CONTENTS

Title Page	i
Certificate by Supervisor	ii
Certificate by the HOD	iii
Declaration	iv
Acknowledgements	v
Abstract	vi
Chapter 1: Introduction	1
Chapter 2: Literature Survey	10
Chapter 3: Requirements & Feasibility Analysis	12
Chapter 4: Proposed Work	15
Chapter 5: Implementation	23
Chapter 6: Conclusion & Future Work	33

LIST OF FIGURES

Fig. 1.1	Evolution of Computer Vision	1
Fig. 1.2	Backpropagation	2
Fig. 1.3	Deep Learning	4
Fig. 1.4	Convolutional Neural Network	5
Fig. 1.5	Convolutional Layer	6
Fig. 1.6	Rectified Linear Unit	7
Fig. 1.7	Pooling Layer	8
Fig. 1.8	CNN Architecture	9
Fig. 4.1	Different Noise Types	17
Fig. 4.2	Preview of Dataset Label Images	18
Fig. 5.1	Squeezenet Architecture	26
Fig. 5.2	Code snippet of Squeeznet	27
Fig. 5.3	Layers of Squeezenet	27
Fig. 5.4	Transfer Learning CNN Training Progress	28
Fig. 5.5	Transfer Learning CNN Accuracy and Loss	28
Fig. 5.6	Vector Median Filter	30

TABLE OF CONTENTS

Table. 5.1	Image of NITM Hostel, Noise Images, and their respective Filter Images	29
Table. 5.2	Different errors to analyze Salt and pepper Noise	31

Chapter 1

Introduction

1.1 Research Background

In the late 1960s and early 1970s, computer vision became a separate field of study, originally concentrating on basic tasks like object recognition, edge detection, and 3D reconstruction. Pattern recognition relied on feature extraction and template matching techniques, whereas edge identification in images was mostly dependent on tools like the 1986-developed Canny edge detector. With the development of the Scale-Invariant Feature Transform (SIFT) and the broad use of Support Vector Machines (SVMs), tremendous progress was made in the 1980s and 1990s. These developments significantly improved the capacity to identify and characterize local features and carry out reliable classification.

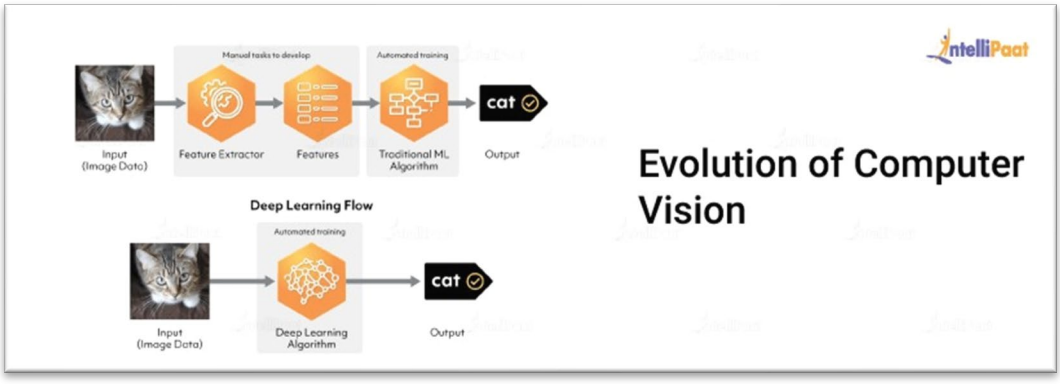


Fig. 1.1 Evolution of Computer Vision

With the invention of backpropagation in the 1980s, neural networks—which were first proposed in the 1940s—became practical. LeNet-5 by Yann LeCun, which demonstrated the potential of convolutional neural networks (CNNs) for handwritten digit recognition in 1998, was a significant turning point. Since AlexNet's victory in the 2012 ImageNet Challenge demonstrated the efficacy of deep CNNs for large-scale image classification, the field of CNN development has advanced significantly in the 2010s.

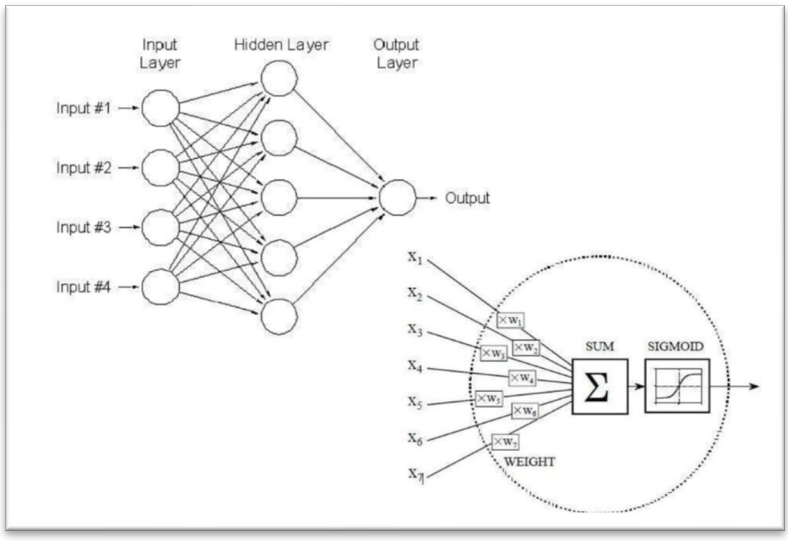


Fig. 1.2 Backpropagation [9]

Large datasets, novel designs, and enhanced hardware have propelled recent advances in CNN. 2015 saw the release of Residual Networks (ResNet), which greatly improved training and performance for deep models by addressing the vanishing gradient issue with residual connections. CNNs are already a mainstay in computer vision, helping with tasks like object recognition and image classification because of these advancements.

1.2 Problem statement

Picture Image noise significantly degrades digital image quality and increases the difficulty of computer vision tasks including object detection, segmentation, and recognition. The variety of real-world noise patterns is too much for conventional noise reduction approaches, which rely on manually defined features. As a result, there is insufficient suppression, and crucial details are lost. The objective of this project is to develop a versatile picture noise classifier that preserves crucial image information while efficiently identifying and removing different noise types, such as Gaussian, Salt-and-pepper, and Speckle noise, using convolutional neural networks (CNNs) and transfer learning.

- Create a sophisticated picture noise classifier that accurately classifies noise by combining CNNs with transfer learning.

- Create customized noise reduction modules for every kind of noise, making use of the hierarchical representations found in CNNs to achieve accurate noise reduction.

- Boost the efficiency of noise reduction and boost computer vision performance afterward.

1.3 Aim and Objective

The goal of this research is to use deep learning techniques to provide a novel method of managing image noise in digital imaging. The goals are to precisely classify common noise types like Gaussian, Salt-and-pepper, and Speckle noise by creating a novel image noise classifier with convolutional neural networks (CNNs) and introducing customized noise removal modules for each type of noise.

To achieve precise noise suppression while maintaining crucial image details, these modules will make use of the hierarchical representations that CNNs have learned. This will improve the accuracy and robustness of noise reduction in digital images and enable better performance in upcoming computer vision tasks.

1.4 Deep Learning

Artificial neural networks (ANNs), a type of algorithm that draws inspiration from the structure and operations of the brain, are the center of deep learning, a branch of machine learning. These networks, which are made up of linked layers of neurons or nodes, process information in intricate ways that let the system learn and make decisions on its own without the need for human interaction. This methodology has transformed domains such as natural language processing, autonomous driving, and picture and speech recognition by effectively managing substantial volumes of unstructured data and identifying complex patterns.

