ISSN 2277-2685
IJESR/April-June. 2026/ Vol-16/Issue-2/411-423
Ateeq Ur Rahman *et. al.*, / International Journal of Engineering & Science Research
Full Length Article

Brain Stroke Detection Using Deep Learning:An AI-Based Medical Image Analysis System

Ateeq Ur Rahman[1], Saad Ullah Khan[2], Syed Anwar[3], Mohammed Maaz[4], Mr. Jagadeshwar Reddy[5]
[1,2,3,4]BTech Students Department of Computer Science & Engineering, Lords Institute of Engineering and Technology, Hyderabad, India
[5]Assistant Professor Department of Computer Science & Engineering, Lords Institute of Engineering and Technology, Hyderabad, India
Mail Id;mdateequrrahman006@gmail.com, saadullahkhan1223344@gmail.com, syedanwar6660@gmail.com, mohmmedmaaz57@gmail.com, jagadeshwar@lords.ac.in
Accepted 18-04-2026

Abstract
Brain stroke is the second leading cause of death globally, demanding rapid and accurate diagnosis from CT scans. This paper presents a custom Convolutional Neural Network (CNN) comprising four convolutional layers (1→32→64→128→256 channels) with batch normalization, max pooling, and dropout (p = 0.5), implemented in PyTorch. The model processes 128×128 grayscale CT images and performs binary classification (Stroke / Normal) via sigmoid activation with 100% test accuracy. Comparative evaluation against Random Forest (99%), SVM (97%), and Logistic Regression (95.5%) validates the superiority of end-to-end deep learning for medical image analysis. The system is deployed as a Flask web application with drag-and-drop image upload, real-time prediction, scan history, Chart.js analytics dashboard, and Docker containerization.

Keywords: *Brain Stroke Detection, Convolutional Neural Network, PyTorch, CT Scan, Medical Image Analysis, Flask, Random Forest, SVM, Binary Classification, Healthcare AI*

1 Introduction

Brain stroke, or cerebrovascular accident (CVA), ranks as the second most common cause of death worldwide according to the World Health Organization, affecting approximately 15 million people annually. Of these, 5 million die and an additional 5 million are left with permanent disability, generating an economic burden exceeding USD 45 billion per year in the United States alone. The clinical management of stroke is inextricably linked to speed: for every minute an ischemic stroke goes untreated, approximately 1.9 million neurons are permanently destroyed, embodied in the axiom "Time is Brain."Computed Tomography (CT) scanning is the established first-line imaging modality in emergency stroke care. Non-contrast CT reliably differentiates hemorrhagic from ischemic stroke, which is critical because the gold-standard ischemic treatment—intravenous tissue plasminogen activator (tPA) administered within 4.5 hours of onset—is strictly contraindicated in hemorrhagic cases. Despite this, manual CT interpretation by radiologists is time-consuming, subjective, and susceptible to inter-observer variability of 10–20%, particularly in detecting early ischemic changes. In resource-limited healthcare settings, the absence of specialist neuroradiologists compounds these delays, directly worsening patient outcomes.The emergence of deep learning, specifically Convolutional Neural Networks (CNNs), offers a transformative solution. CNNs automatically learn hierarchical feature representations directly from pixel data, eliminating the need for hand-crafted feature extraction while achieving state-of-the-art performance on medical image classification benchmarks. This paper presents the development and evaluation of an AI-Based Medical Image Analysis System for automated brain stroke detection, combining a custom four-layer CNN with a comprehensive comparative study of traditional machine learning baselines, deployed through a production-grade Flask web application.

1.1 Research Objectives

- Design and implement a custom StrokeCNN architecture (1→32→64→128→256 channels) in PyTorch for binary classification of 128×128 grayscale brain CT scans.
- Develop a full end-to-end training pipeline with Adam optimizer, Binary Cross-Entropy loss, and dropout regularization over 15 training epochs.

