy Logic Pothole Detection Flowchart

A. Image Preprocessing and Enhancement:

The implemented pothole detection system is based on fuzzy logic and is developed within the environment of MATLAB using a Mamdani-type FIS. The general goal of the system is to detect and localize pothole regions of a road image by performing reasoning over several low-level visual features extracted from localized image patches. The entire pipeline holds end to end processing, including image preprocessing, sliding-window feature extraction, fuzzy inference and defuzzification, confidence heatmap generation, and final image-level decision making.

The implementation starts by first loading a road image and then converting it to grayscale. This is followed by the applying Gaussian smoothing, which reduces noise and enhances local contrast, and Contrast Limited Adaptive Histogram Equalization. This step improves the

performance in visualizing surface irregularities like potholes under non-uniform lighting conditions. Grayscale images now serve as input to the subsequent sliding window analysis.

B. Sliding Window Feature Extraction:

For localized reasoning, the preprocessed image is divided into overlapping square patches by a sliding-window approach. The size of the window is selected dynamically with respect to image dimensions. A fixed selection in the sliding-window approach ensures enough overlap between the patches that are nearby. Because of this unique way of implementation, it can capture potholes at diverse spatial locations and scales. Four discriminative features are computed for each image patch: mean intensity, standard deviation of intensity, edge density, and shape aspect ratio. The mean intensity captures local changes in brightness, which are often related to a depression in the pothole area. Standard deviation quantifies the roughness of texture. Edge density is determined by the Canny edge detector, and it denotes the presence of sharp structural boundaries around the rim of the pothole. Shape information is obtained by adaptive thresholding followed by the analysis of connected components; the ratio between the length of the minor and major axes is considered a proxy of circularity.

The following code snippet gives a demonstration of the extraction of intensity, texture, and edge-based features.

```
I_vals(idx)   = mean(patch(:));
sig_vals(idx) = std(double(patch(:)));

edges         = edge(patch,'canny');
E_vals(idx)   = sum(edges(:))/numel(edges);
```

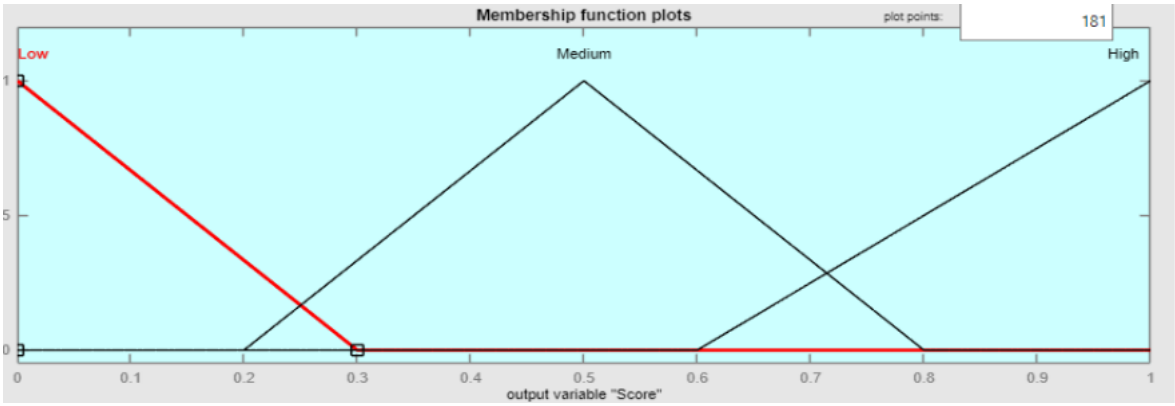
Similarly, the shape-based feature computation, which supports geometric reasoning in the fuzzy system, is shown below