Training deep neural networks typically demands substantial data and computational resources. However, the training process has been made easier by the introduction of cloud computing and specialized technology, including graphics processing units (GPUs). Deep learning is predicted to become more widely used and to have a greater impact on a wider range of industries as processing power and data availability increase, spurring more

success and innovation. And also, Deep Learning is the subsection of Artificial Intelligence as shown in fig. 1.3.

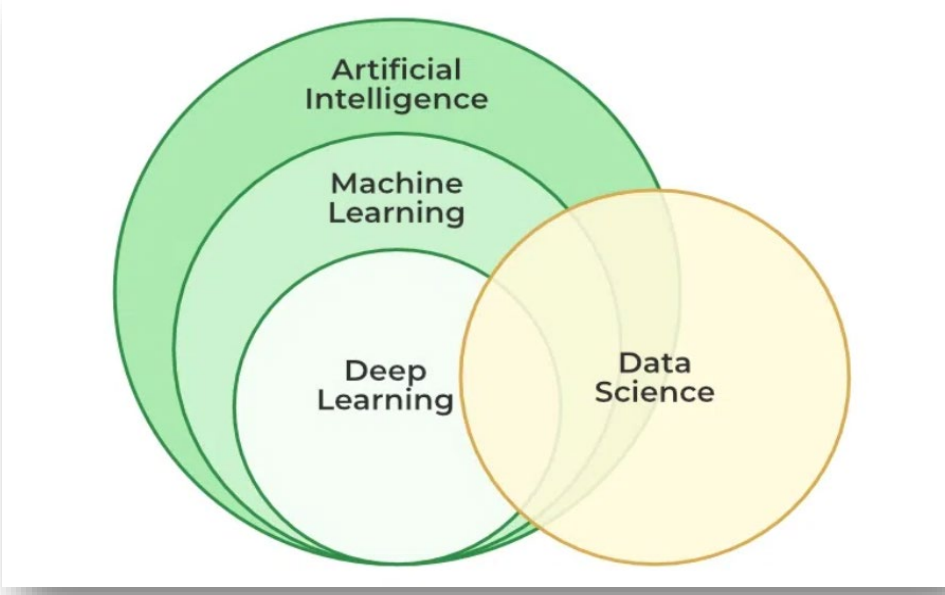


Fig. 1.3 Deep Learning

Today Deep learning has become one of the most popular and visible areas of machine learning, due to its success in a variety of applications, such as computer vision, natural language processing, and Reinforcement learning.

1.5 Convolutional Neural Networks

Convolutional Neural Network (CNN) is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example, data patterns play an extensive role in visual datasets like images or videos. Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers as shown in Fig. 1.4.

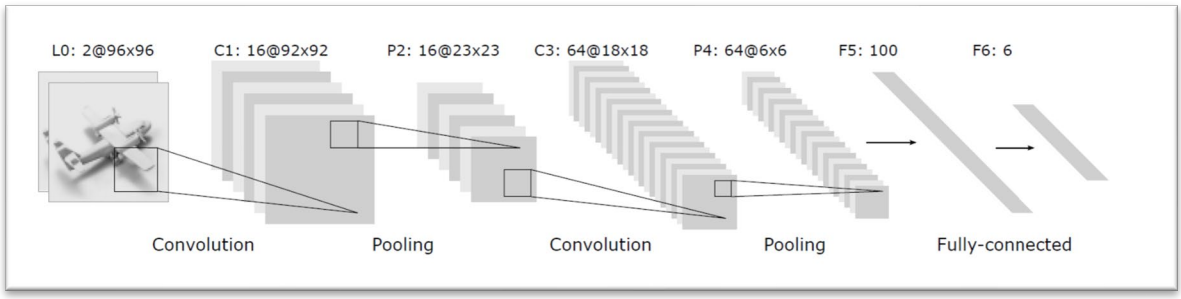


Fig. 1.4 Convolutional Neural Networks [3]

The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

1.5.1 How Convolutional Layers Work

Convolution Covnets, also known as neural networks, are neural networks with shared parameters. Assume you possess a picture. With its length, breadth (the image's dimension), and height (the channel—pictures often contain red, green, and blue channels—it can be represented as a cuboid. Consider applying a small neural network, also known as a filter or kernel, with, say, K outputs on a small patch of this image, and expressing the results vertically. When we slide the neural network over the entire image, we will have a new image with varying widths, heights, and depths. We now have extra channels, but they are smaller in width and height than the original R, G, and B channels. Convolution is the name of this operation.

1.5.2 Input Layers

It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.

1.5.3 Convolutional Layers

This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The

filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred to as feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12. Fig 1.5 describes about the convolution of the 32x32x3 image with the filter size of 5x5x3 to get the output size of 28x28x1.

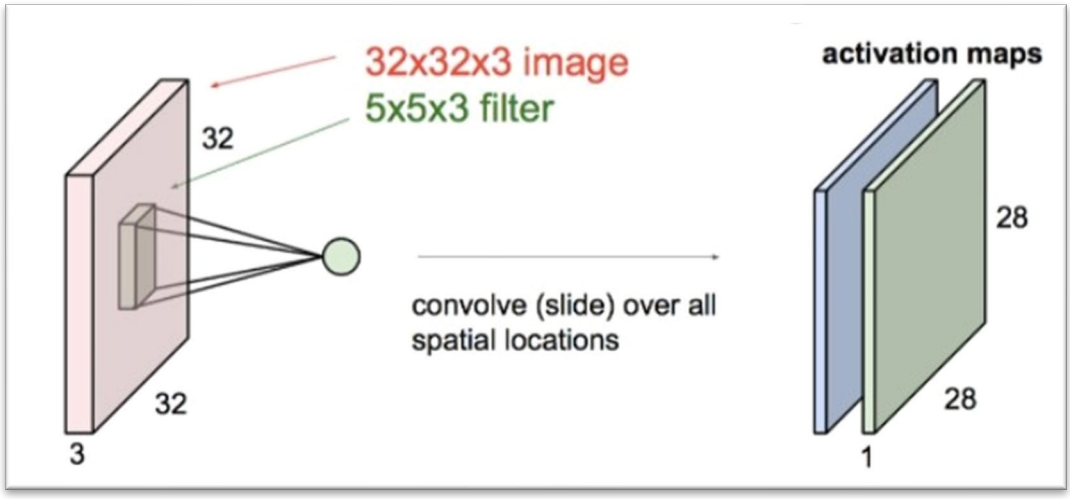


Fig. 1.5 Convolutional Layer

1.5.4 Activation Layer

By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max (0, x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12. Fig. 1.6 shows a ReLU (Rectified Linear Unit) layer in neural networks applies an element-wise activation function that replaces negative values with zero and keeps positive values unchanged.