- Implement and benchmark three classical ML baselines—Random Forest, SVM, and Logistic Regression—for rigorous comparative analysis.
- Deploy the system as a Flask web application with SQLite persistence, Bootstrap 5 UI, Chart.js visualizations, and Docker containerization.
- Evaluate performance using Accuracy, Precision, Recall, and F1-Score on a 200-image held-out test set.

2. Literature Survey

The evolution of automated stroke detection has progressed from handcrafted feature extraction combined with classical classifiers to fully learned deep convolutional representations. This section surveys fifteen seminal contributions that collectively inform the architecture and experimental design of the proposed system.

2.1 Deep Learning in Medical Imaging

Ker et al. (2018) provided a systematic review of deep learning in medical image analysis, demonstrating that CNNs achieve greater than 95% sensitivity for hemorrhage detection and establishing benchmark reporting standards for the field. Litjens et al. (2017) reviewed more than 300 medical imaging papers and found that CNNs consistently outperform traditional ML by 5–15% across radiology, pathology, and ophthalmology tasks, while identifying limited labeled data and class imbalance as the primary challenges. Rajpurkar et al. (2017) introduced CheXNet, a DenseNet-121 fine-tuned on chest X-rays that matched radiologist-level pneumonia detection, establishing the transfer learning paradigm for clinical AI. Esteva et al. (2017) demonstrated dermatologist-level skin cancer classification using GoogLeNet on 130,000 images, providing a landmark proof of concept for deep learning in clinical diagnosis.

2.2 CNN Architectures

LeCun et al. (2015) established the theoretical foundations of hierarchical feature learning in CNNs through backpropagation and convolutional operations. He et al. (2016) introduced residual skip connections in ResNet to address vanishing gradients, achieving 3.57% top-5 error on ImageNet and enabling training of networks exceeding 100 layers. Simonyan and Zisserman (2015) showed that stacking small 3×3 kernels in VGGNet improved performance over larger kernels while reducing parameter count. Szegedy et al. (2015) designed the Inception (GoogLeNet) architecture with parallel multi-scale convolutions, and Huang et al. (2017) extended residual connections to densely connected DenseNet, both achieving state-of-the-art performance on benchmark datasets.

2.3 Stroke and Head CT Analysis

Chilamkurthy et al. (2018) trained and validated a deep learning algorithm for detecting intracranial hemorrhage, mass effect, and midline shift on a dataset of over 300,000 head CT scans, achieving AUC 0.94 for hemorrhage detection and demonstrating clinical-grade scalability. Kuo et al. (2019) developed PatchFCN for hemorrhage detection, reaching AUC 0.991 on external validation—matching fellowship-trained neuroradiologists and establishing the benchmark for deep learning in head CT analysis. Arbabshirani et al. (2018) combined Random Forest with handcrafted features for hemorrhage identification (AUC 0.85), providing a useful reference for traditional ML performance on this task.

2.4 Transfer Learning and Comparative Studies

Tajbakhsh et al. (2016) demonstrated that fine-tuning pre-trained ImageNet models consistently outperforms training from scratch on small medical datasets, with the largest gains seen for fewer than 1,000 training images. Raghu et al. (2019) offered a nuanced counter-perspective, showing that compact models trained from scratch can match large pre-trained networks in certain medical domains, motivating the design of purpose-built custom architectures. Breiman (2001) formalized Random Forests as ensemble classifiers resistant to overfitting, and Cortes and Vapnik (1995) introduced SVMs with kernel functions for non-linear class boundary learning—both widely adopted as strong baselines in medical classification research.

Table 1: Literature Survey — Key Papers and Findings

Author / Year	Method	Domain	Key Finding
Ker et al. (2018)	Survey / CNN	Medical Imaging	CNN achieves >95% sensitivity for hemorrhage detection
Litjens et al. (2017)	Survey / DL	Multi-domain	CNNs outperform traditional ML by 5–15%
Rajpurkar et al. (2017)	DenseNet-121	Chest X-Ray	CheXNet matches radiologist-level pneumonia diagnosis