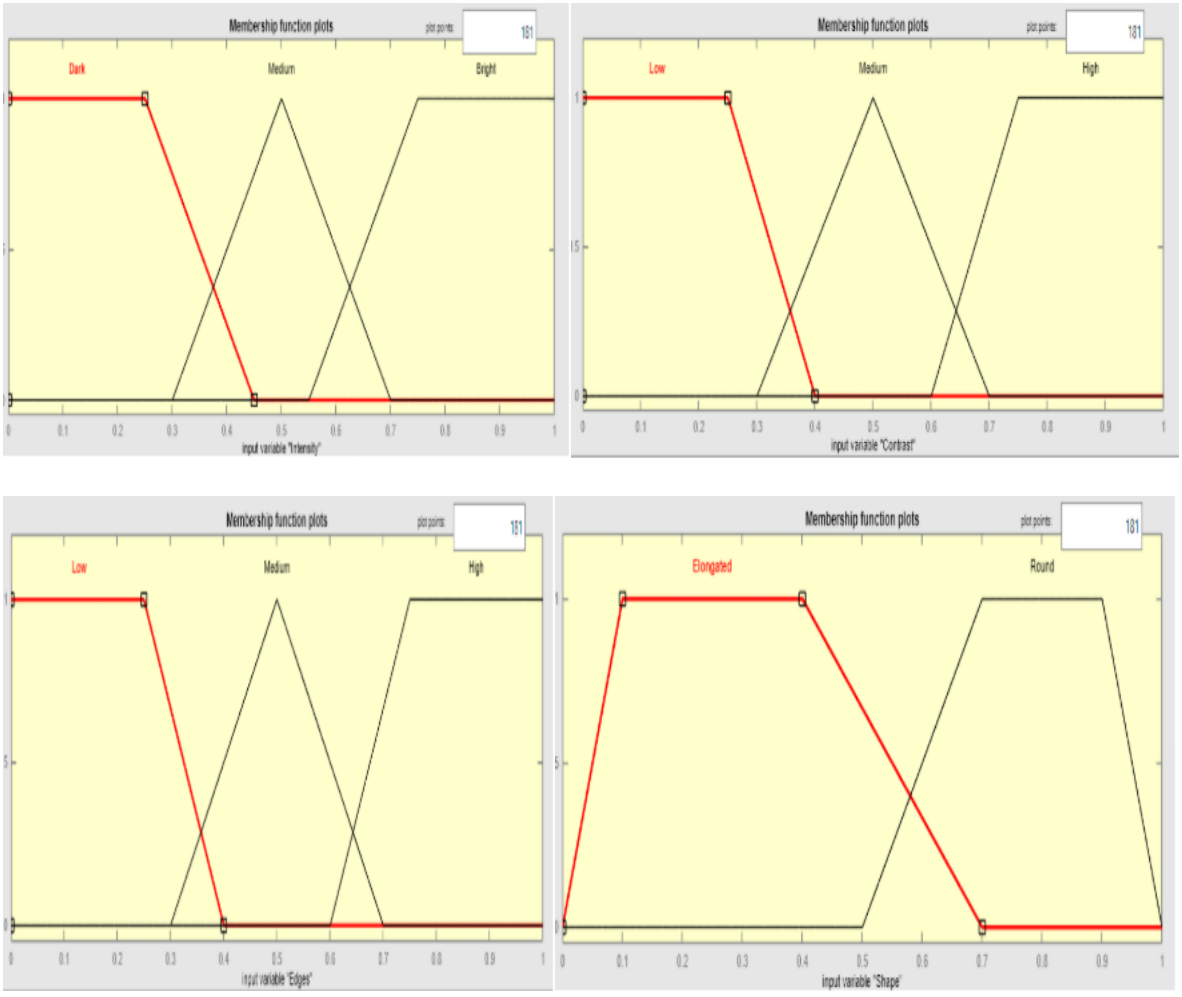
```
stats = regionprops(BW,'Area','MajorAxisLength','MinorAxisLength');

if ~isempty(stats)
    [~,bigIdx] = max([stats.Area]);
    if stats(bigIdx).Area > 0.02*numel(patch)
        A_vals(idx) = stats(bigIdx).MinorAxisLength / stats(bigIdx).MajorAxisLength;
    else
        A_vals(idx) = 1.0;
```

C. Future Normalization and FIS Input Mapping:

All extracted features are scaled to the range [0,1] before inference because the fuzzy inference system uses normalized inputs. This normalization ensures that the same membership in a fuzzy set is activated, independently of absolute image intensities or resolutions. Normalized features directly correspond to the four FIS input variables: Intensity, StdDev, EdgeDensity, and AspectRatio.