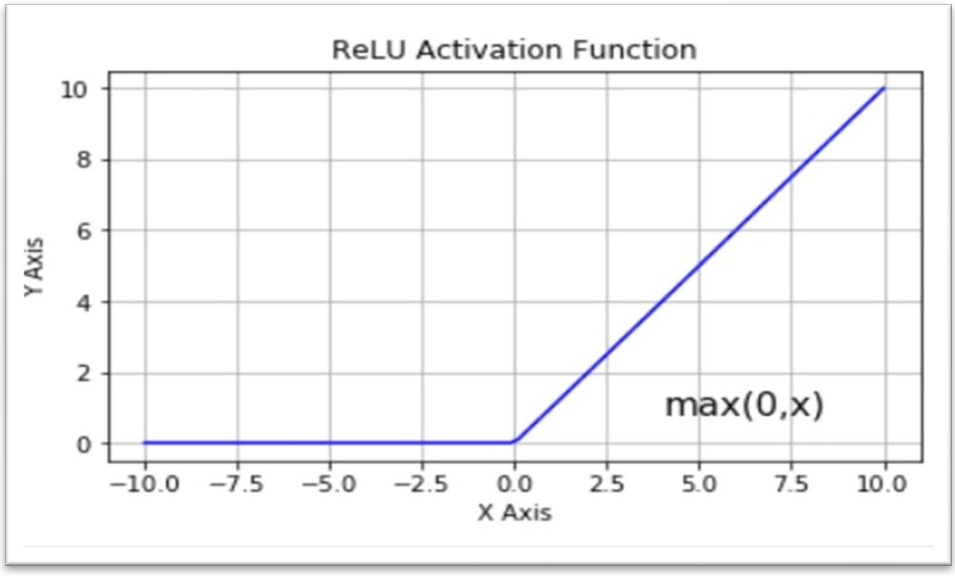


Fig. 1.6 Rectified Linear Unit (ReLU)

1.5.5 Pooling layer

The purpose of the pooling layers is to achieve spatial invariance by reducing the resolution of the feature maps. Each pooled feature map corresponds to one feature map of the previous layer. Their units combine the input from a small n × n patch of units, as indicated in Figure 1.4. This pooling window can be of arbitrary size, and windows can overlap.

We evaluate two different pooling operations: max pooling and subsampling. The subsampling functions.

$$a_j = \tanh\left(\beta \sum_{NxN} a_i^{nxn} + b\right) \qquad (1.1)$$

takes the average over the inputs, multiplies it with a trainable scalar β, adds a trainable bias b, and passes the result through the non-linearity. The max pooling functions.

$$a_j = \underset{NxN}{max} a_i{}^{nxn} u(nxn)) \qquad (1.2)$$

applies a window function u(x, y) to the input patch, and computes the maximum in the neighborhood. In both cases, the result is a feature map of lower resolution.

Fig 1.7 explains the pooling operation performed by wit stride two which is single depth slice matrix. And it uses the equation 1.2 to perform it

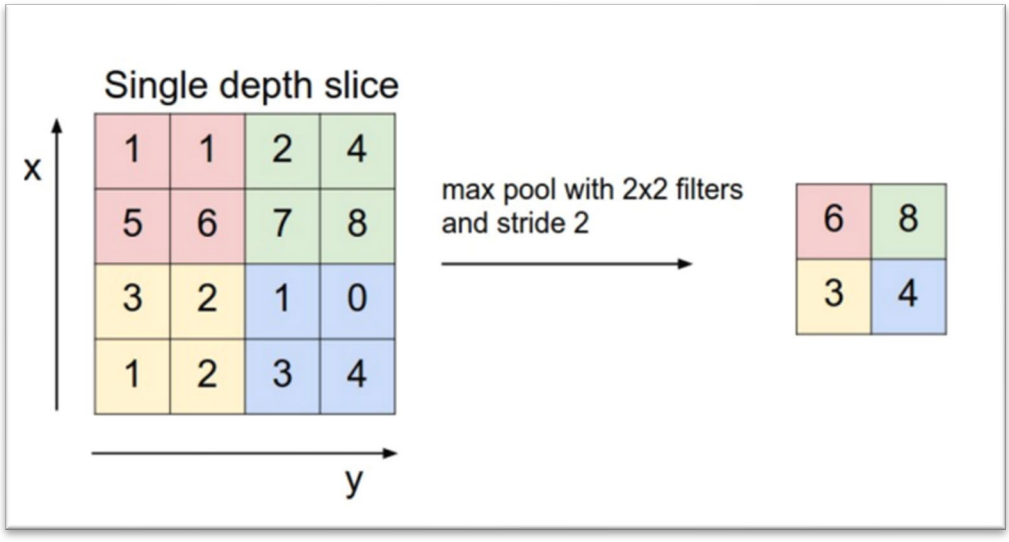


Fig. 1.7 Pooling Layer

1.5.6 Flattening

The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.

1.5.7 Fully Connected Layers

It takes the input from the previous layer and computes the final classification or regression task.

Convolutional Neural Networks (CNNs) are a class of deep learning algorithms particularly well-suited for processing and analyzing visual data. Inspired by the human visual cortex, CNNs are designed to learn spatial hierarchies of features automatically and adaptively from input images. They consist of multiple layers,

including convolutional layers that apply filters to the input image to extract features such as edges, textures, and shapes. Pooling layers then reduce the spatial dimensions of these feature maps, which helps in minimizing computational complexity and controlling overfitting. Finally, fully connected layers interpret these features to classify the image or detect objects. CNNs have revolutionized computer vision tasks, leading to breakthroughs in image recognition, object detection, and facial recognition, among other applications. Their ability to learn directly from raw pixel data without the need for manual feature extraction has made them a powerful tool in the field of artificial intelligence.

Chapter 2

Literature Survey

2.1 Literature Review

[12] Noise is caused due to various sources which include many external causes in the transmission system and environmental factors which includes noise like Gaussian, Poisson, Blurred, Speckle, and salt-and-pepper noise. The noise removing [4,16] method has become an important factor in medical imaging applications and the most commonly used filters preserve the edges. Two types of models are used for de-noising linear models [12] and non-linear models [12]. Most of the time linear models are experimented with because of their speed even though it has the limitation of not being able to preserve the edges of an image in an efficient way.

The classical vector filters are the extensions of median filtering of grayscale images [15], including vector median filter (VMF) [16], basic vector directional filter (BVDF) [17], and directional-distance filter (DDF) [18]. These filters are based on reduced ordering principles and output the samples inside filter windows that minimize different aggregated color distances. However, these classical vector filters introduce a maximum amount of smoothing when processing each pixel, regardless of whether the pixel is contaminated by noise. Therefore, some natural improvements have been developed introducing more effective switching vector filters [19–22] and weighted vector filters [23, 24]. Weighted vector filters perform different amounts of smoothing according to different weighting coefficient sets. Switching vector filters process only the noisy pixels detected, leaving the noise-free ones unaltered.

To address these limitations, adaptive filtering techniques [4,16] have been developed. For instance, structure-adaptive filters, which adjust the filtering window based on local image characteristics, have shown promise. Methods utilizing quaternion algebra for local orientation estimation [15] have been particularly effective in maintaining the structural integrity of images while reducing noise.

More recently, the integration of deep learning, specifically convolutional neural networks (CNNs) [1,4], into noise reduction frameworks has demonstrated significant improvements. CNNs possess strong learning capabilities and can effectively detect and differentiate between noise and image details. For example, Jin et al. introduced a deep CNN-based impulse noise detector [7] that significantly enhances the performance of structure-adaptive vector filters. This approach not only boosts noise suppression but also preserves essential image details by leveraging the hierarchical representations learned by CNNs.

In our proposed methodology, we extend the application of deep learning [9] to develop a novel image noise classifier utilizing transfer learning with CNNs. This classifier is designed to categorize three prevalent types of noise: Gaussian, Salt-and-pepper, and Speckle noise [4]. By leveraging transfer learning, our system efficiently learns discriminative features from noisy image data, enhancing the robustness and accuracy of noise classification.

Moreover, we introduce a noise removal [1,4] module tailored to address each specific type of noise. Utilizing the hierarchical representations learned by CNN, our system distinguishes between noise and signal components, enabling precise noise suppression while preserving essential image details. This approach builds upon the strengths of existing methods while addressing their limitations, offering a comprehensive solution for image noise reduction [1,4] in real-world applications [12].