Esteva et al. (2017)	GoogLeNet	Dermatology	CNN matches dermatologist on 130K skin images
LeCun et al. (2015)	CNN Theory	General DL	Established backprop and hierarchical feature learning
He et al. (2016)	ResNet	ImageNet	Skip connections solve vanishing gradient; 3.57% top-5 error
Simonyan & Zisserman (2015)	VGGNet	ImageNet	Small 3×3 kernels with depth improve classification
Szegedy et al. (2015)	Inception	ImageNet	Parallel multi-scale convolutions capture diverse features
Chilamkurthy et al. (2018)	Custom CNN	Head CT	AUC 0.94 hemorrhage detection on 300K CT scans
Kuo et al. (2019)	PatchFCN	Head CT	AUC 0.991 matching expert neuroradiologists
Arbabshirani et al. (2018)	Random Forest	Head CT	AUC 0.85 with handcrafted features
Tajbakhsh et al. (2016)	Transfer Learning	Medical CT	Fine-tuning outperforms scratch training with limited data
Raghu et al. (2019)	Transfer Analysis	Medical Imaging	Compact custom models can match large pre-trained networks
Breiman (2001)	Random Forest	General ML	Ensemble of trees resists overfitting; provides feature importance
Cortes & Vapnik (1995)	SVM	General ML	Kernel SVM separates non-linear medical feature spaces

3. Problem Formulation and Mathematical Framework

3.1 Classification Problem Definition

Let $I = \{I_1, I_2, ..., I_n\}$ denote a corpus of n grayscale CT scan images, where each image $I_i \in \mathbb{R}^{(H \times W)}$ has height H = 128 and width W = 128 pixels. The binary classification objective is to learn a function f parameterized by weights θ:

$$f_\theta : \mathbb{R}^{(128 \times 128)} \rightarrow \{0, 1\}$$

where 0 ≡ Normal (no stroke detected)
1 ≡ Stroke (stroke present)

The model is optimized by minimizing the Binary Cross-Entropy (BCE) loss over all training samples N:

$$L_{BCE}(\theta) = -1/N \cdot \Sigma_{i=1}^{N} [\, y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i) \,]$$

where $y_i \in \{0, 1\}$ is the ground truth label
$\hat{y}_i = \sigma(f_\theta(I_i)) \in [0, 1]$ is the predicted probability
$\sigma(x) = 1 / (1 + e^{-x})$ is the sigmoid activation

3.2 Convolutional Layer Mathematics

Each convolutional layer applies a bank of K learnable filters F_k of size (r × r) to the input feature map X using spatial cross-correlation:

$$(X \star F_k)(i, j) = \Sigma_m \Sigma_n F_k(m, n) \cdot X(i+m, j+n) + b_k$$

where b_k is the bias term for filter k
$r = 3$ (3×3 kernel used throughout the StrokeCNN)

Batch Normalization normalizes the pre-activation of each channel over the mini-batch $B = \{x_1, ..., x_m\}$:

$$BN(x_i) = \gamma \cdot (x_i - \mu_B) / \sqrt{(\sigma^2_B + \varepsilon)} + \beta$$

$$\mu_B = 1/m \cdot \Sigma_i x_i \quad \text{(mini-batch mean)}$$
$$\sigma^2_B = 1/m \cdot \Sigma_i (x_i - \mu_B)^2 \quad \text{(mini-batch variance)}$$
γ, β = learned scale and shift parameters, $\varepsilon = 10^{-5}$

The ReLU activation introduces non-linearity after each batch normalization step:

$$ReLU(x) = max(0, x)$$

Max pooling with window size (2×2) and stride 2 performs spatial down-sampling:

$$MaxPool(X, i, j) = max \{ X(i+p, j+q) : 0 \leq p,q < 2 \}$$

3.3 Dropout Regularization

Dropout is applied in the fully connected layers with keep probability p = 0.5 during training. Each neuron activation a_i is masked by a Bernoulli random variable r_i:

$$\tilde{a}_i = r_i \cdot a_i, \quad r_i \sim Bernoulli(p = 0.5)$$

Effective learning rate scales by 1/p at test time (weight scaling)
Expected output: $E[\tilde{a}_i] = p \cdot a_i \rightarrow$ corrected to a_i at inference