Each input variable is represented using a triangular membership function, as explained with the help of membership function plots in the earlier section. Intensity, standard deviation, and edge density are represented using Low, Medium, and High linguistic terms, while the membership functions for aspect ratio are represented as circular and elongated. These membership functions will enable the fuzzy system to draw conclusions about pothole characteristics in a humanistic interpretable way. The figures showing these membership functions are given here.



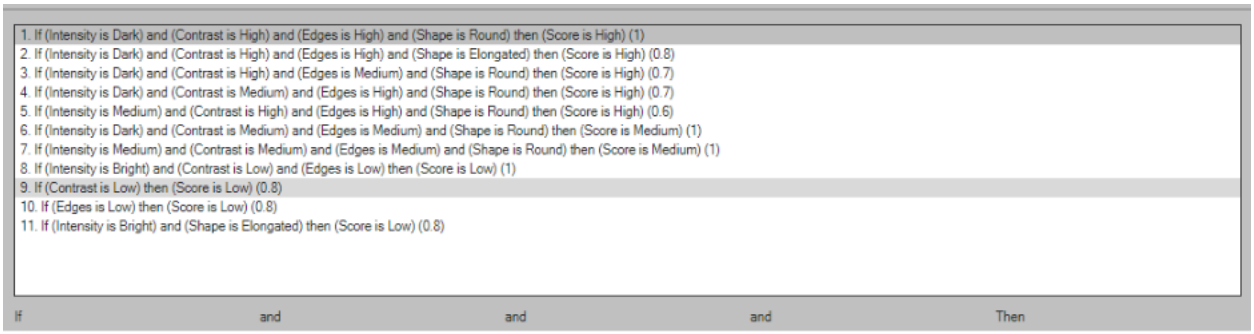


D. Fuzzy Inference and Rule Evaluation:

Mamdani FIS is directly loaded from the final .fis file, with the rule base and membership definitions being consistent with the experimental set-up. The fuzzy inference system assesses each image patch individually by translating the normalized vector of attributes into a pothole likelihood value. This is carried out by rule firing, max-aggregation, and centroid defuzzification. This is how fuzzy inference is carried out on all the sliding window patches, as shown in the following piece of code:

```
fis = readfis('PotholeFIS_final.fis');

scores = zeros(length(I_n),1);
for i = 1:length(I_n)
    scores(i) = evalfis(fis,[I_n(i) S_n(i) E_n(i) A_n(i)]);
end
```

The rule base combines texture, edge, and shape cues to detect pothole likelihood. For instance, patches having high edge density and high texture variation with circular shape characteristics point towards a high chance of being a pothole, while smooth or elongated regions are on the lower end of the spectrum. The complete rule base is best visualized in the rule visualization figure and is referred to here.