Chapter 3

Requirement and Feasibility Analysis

3.1 Requirements Analysis

3.1.1 Functional Requirements

- Image Input: A variety of image file formats, such as JPEG and PNG, should be supported by the system.
- Noise Classification: Analysing and categorizing the many types of noise (e.g., Gaussian, Salt-and-pepper, Speckle) in an image is the main function.
- Accuracy Reporting: For every sort of noise classification, the system ought to offer a probability or confidence score expressing how likely it is.
- Noise Removal: Use a noise removal module designed to deal with each unique kind of noise to ensure accurate noise reduction while maintaining important image details.
- User Interface: Offer an easy-to-use interface for uploading photos, seeing the outcomes of noise categorization, and using noise reduction.
- Batch Processing: Ability to handle several photos at once.

3.1.2 Non-Functional Requirements

- Performance: Photos should be processed by the system fast, preferably in a matter of seconds.
- Scalability: As the system grows, it should be able to accommodate more users and photos without experiencing appreciable performance drops.
- Reliability: There should be little downtime, low mistake rates, and a sturdy system.
- Security: Make sure that results and uploaded photos are handled and kept safely using the proper encryption and access controls.
- Compatibility: Should work across different operating systems and devices, particularly within Matlab.

3.1.3 Technical Requirements

3.1.3.1. Hardware Requirements

- Processor: A multi-core processor (e.g., Intel i7 or above, AMD Ryzen 7 or above).
- RAM: At least 16 GB of RAM, 32 GB or more recommended for large batch processing.
- GPU: A dedicated GPU (e.g., NVIDIA RTX 2070 or higher) is highly recommended for faster deep-learning model inference.
- Storage: At least 500 GB of SSD storage for quick read/write operations and sufficient space for storing datasets and results.

3.1.3.2. Software Requirements

- Operating System: Linux (Ubuntu preferred), Windows 10, or macOS.
- Matlab: Matlab 2020a or higher.
- Deep Learning Toolbox: For Network Analyser and Squeezenet.

3.2 Feasibility Analysis

3.2.1 Technical Feasibility

The group is proficient in MATLAB programming, computer vision, and deep learning. The necessary libraries and frameworks are compatible with MATLAB, enabling the building of custom CNNs and SqueezeNet transfer learning. Sufficient computational resources, including GPUs, are accessible for both inference and training, guaranteeing effective processing. The stable environment of MATLAB makes it easier to design and test deep learning models and image processing algorithms interactively. Implementation of image noise reduction and classification models is dependable and efficient due to their maturity.

3.2.2 Economic Feasibility

Investing in software (MATLAB licenses, image processing toolboxes) and hardware (GPUs, high-performance servers). the price of employing qualified workers with experience in image processing and deep learning, as well as the time needed for development. ongoing expenditures for system upkeep, data storage, and cloud services. Possibility of a large return on investment because of the growing need for efficient picture noise reduction techniques across a range of industries (surveillance, medical imaging, and photography).

3.2.3 Operational Feasibility

If the system is made to be user-friendly, with an easy-to-use interface for submitting images and obtaining noise classification results, then little training is needed. For smooth functioning, the system can be incorporated into already-existing workflows. scalable in response to an increase in users and image volume. To accommodate novel noise patterns, maintenance entails routinely updating filters and models for noise classification. Assure adherence to ethical standards for the handling and processing of photographs as well as data protection laws.

Chapter 4

Proposed Work

4.1 Data Collection

For the successful development of our image noise classification and removal system, acquiring a robust and diverse dataset is crucial. This section outlines the process of selecting, obtaining, and preparing the dataset for our project.

The dataset utilized in this research was sourced from the CamVid dataset, a widely recognized benchmark dataset in the field of computer vision. The CamVid dataset offers a rich collection of high-resolution images with corresponding pixel-wise semantic annotations, making it well-suited for our noise classification and removal tasks.

The decision to use the CamVid dataset was motivated by its suitability, quality, and availability for our specific research objectives. Its diverse scenes, varying lighting conditions, and complex urban environments provide an excellent foundation for training and evaluating our noise classification and removal algorithms.

The dataset comprises original images alongside their corresponding noisy counterparts, artificially generated by adding varying levels of Gaussian, Salt-and-pepper, and Speckle noise. This augmentation process ensures a comprehensive representation of real-world noise patterns and challenges commonly encountered in digital imaging.

To facilitate robust model training and evaluation, the dataset was partitioned into three subsets:

- Training Set: Approximately 70% of the dataset was allocated for training our noise classification and removal models. Augmentation techniques, such as adding synthetic noise to original images, were employed to prevent overfitting and enhance the model's ability to generalize across different noise types and intensities.
- Validation Set: 15% of the dataset was reserved for validating the performance of our models during training. This subset enabled fine-tuning of hyperparameters and facilitated the selection of the best-performing models.
- Test Set: The remaining 15% of the dataset was held out for final testing and comprehensive evaluation of the trained models' effectiveness in noise classification and removal. This independent test set ensured an unbiased assessment of model performance on unseen data.

4.2 Noise Injection Methodology

The injection of synthetic noise into the original CamVid dataset was performed using MATLAB's image processing toolbox. The following steps outline the noise injection.

4.2.1 procedure:

Image Selection: Original images from the CamVid dataset were randomly sampled to serve as the basis for noise injection.

Fig. 4.1 Different Noise Types

Noise Generation and Addition: Synthetic noise patterns corresponding to Gaussian, Salt-and-pepper, and Speckle noise were generated using appropriate noise models. MATLAB's imnoise function was utilized for injecting noise into the selected original images. For example, to add Salt-and-pepper noise with a variable density, the following MATLAB code snippet was employed:

J = imnoise(I, 'salt & pepper', density);

where I represent the original image and density is a parameter controlling the noise intensity.

J = imnoise(I,'speckle',var_speckle);

where var_speckle represents the variance as a parameter controlling the noise intensity.

J = imnoise(I,'gaussian',m,var_gauss);

where m specifies the average value around which the noise values are centered.var_speckle represents the variance as a parameter controlling the noise intensity. The above function are used to add the noise in images as shown in Fig. 4.1.

4.3 Noise Classification Approach

In this section, we present our approach to image classification using a custom-built convolutional neural network (CNN) architecture. Unlike transfer learning, which utilizes pre-trained models, our methodology involves designing and training a CNN from scratch to classify images into different noise categories—Gaussian, Salt-and-pepper, and Speckle noise.

4.3.1 Dataset Preparation

We begin by loading the dataset containing images with various types of noise, sourced from the specified directory. The dataset is structured using an image datastore object, enabling seamless integration with MATLAB's image processing and deep learning functionalities. Each image is resized to a standard size of 227x227 pixels to ensure uniformity across the dataset.I have used the camvid dataset in fig. 4.2.

Fig. 4.2 Preview of Dataset Label Images

4.3.2 Exploratory Data Analysis

To gain insights into the dataset, we visualize a subset of images using MATLAB's plotting capabilities. Random images are sampled from the dataset and displayed to provide a qualitative understanding of the data distribution and characteristics.

4.3.3 Data Preprocessing

Before model training, the dataset is preprocessed to facilitate effective learning. This includes resizing images to a consistent size, which is essential for compatibility with the CNN architecture. Additionally, normalization and augmentation techniques may be applied to enhance model generalization and robustness.

4.3.4 Neural Network Architecture

Our custom CNN architecture consists of several layers designed to extract relevant features from input images and perform classification. The architecture comprises the following layers:

Input layer: Accepts input images with dimensions 227x227x3 representing height, width, and RGB channels.

Convolutional layers: Responsible for feature extraction through convolutional operations, followed by batch normalization and rectified linear unit (ReLU) activation functions to introduce non-linearity.

Max pooling layers: Downsample feature maps to reduce spatial dimensions and extract dominant features.

Fully connected layer: Performs classification based on the extracted features, followed by a softmax layer to generate probability scores for each class.