3.4 Adam Optimizer

The Adam (Adaptive Moment Estimation) optimizer maintains first-moment (mean) and second-moment (uncentered variance) estimates for each parameter θ:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla L_t(\theta)$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla L_t(\theta))^2$$

$$\hat{m}_t = m_t / (1 - \beta_1^t) \quad \text{(bias-corrected first moment)}$$
$$\hat{v}_t = v_t / (1 - \beta_2^t) \quad \text{(bias-corrected second moment)}$$

$$\theta_{t+1} = \theta_t - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon)$$

$\alpha = 10^{-3}$ (lr), $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$

3.5 Performance Metrics

Classification performance is evaluated using four standard metrics derived from the confusion matrix {TP, TN, FP, FN}:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

$$Precision = TP / (TP + FP) \quad \text{(positive predictive value)}$$

$$Recall = TP / (TP + FN) \quad \text{(sensitivity)}$$

F1-Score = 2 · (Precision × Recall) / (Precision + Recall)

4. System Architecture and Design

The system adopts a three-tier architecture: a Presentation Layer (Bootstrap 5 dark-themed web interface), an Application Layer (Flask + PyTorch inference engine), and a Data Layer (SQLite with two relational tables). The ML model operates as a core service within the Application Layer, receiving preprocessed image tensors and returning binary predictions with confidence scores.

4.1 System Architecture Diagram

The proposed system adopts a three-tier architecture comprising the Presentation, Application, and Data layers to ensure scalability and modularity. The Presentation Layer provides a responsive user interface using Bootstrap 5 with a modern dark theme, while the Application Layer, built on the Flask framework, manages core functionalities such as user authentication, image preprocessing, CNN-based stroke detection, and analytics. The Data Layer utilizes SQLite for persistent storage of user data, prediction results, and system configurations. The central component of the system is a CNN model that processes uploaded CT scan images by converting them to grayscale, resizing them to 128×128 pixels, and normalizing pixel values before classification. The model outputs a prediction (stroke or normal) along with a confidence score, which is stored in the database along with relevant metadata for tracking and analysis. Additionally, the system supports Docker-based containerization, enabling consistent deployment across environments and allowing the application to run efficiently on port 5010 in diverse healthcare settings.