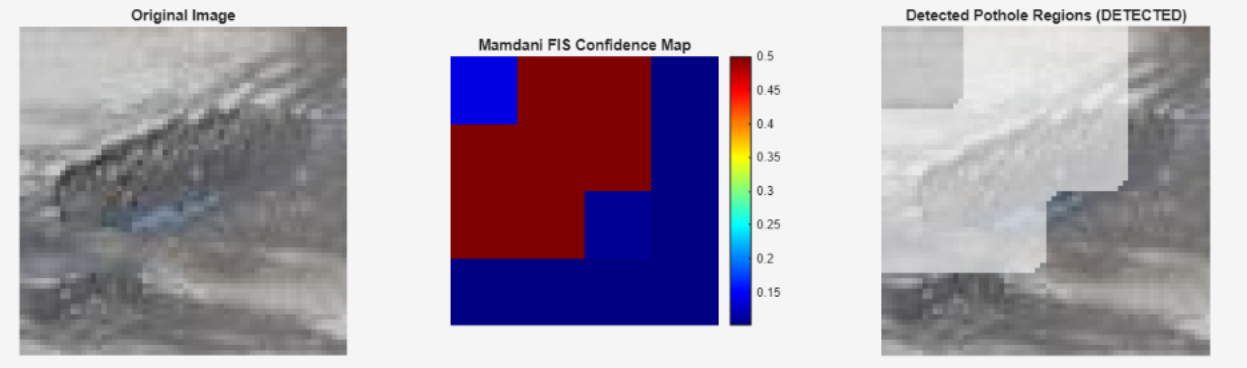
E. Confidence Heatmap Generation and Localization:

These patch-level fuzzy confidence scores are projected back onto the image plane to generate a dense confidence heatmap. Each sliding-window patch will contribute its fuzzy score for the corresponding spatial region, and the combination of overlapping regions is performed through a maximum operation. This mechanism preserves strong pothole responses while suppressing noise from non-relevant regions. The following snippet shows the accumulation process of the heatmap and would be included, linking

the fuzzy outputs with the spatial localization:

```
heatmap = zeros(h,w);
for i = 1:length(scores)
    cy = round(centers(i,1));
    cx = round(centers(i,2));

    y1 = max(1, cy - win/2);
    y2 = min(h, cy + win/2 - 1);
    x1 = max(1, cx - win/2);
    x2 = min(w, cx + win/2 - 1);

    heatmap(y1:y2,x1:x2) = max(heatmap(y1:y2,x1:x2), scores(i));
end
```

It results in a heatmap that shows visually the likelihood of potholes across the image, as shown in the heatmap figures below:



F. Image Level Decision and Visualization

For making a final detection, the global statistical properties of the heatmap come into play. The presence of a pothole is considered detected if the maximum value of the confidence is greater than a certain threshold value, and a large enough contrast is obtained between the maximum and average values of the confidence.

The identified regions are visualized by thresholding the heatmap, where a binary mask is used to overlay on the original image. The result provides a final output that includes the original image, fuzzy confidence map, and the identified pothole regions marked in the image.

Function Name	Type	Description
pothole_fuzzy_heatmap()	Main controller	A function that carries out the entire fuzzy logic pipeline, including the creation of a confidence map, sliding-window feature extraction, image preprocessing, fuzzy inference evaluation, and pothole region visualization.
readfis()	FIS loader	loads the Mamdani fuzzy inference system, which is defined in the external.fis file and includes the pothole reasoning rule base and membership functions.
evalfis()	FIS evaluator	generates a continuous pothole confidence score by evaluating the fuzzy inference system for every image patch using normalized feature inputs.

G. Implementation of CNN-Based Pothole Detection Models

Two CNN architectures are implemented: VGG-16 and ResNet-50; due to their versatility for visual feature extraction and usage in image classification applications. Both models used transfer learning to adapt the pre-trained weights learned on the ImageNet dataset. This significantly reduces the training time while improving the generalization performance on the relatively limited dataset.

This is a structured call sequence from loading the dataset, pre-processing, developing models,

training, evaluation, and prediction display. The high-level call graph of this is illustrated in figure 2, which shows how the master script coordinates all the other components.