Classification layer: Assigns labels to input images based on the highest probability score.

4.3.5 Training Procedure

The CNN is trained using the specified training options, including stochastic gradient descent with momentum (SGDM) optimizer, maximum epochs, mini-batch size, and initial learning rate. During training, the network learns to minimize the classification error by adjusting its parameters iteratively based on the provided training data.

4.3.6 Model Evaluation

Upon training completion, the trained CNN is evaluated on a separate validation dataset to assess its classification performance. The predicted labels are compared with ground truth labels, and the classification accuracy is computed as the ratio of correctly classified images to the total number of validation images.

4.3.7 Results and Analysis

The classification accuracy obtained on the validation dataset serves as a metric for evaluating the effectiveness of our custom CNN approach. Additionally, qualitative analysis through visual inspection of classification results aids in understanding the model's ability to distinguish between different noise types.

4.4 Transfer Learning

Transfer learning is a powerful technique that leverages pre-trained models to improve the efficiency and accuracy of classification tasks, especially when dealing with limited datasets. In this section, we describe the methodology employed for classifying different types of noise—Gaussian, Salt-and-pepper, and Speckle noise—using transfer learning with the SqueezeNet[6] architecture. In transfer learning, the process is nearly identical to that of a custom-built CNN, except for the modification of the following two steps.

4.4.1 Transfer Learning Setup

The transfer learning process involves modifying the pre-trained SqueezeNet model to suit our specific classification task. Key steps include:

- **Loading the Pre-trained Model:** SqueezeNet[6] is selected for its compact architecture and efficiency.
- **Layer Modification:** The final layers of SqueezeNet[6] are replaced with new layers tailored to our classification task[4]. This includes:

A new convolutional layer to match the number of noise categories.

1. Convolutional Layer
2. A ReLU activation layer.
3. A global average pooling layer.
4. A softmax layer.
5. A classification output layer.

4.4.2 Training Procedure

The modified network is trained using the training dataset. Training options are specified, including the stochastic gradient descent with momentum (SGDM) optimizer, learning rate, mini-batch size, and the number of epochs. The training process involves fine-tuning the pre-trained weights to adapt to our specific classification task.

4.5 Noise Removal

In addition to noise classification, addressing the detrimental effects of noise on image quality is crucial for various computer vision applications. This section outlines the methodology and techniques employed for noise removal in the context of our project.

4.5.1 Preprocessing

Before delving into noise removal techniques, the dataset underwent preprocessing to ensure consistency and compatibility with the subsequent processing steps. This included resizing all images to a standardized resolution of 227x227 pixels to match the input requirements of the neural network models utilized for noise removal. This is an optional step.

4.5.2 Image Denoising Algorithms

To effectively remove noise from the images, we employed a combination of classical image denoising algorithms and deep learning-based approaches. Specifically, three types of noise—Gaussian, Salt-and-pepper, and Speckle—were targeted using appropriate denoising techniques:

Gaussian Noise Removal: The Wiener filter, a classical denoising method based on signal-to-noise ratio estimation, was utilized to attenuate Gaussian noise while preserving image details.

Salt-and-pepper Noise Removal: Given its impulse-like nature, salt-and-pepper noise was effectively mitigated using the median filter, which replaces each pixel's value with the median value within a local neighborhood.

Speckle Noise Removal: Speckle noise, characteristic of ultrasound and radar imagery, was addressed using the Lee filter or weiner2 filter, which leverages statistical properties of speckle noise to suppress its effect on image quality.

Chapter 5

Implementation

5.1 Data Preprocessing

Effective data preprocessing is crucial for the success of any deep learning project, as it ensures that the input data is in the best possible condition for model training and evaluation. This section outlines the preprocessing steps applied to the CamVid dataset, which has been augmented with synthetic noise, to prepare it for the classification and noise removal tasks.

5.1.1 Image Resizing

The original images from the CamVid dataset vary in size and resolution. To ensure uniformity and compatibility with the input requirements of our convolutional neural network (CNN) architectures, all images are resized to 227x227 pixels. This resizing operation is performed using the imresize function in MATLAB, as shown below:

imds.ReadFcn = @(filename) imresize(imread(filename), [227, 227]);

Resizing standardizes the input dimensions, facilitating efficient processing by CNN.

5.1.2 Noise Injection

To create a comprehensive dataset for training and evaluating the noise classification model, synthetic noise is injected into the images. Three types of noise—Gaussian, Salt-and-pepper, and Speckle—are added to the images using MATLAB's imnoise function. The noise injection process ensures that the dataset encompasses a wide range of noise levels and patterns, simulating real-world scenarios.

Gaussian Noise: Injected using:

J = imnoise(I, 'gaussian', rand, var_gauss);

Here, rand generates random values for the mean and var_gauss controls the variance of the noise.

Salt-and-Pepper Noise: Injected using:

J = imnoise(I, 'salt & pepper', rand);

The rand parameter determines the noise density, simulating random occurrences of black and white pixels.

Speckle Noise: Injected using:

J = imnoise(I, 'speckle', var_speckle);

The var_speckle parameter controls the variance of the multiplicative noise.

5.1.3 Data Augmentation

To enhance the generalization capabilities of the model and prevent overfitting, data augmentation techniques are applied to the training set. These techniques include random cropping, rotation, flipping, and scaling. Data augmentation introduces variability in the training data, allowing the model to learn more robust features that are invariant to these transformations.

5.1.4 Normalization

Normalization is a vital preprocessing step that scales the pixel values of the images to a standard range, typically [0, 1] or [-1, 1]. This ensures that the input data is centered and has a consistent scale, which helps in speeding up the convergence of the training process and improving model performance. In our project, pixel values are normalized to the [0, 1] range by dividing by 255.

5.2 Implementation of Squeezenet Model

SqueezeNet is a compact convolutional neural network (CNN) architecture designed to achieve high accuracy while maintaining a significantly reduced model size. Developed by

researchers at DeepScale, SqueezeNet's primary objective is to reduce the number of parameters in a neural network without compromising performance. This characteristic makes SqueezeNet highly suitable for deployment on devices with limited computational resources, such as mobile devices and embedded systems.

The core innovation of SqueezeNet lies in its architectural design, which introduces the concept of Fire modules. A Fire module consists of two main components: a squeeze layer and an expands layer. The squeeze layer uses 1x1 convolutions to reduce the number of input channels, thereby "squeezing" the data. The expand layer follows the squeeze layer and applies a combination of 1x1 and 3x3 convolutions to the squeezed data, effectively "expanding" it to produce the desired number of output channels. This use of Fire modules allows SqueezeNet to significantly reduce the total number of parameters, making the model smaller and more efficient compared to traditional CNN architectures such as AlexNet or VGGNet.

One of the primary benefits of SqueezeNet is its parameter reduction, achieved through the extensive use of 1x1 convolutions in the squeeze layers. This reduction results in a smaller model size, which facilitates easier compression and deployment on hardware with constrained memory and processing capabilities. Despite its compact size, SqueezeNet maintains a high level of performance, capable of achieving comparable accuracy to larger networks. This makes SqueezeNet a versatile choice for various image classification tasks, balancing the trade-off between model size and accuracy effectively.

Furthermore, the fewer parameters in SqueezeNet translate to reduced computational complexity, enabling faster inference times. This computational efficiency is particularly beneficial for real-time applications where quick responses are critical. The overall architecture of SqueezeNet, which typically includes an initial convolution layer, stacked Fire modules, interspersed max-pooling layers, and a final convolutional layer followed by global average pooling and a softmax layer, ensures that it remains both flexible and performant.