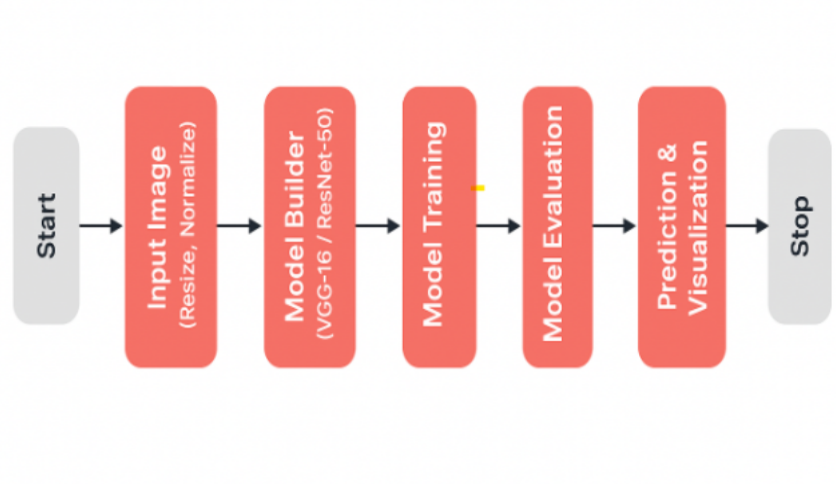


Figure 2 CNN Models flow chart

IV. VGG - 16

A. Dataset Loading and Preprocessing:

The implementation of the VGG-16 model starts with the preparation of the dataset with the use of the PyTorch library. The dataset of images of the road surface is divided into a training set and a validation set from torchvision.datasets. For the images to conform to the pre-built VGG-16 structure, they are all resized and normalized because of the statistics that ImageNet provides. Additionally, in the process of training, they are subjected to random resizing, cropping, and/or horizontal flipping, but in the validation set, they are randomized, with center cropping being used.

Loading and preprocessing of data was done with the following code:

```
data_transform = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

B. Data Loading Pipeline:

After setting up the preprocessing, the dataset is loaded from PyTorch's DataLoader, which is optimized for handling mini batches as well as randomizing samples. To remain within the computational requirement, a batch size of four is considered, with the potential for concurrent loading via worker threads for multi-threading.

```
data_dir = 'Dataset'

image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x), data_transform[x])
                  for x in ['train', 'val']
                  }
dataloaders = {
    x: torch.utils.data.DataLoader(
        image_datasets[x],
        batch_size=4,
        shuffle=True,
        num_workers=4
    )
    for x in ['train', 'val']
}
```

C. Model Architecture and Transfer Learning Strategy:

The VGG-16 convolutional neural network, pre-trained on the ImageNet dataset, is now integrated with transfer learning. All convolutional layers are frozen to retain previously learned low-level and mid-level visual features. To adapt the network for the pothole detection task, the final fully connected layer of the classifier is replaced with a two-class output layer corresponding to Normal Road and Pothole. Only this last layer is retrained, with notably reduced training time while retaining strong performance in classification.

```
model = models.vgg16(pretrained=True)
for param in model.features.parameters():
    param.requires_grad = False
for param in model.classifier[:-1].parameters():
    param.requires_grad = False
num_ftrs = model.classifier[6].in_features
model.classifier[6] = nn.Linear(num_ftrs, 2)
```

D. Training Configuration:

The cross-entropy loss function is used to train the network. The stochastic gradient descent with momentum is used as the optimizer, which is used only on the learnable classifier layer. The training is carried out for a total of ten epochs, and the network is run on the CPU devices.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.classifier[6].parameters(), lr=0.001, momentum=0.9)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)
print(device)
```

E. Model Training and Validation:

Training, validation, and updates occur iteratively for each epoch. In the training process, gradients are calculated and weights are adjusted, but in validation, gradients are calculated, and no adjustments are made to the weights. The accuracies and losses for classification are calculated for training as well as validation.

```
for epoch in range(num_epochs):
    print(f'Epoch {epoch+1}/{num_epochs}')
    print('-' * 10)

    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()
        else:
            model.eval()

        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in dataloaders[phase]:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)

                if phase == 'train':
                    loss.backward()
                    optimizer.step()
```

F. Function Table VGG-16:

Function Name	Type	Description
Models.vgg16()	Model constructor	sets up the VGG-16 convolutional neural network, which has already been trained and is used to classify potholes.