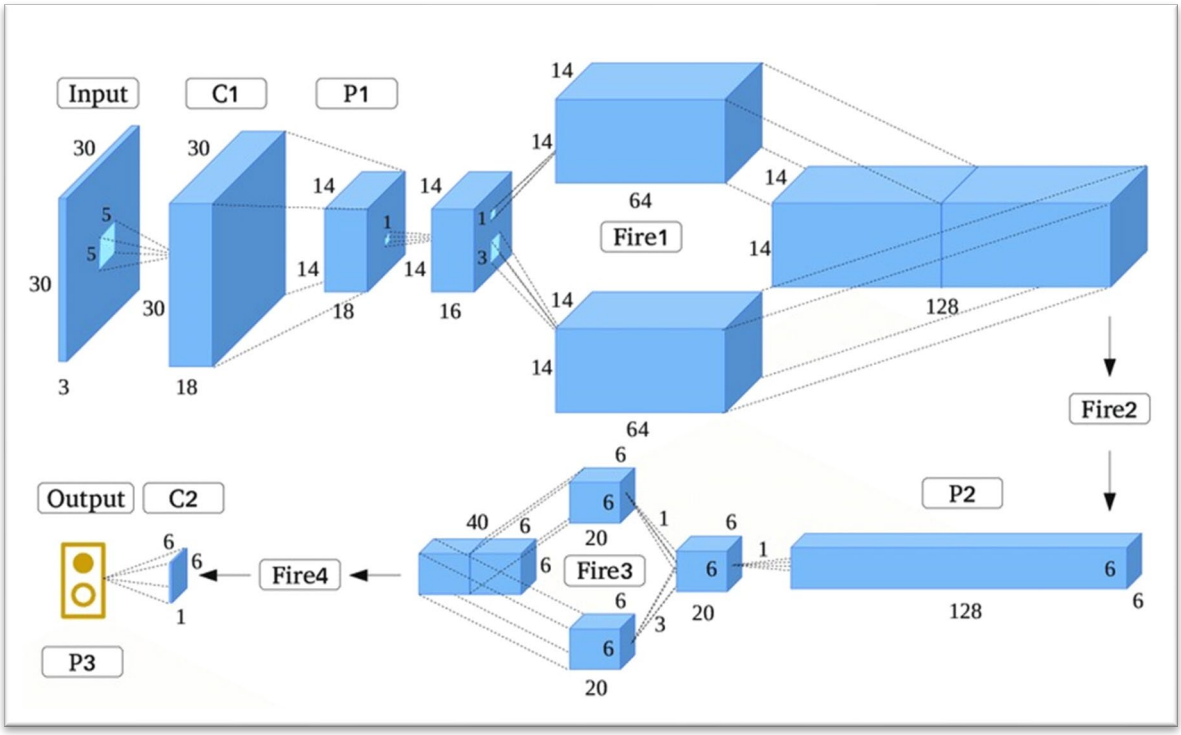


Fig. 5.1 Squeezenet Architecture

5.3 Training the model

For noise classification, we adapted the pre-trained SqueezeNet model using transfer learning. Images were resized to 227x227 pixels and split into training and validation sets. We modified SqueezeNet by replacing its final layers with new ones for classifying Gaussian, Salt-and-pepper, and Speckle noise. The network was trained using stochastic gradient descent with momentum (SGDM) for seven epochs, with a mini-batch size of 32 and an initial learning rate of 1e-4. The training process was monitored via a training-progress plot, and the model achieved high accuracy on the validation set.

```
%% Load Pre-trained Network (AlexNet)
net = squeezenet;

%% Modify Network for Transfer Learning
% Create the layer graph
lgraph = layerGraph(net);

% Find the layer to replace
oldLayer = lgraph.Layers(end);

% Number of classes (noises)
numClasses = numel(categories(imdsTrain.Labels));

% Define the new layers
newConvLayer = convolution2dLayer(1, numClasses, 'WeightLearnRateFactor', 10, 'BiasLearnRateFactor', 10, 'Name', 'new_conv');
newConvLayer.Weights = randn([1, 1, 512, numClasses])*0.01;
newReluLayer = reluLayer('Name', 'new_relu');
newPoolLayer = globalAveragePooling2dLayer('Name', 'new_pool');
newSoftmaxLayer = softmaxLayer('Name', 'new_softmax');
newClassificationLayer = classificationLayer('Name', 'new_classoutput');

% Replace layers
lgraph = replaceLayer(lgraph, 'conv10', newConvLayer);
lgraph = replaceLayer(lgraph, 'relu_conv10', newReluLayer);
lgraph = replaceLayer(lgraph, 'pool10', newPoolLayer);
lgraph = replaceLayer(lgraph, 'prob', newSoftmaxLayer);
```

Fig. 5.2 Code snippet of Squeezenet training

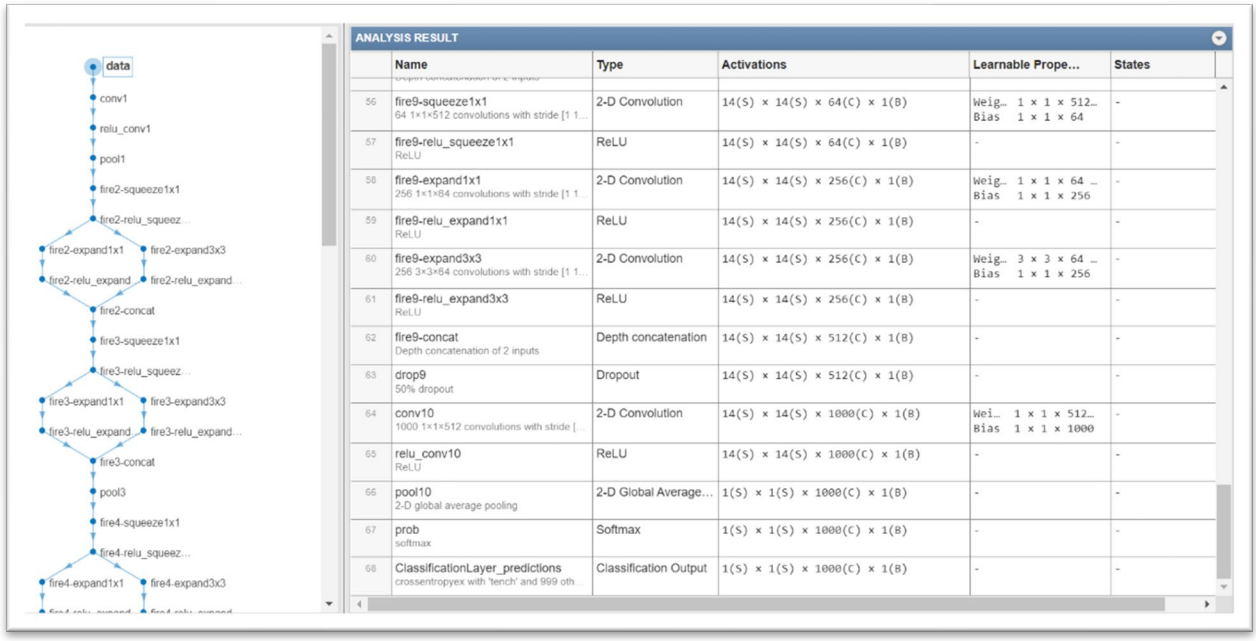


Fig. 5.3 Layers of Squeezenet

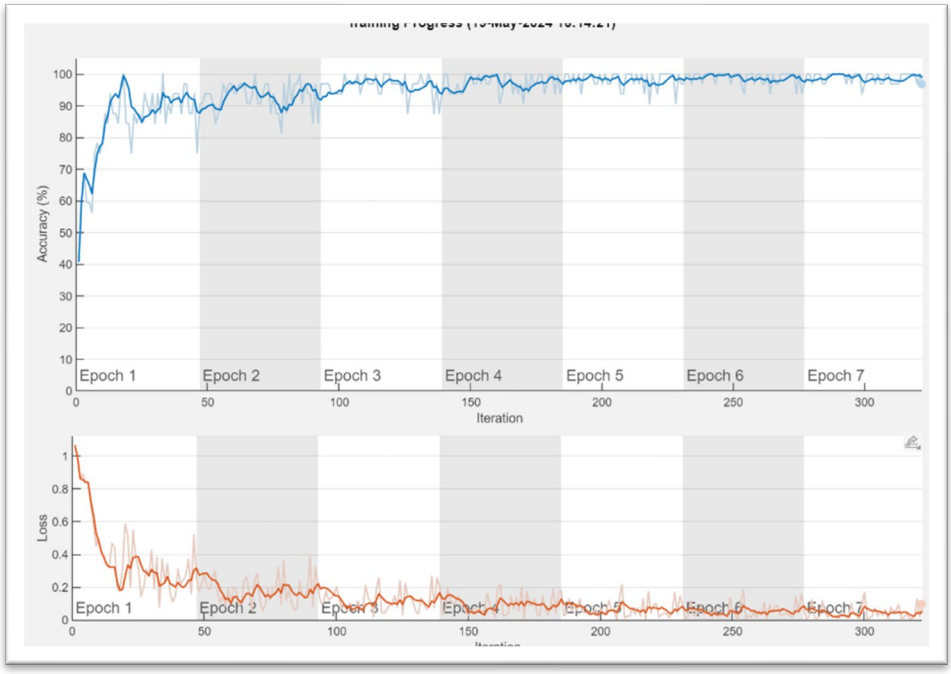
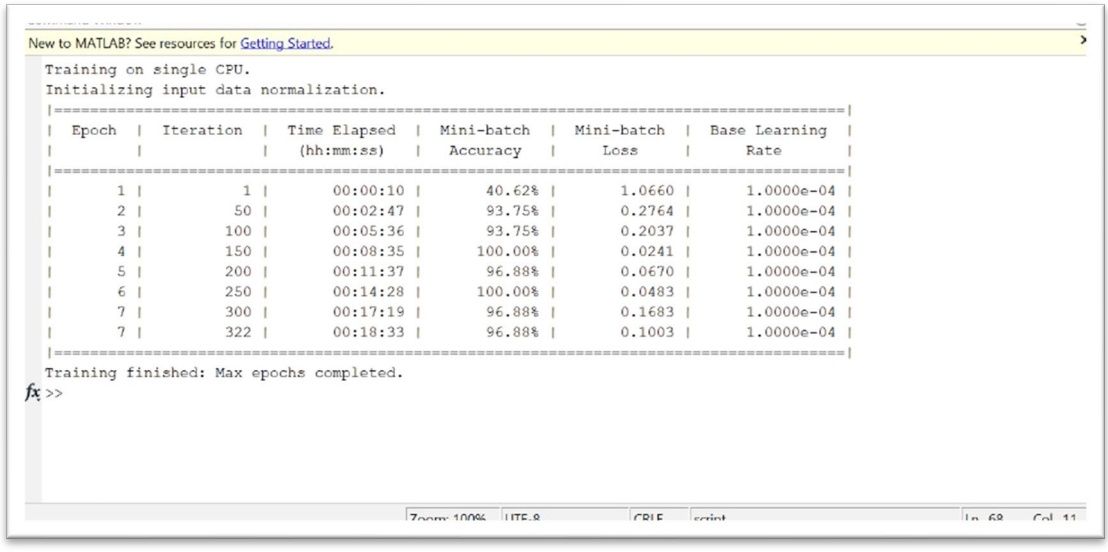


Fig. 5.4 Transfer Learning CNN Training Progress



New to MATLAB? See resources for Getting Started.

```
Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:10 |       40.62% |       1.0660 |      1.0000e-04 |
|       2 |          50 |       00:02:47 |       93.75% |       0.2764 |      1.0000e-04 |
|       3 |         100 |       00:05:36 |       93.75% |       0.2037 |      1.0000e-04 |
|       4 |         150 |       00:08:35 |      100.00% |       0.0241 |      1.0000e-04 |
|       5 |         200 |       00:11:37 |       96.88% |       0.0670 |      1.0000e-04 |
|       6 |         250 |       00:14:28 |      100.00% |       0.0483 |      1.0000e-04 |
|       7 |         300 |       00:17:19 |       96.88% |       0.1683 |      1.0000e-04 |
|       7 |         322 |       00:18:33 |       96.88% |       0.1003 |      1.0000e-04 |
|=========================================================================================|
Training finished: Max epochs completed.
>>
```

Fig. 5.5 Transfer Learning CNN Accuracy and Loss

Table. 5.1 Image of NIMN Manipur, Noise Images and Their Respective Filter Images

5.4 Implementation of Noise Removal

This section outlines the implementation of noise removal techniques to enhance image quality for computer vision tasks. Classical denoising filters (Wiener, median, and Lee) were applied to mitigate Gaussian, salt-and-pepper, and speckle noise. Deep learning-based approaches using convolutional neural networks (CNNs) with transfer learning were explored for noise reduction. The evaluation involved quantitative metrics (NCD, MAE, MSE, PSNR) and qualitative visual assessments.

5.4.1 Vector Median Filter

The vector median filter [4] is a technique used for filtering multidimensional data, particularly in the context of image processing. Unlike scalar median filters that operate on scalar values, vector median filters are designed to handle vector-valued data, such as color images.

Simplified Explanation: -

1. For each pixel, consider a neighborhood of pixels around it.
2. Calculate the distance between all pairs of vectors in this neighborhood.
3. Select the vector that has the smallest sum of distances to all other vectors.
4. Replace the original pixel value with the median vector.

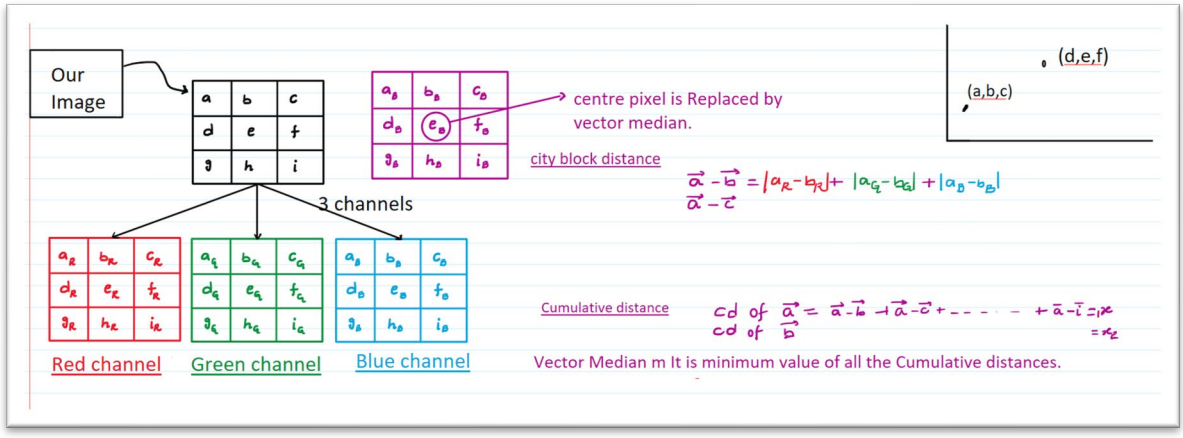


Fig. 5.6 Vector Median Filter

Table. 5.2 Different errors to analyse Salt and pepper Noise

Salt & pepper Nosie	NCD	MSE	PSNR	MAE
0.1	0.0025	2.9714	43.4012	0.2584
0.2	0.0050	5.5792	40.6651	0.5094
0.3	0.0077	8.8419	38.6654	0.7868
0.4	0.0107	13.0153	36.9862	1.0965
0.5	0.0141	18.3422	35.4963	1.4493
0.6	0.0183	26.2610	33.9377	1.8738
0.7	0.0237	38.2298	32.3068	2.4140
0.8	0.0309	56.4207	30.6164	3.1309
0.9	0.0425	88.8734	28.6431	4.2528

5.4.1 Weiner2 Filter

The Wiener filter is a classical signal processing technique used for image restoration and denoising. It aims to minimize the mean square error between the original image and the filtered image by leveraging statistical properties of both the image and the noise present in it.