Train()/eval()	Mode Controller	During the learning and assessment stages, the VGG-16 model alternates between training and inference modes.
Training Loop	Main training routine	carries out parameter changes, backpropagation, forward propagation, and loss computation over several epochs.
Testing routine	Evaluation controller	Loads the fine-tuned model and performs inference on unseen test images to generate classification predictions.

G. Model Persistence:

After training completion, the fine-tuned VGG-16 model is saved.

```
torch.save(model.state_dict(), 'vgg16_finetuned.pth')
```

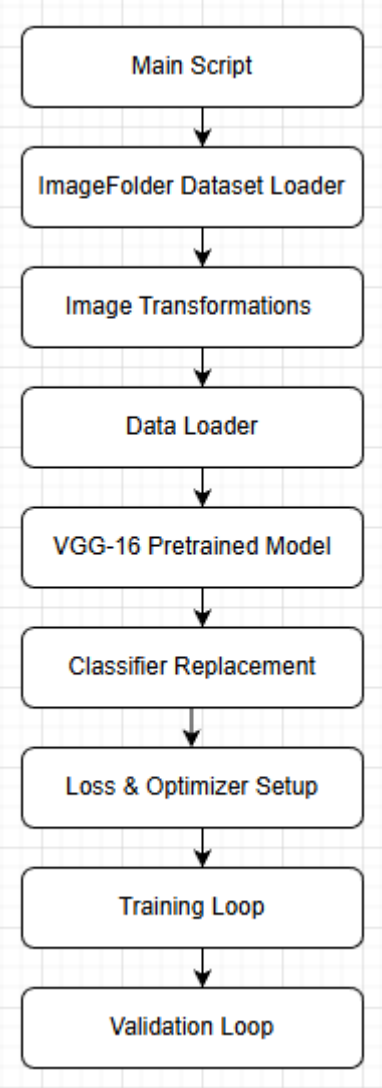


Figure 3 VGG-16 Call Graph

V. ResNet-50

A. Dataset Preparation and Preprocessing:

The labeled dataset used for training the models consists of two classes: Normal Road and Pothole. We split our dataset into training and validation directories. For increasing the generality of our model, the training images are augmented by randomly resizing the crop and flipping horizontally. Normalization on all images is performed by applying the ImageNet mean and standard deviation; this helps the network remain compatible with the pre-trained ResNet-50 architecture.

B. Data Loading and Batching:

PyTorch's Data Loader is used for the effective feeding of images to the network, both at training and validation. The mini-batch size is set to four for a good balance between memory constraints and stability of gradients. Shuffling is enabled for both the training and validation sets against ordering bias at optimization.

```
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4, shuffle=True, num_workers=4) for x in ['train', 'val']}
dataset_sizers = {x: len(image_datasets[x]) for x in ['train', 'val']}
print(f"Dataset sizes: {dataset_sizers}")
```

C. Model Initialization and Transfer Learning Strategy:

The weights are initialized from the pre-trained ResNet-50 architecture on ImageNet. Transfer learning is implemented by freezing all convolutional layers, while only the last fully connected layer is allowed to be trainable. This strategy results in considerably reducing training time while benefiting from robust low-level and mid-level feature representations learned from large-scale image data.

```
model = models.resnet50(pretrained=True)

for name, param in model.named_parameters():
    if "fc" in name:
        param.requires_grad = True
    else:
        param.requires_grad = False
```

D. Loss Function and Optimization:

The learning goal is specified by formulating a categorical cross-entropy loss function and optimized by applying a Stochastic Gradient Descent (SGD) with a momentum technique with a small learning rate to avoid diverging during the fine-tuning process.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

E. Training and Validation Procedure:

For this, the model is trained for ten epochs in an alternating training and validation phase. In the training phase, gradients are computed and model parameters updated by backpropagation, while in the validation phase it is used in inference mode without updating the gradients. For each epoch, loss and classification accuracy is computed separately for the training and validation sets to monitor the convergence and detect overfitting.

```
for epoch in range(num_epochs):
    print(f'Epoch {epoch+1}/{num_epochs}')
    print('-' * 10)

    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()
        else:
            model.eval()

        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in dataloaders[phase]:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)

                if phase == 'train':
                    loss.backward()
                    optimizer.step()
```