The theory behind the Wiener filter revolves around the concept of signal and noise estimation in the frequency domain. It assumes that the noise in the observed image is additive and follows a stationary random process. The filter operates by estimating the power spectral density (PSD) of the original image and the noise, allowing for the calculation of the Wiener filter's transfer function.

In MATLAB, the wiener2 function implements the Wiener filter for 2D images. It takes the noisy input image and estimates the local mean and variance of the noise-free image. Using these estimates, along with a user-defined neighborhood size, the function computes the filter weights to adaptively suppress noise while preserving image details.

$$\mu(i,j) = \frac{1}{MN}\sum_{(m,n)\in Window} I(m,n) \tag{5.1}$$

$$\sigma^2(i,j) = \frac{1}{MN}\sum_{(m,n)\in Window}(I(m,n) - \mu(i,j))^2 \tag{5.2}$$

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This project tackled image noise through a comprehensive framework encompassing noise classification and removal. Leveraging convolutional neural networks (CNNs) and classical denoising algorithms, we achieved accurate noise identification and effective noise suppression. Transfer learning enhanced classification accuracy, while deep learning-based approaches offered robust noise reduction capabilities. Evaluation metrics validated the efficacy of these techniques, underscoring their potential in diverse real-world applications. Future research could explore advanced architectures and real-time deployment, but overall, this project demonstrates the significance of addressing image noise for improved image analysis and interpretation.

6.2 Future Work

In addition to traditional denoising algorithms, deep learning-based approaches were explored for their ability to learn complex noise patterns and produce superior results. Specifically, convolutional neural networks (CNNs) were trained on noisy image data to predict clean image patches, effectively denoising the input images. Transfer learning, a technique that leverages pre-trained models to improve learning efficiency, was employed to fine-tune existing CNN architectures for noise removal in our specific domain.

References

[1] Jinzhu, L., Lijuan, T., Huanyu, J.(2021, July 27) "Review on Convolutional Neural Network (CNN) Applied to Plant Leaf Disease Classification"-Agriculture Journal.

[2] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.(2023, Jan 6). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998, 86 (11), pp.2278-2324. 10.1109/5.726791 . hal-03926082

[3] Scherer, D., Muller, A., Behnke, S.() Autonomous Intelligent Systems Group, R¨omerstr. 164, 53117 Bonn, Germany {scherer|amueller}@ais.uni-bonn.de, behnke@cs.uni-bonn.de http://www.ais.uni-bonn.de

[4] Jin, L., Song, E., & Zhang, W. (2020, January 9). Denoising Color Images Based on Local Orientation Estimation and CNN Classifier. *Journal of Mathematical Imaging and Vision*, *62*(4), 505–531. https://doi.org/10.1007/s10851-019-00942-8

[5] Lazcano-Herrera, A. G., Fuentes-Aguilar, R. Q., Ramirez-Morales, A., & Alfaro-Ponce, M. (2023). BiLSTM and SqueezeNet With Transfer Learning for EEG Motor Imagery Classification: Validation With Own Dataset. *IEEE Access*, *11*, 136422–136436.

[6] Hassanpour, M., Malek, H.(2018, Jan 4)Learning Document Image Features With SqueezeNet Convolutional Neural Network. (2020, July). *International Journal of Engineering*, *33*(7). https://doi.org/10.5829/ije.2020.33.07a.05

[7] FAROOQ, Y., & SAVAŞ, S. (2024, March 10). Noise Removal from the Image Using Convolutional Neural Networks-Based Denoising Auto Encoder. *Journal of Emerging Computer Technologies*, *3*(1), 21–28. https://doi.org/10.57020/ject.1390428

[8] Fabijańska, A., & Sankowski, D. (2011). Noise adaptive switching median-based filter for impulse noise removal from extremely corrupted images. *IET Image Processing*, *5*(5), 472. https://doi.org/10.1049/iet-ipr.2009.0178

[9] Cilimkovic, M. Neural Networks and Back Propagation Algorithm. Institute of Technology Blanchardstown Blanchardstown Road North https://www.academia.edu/download/51924347/NeuralNetworks.pdf

[10] Goceri, E. (2023). Evaluation of denoising techniques to remove speckle and Gaussian noise from dermoscopy images. Computers in Biology and Medicine, 152, 106474. https://doi.org/10.1016/j.compbiomed.2022.106474

[12] Kumar, N., & Nachamai, M. (2017). Noise Removal and Filtering Techniques Used in Medical Images. Oriental Journal of Computer Science and Technology, 10(1), 103–113. https://doi.org/10.13005/ojcst/10.01.14

[13] Attlas, N., & Gupta, D. S. (2012). Reduction of Speckle Noise in Ultrasound Images Using Wavelet Techniques and its Performance Analysis. Paripex - Indian Journal of Research, 3(5), 68–71. https://doi.org/10.15373/22501991/may2014/24

[14] Thakur, P. (2016). High-Density Salt and Pepper Noise Removal in Images Using Adapted Decision-Based Unsymmetrical Trimmed Mean Filter Cascaded with Gaussian Filter. International Journal of Engineering and Computer Science. https://doi.org/10.18535/ijecs/v4i10.39

[15] Villar, S.A., Torcida, S., Acosta, G.G.: Median filtering: a new insight. J. Math. Imaging Vis. 58(1), 130–146 (2017)

[16] Astola, J., Haavisto, P., Neuvo, Y.: Vector median filters. Proc. IEEE 78(4), 678–689 (1990)

[17] Trahanias, P.E., Venetsanopoulos, A.N.: Vector directional filters: a new class of multichannel image processing filters. IEEE Trans. Image Process. 2(4), 528–534 (1993)

[18] arakos, D.G., Trahanias, P.E.: Generalized multichannel image filtering structures. IEEE Trans. Image Process. 6(7), 1038–1045 (1997)

[19] Lukac, R.: Adaptive vector median filtering. Pattern Recogn. Lett. 24(12), 1889–1899 (2003)

[20] Smolka, B., Chydzinski, A.: Fast detection and impulsive noise removal in color images. Real Time Images 11(5/6), 389–402 (2005)

[21] Malinski, L., Smolka, B.: Fast averaging peer group filter for the impulsive noise removal in color images. J. Real Time Image Pro cess. 11(3), 427–444 (2016)

[22] Celebi, M.E., Aslandogan, Y.A.: Robust switching vector median filter for impulsive noise removal. J. Electron. Imaging 17(4), 043006 (2008)

[23] Shen, Y., Barner, K.E.: Fast adaptive optimization of weighted vector median filters. IEEE Trans. Signal Process. 54(7), 2497 2510 (2006)

[24] Lukac, R., Smolka, B., Plataniotis, K.N., Venetsanopoulos, A.N.: Selection weighted vector directional