F. Model Persistence:

After training completion, the fine-tuned Resnet-50 model is saved.

```
torch.save(model.state_dict(), 'resnet50_finetuned.pth')
```

Function Name	Type	Description
models.resnet50()	Model constructor	Initializes the pretrained ResNet-50 architecture for pothole detection using residual learning.
Layer freezing logic	Training controller	Freezes convolutional layers and enables training only for the final fully connected layer to reduce overfitting
Training loop	Main training routine	uses labeled road photos from both training and validation datasets to carry out supervised learning.
Inference routine	Prediction controller	generates final class predictions by applying the trained ResNet-50 model to test images.

V. Results and Comparative Analysis:

Method	Accuracy	Time Taken
Fuzzy Logic	Positive: 75%, Normal: 60%	0.0164s, 0.27s
VGG-16	94.12%	9 min 3.6 sec
ResNet-50	90.44%	4 min 57.8 sec

The performance test in the three methods includes Fuzzy logic, VGG-16, and ResNet-50 by measuring their accuracy and computational time. Because of its high capability for deep and discriminative feature extraction, VGG-16 has yielded the highest accuracy, 94.12%. ResNet-50 trailed behind, with an accuracy of 90.44%, while residual connections enhance the efficiency of learning. However, this was low compared to VGG-16 because residual connections did not fit very well for this task. On the other hand, the Fuzzy logic approach yielded lower accuracy compared to the deep learning models: 75% of positive cases against 60% of normal cases. Though simpler, this method lacks representational power compared to these deep learning models, which impacts its classification performance.

That represents a speedup of the Fuzzy approach over the other techniques, which took considerably longer to compute both positive and normal cases, by 0.0164 s and 0.27 s, respectively. Therefore, it may be appropriate for applications in real time or with constrained resources.

Among deep learning models, ResNet-50 succeeded in being more efficient and finished its training in 4 minutes 57.8 seconds, while VGG-16 took more time.

- **Best accuracy:** VGG-16
- **Best trade-off between accuracy and time:** ResNet-50
- **Fastest approach:** Fuzzy logic

Overall, VGG16 provides superior performance when accuracy is weighed against the amount of time taken, while ResNet50 ensures that equal performance is obtained with high accuracy to a large extent, and with less computation time. Although less accurate, the Fuzzy technique has high speed; hence, it may be preferred for scanning purposes.

VI. Discussion and Future Works

The experiments are performed on a CPU-based computer system, which is a limitation towards faster training time. Future research directions include taking advantage of GPU acceleration for faster training times, and multi-threaded loading of data. The system needs to be scaled to run on a larger set of images. The images require better quality, varied lighting conditions, multiple angles, and a higher number of pothole instances. This is essential to generalize different roadway types.

Future work might include hybrid solutions that leverage the interpretability of fuzzy logic systems with the strengths of deep learning models. It would be an advantage for fuzzy logic systems to use CNNs to make predictions, which fuzzy logic rules might be able to refine, hence enhancing the precision of deep learning models. The fuzzy logic system might be optimized by enhancing feature selection, membership, and fuzzy rule adaptation, hence eliminating false positives.

VII. Literature Survey and State of the Art

Automated pothole/normal detection has been an active area of research for over two decades because of the impending risks it poses. The existing solutions available can be broadly classified into three categories: image processing solutions, fuzzy logic-based solutions, and deep learning solutions, with varying strengths and weaknesses. In the past, classical image processing methods such as thresholding, edge detection, texture analysis, and morphological processing were predominantly used in pavement distress detection. Koch and Brilakis showed in reference [1] how edge detection can identify intensity variations in pavement images. Although these classical methods are efficient, accurate, and fast, they are highly vulnerable to varying lighting conditions. In recent years, the use of deep convolutional neural networks (CNNs) has emerged as state-of-the-art for the detection of road surface defects. CNNs learn the hierarchical features from the raw images, providing robustness to changes in lighting conditions, textures, size, and image perspectives. Zhang et al. [3] illustrated that the use of deep learning for crack detection is far more accurate than the use of hand-crafted models. Fan et al. [4] demonstrated that the use of transfer learning with pre-trained models such as VGG-16 enhances the accuracy of detection even when limited images are available.

The systems used for identifying potholes on the commercial as well as municipal level are generally based on deep learning models that run on cloud platforms/GPUs, with a major emphasis on accuracy rather than interpretability. Fuzzy logic systems are typically never used on a standalone basis but are highly sought after in research work because of their interpretability and low computational complexity. This research proposal takes a stance somewhere in between approximated fuzzy systems and deep learning models that exhibit high performance by conducting a direct comparison of a Mamdani fuzzy inference system with a CNN transfer

learning architecture in VGG-16 as well as ResNet-50 on similar conditions. Unlike other research that has been particular to a certain approach alone. This research clearly establishes that although CNN models are precise, fuzzy logic is characterized by efficiency as well as interpretability.

VIII. Conclusion

The project followed the Waterfall methodology, beginning with data collection, where images containing potholes were gathered and organized into labeled datasets. This was followed by data processing, which included image preprocessing, normalization, and preparation for both fuzzy logic and deep learning pipelines. Secondly, we set up the fuzzy control tool, with fuzzy membership functions, linguistic variables, and Mamdani fuzzy inference rules. The final step towards the implementation of the fuzzy system involved combining the process of feature extraction, fuzzy inference, and the decision logic within a fuzzy system developed using a MATLAB environment. Finally, the output phase produced confidence heatmaps, detected pothole visualizations, and deduced performance results from it. The results from the experiments indicate that the fuzzy logic system has high interpretability, low computational complexity, and suitability for lightweight computations. In particular, the fuzzy logic system tends to misclassify normal road segments as pothole segments, particularly in complicated imaging conditions. On the contrary, the deep learning models performed better than the fuzzy logic system in terms of accuracy, precision, robustness, and overall performance. In conclusion, it is evident from this research that, although fuzzy logic systems are still useful for lightweight edge computing tasks, deep learning models are more ideal for applications involving large-scale real-world pothole detection systems. The proposed pothole detection framework has high

potential for application in intelligent transportation systems with the use of GPU acceleration, improved image datasets, and a hybrid algorithmic solution.

Reference:

[1] C. Koch and I. Brilakis, “Pothole detection in asphalt pavement images,” Adv. Eng. Inform., vol. 25, no. 3, pp. 507–515, 2011.
[2] N. B. C. Ahmed, S. Lahouar, C. Souani, and K. Besbes, “Automatic pothole detection from pavement images using fuzzy thresholding,” in Proc. Int. Conf. Control, Automation and Diagnosis (ICCAD), 2017, pp. 528–537.
[3] L. Zhang, F. Yang, Y. D. Zhang, and Y. J. Zhu, “Road crack detection based on deep convolutional neural network,” in Proc. IEEE Int. Conf. Image Process. (ICIP), 2016, pp. 3708–3712.
[4] R. Fan and X. Zhao, “Road surface defect detection using deep learning,” IEEE Access, vol. 8, pp. 149000–149013, 2020.
[5] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 770–778.

[6] A. Shaout, N. Png and Z. Li, "Concrete crack detection: Fuzzy logic vs neural network," in 2023, . DOI: 10.1109/ACIT58888.2023.10453768.

[7] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proc. Int. Conf. Learn. Representations (ICLR), San Diego, CA, USA, 2015.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” Advances in Neural Information Processing Systems (NeurIPS), vol. 25, pp. 1097–1105, 2012.

[9] J. Deng et al., “ImageNet: A large-scale hierarchical image database,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Miami, FL, USA, 2009, pp. 248–255.

[10] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[11] C. Szegedy et al., “Going deeper with convolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Boston, MA, USA, 2015, pp. 1–9.