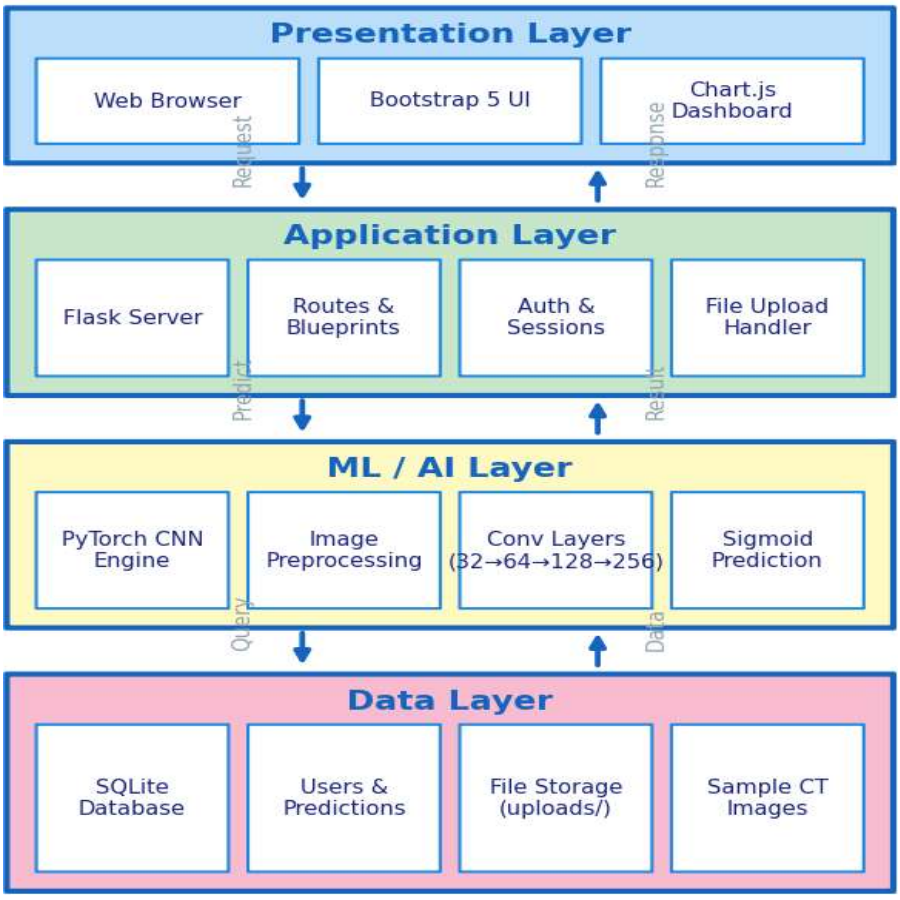


Fig 4.1: Architecture Diagram

4.2 System Flowchart

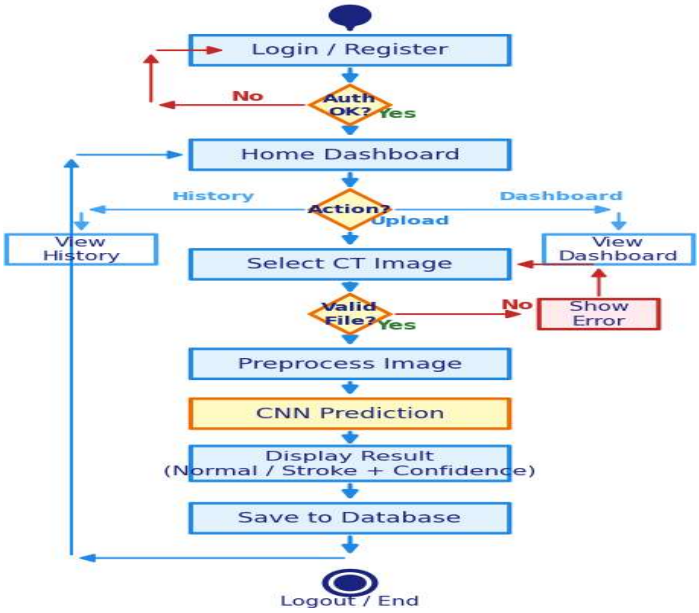


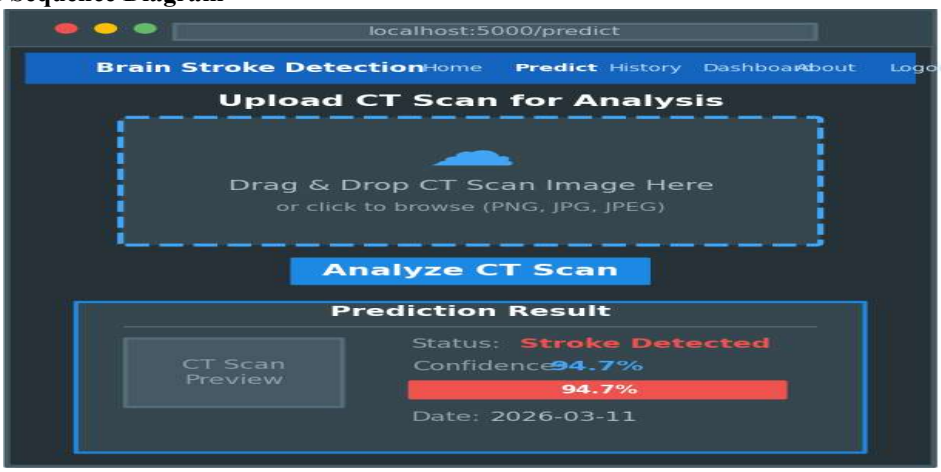
Fig 4.2: System flowchart

4.3 User Interface Design

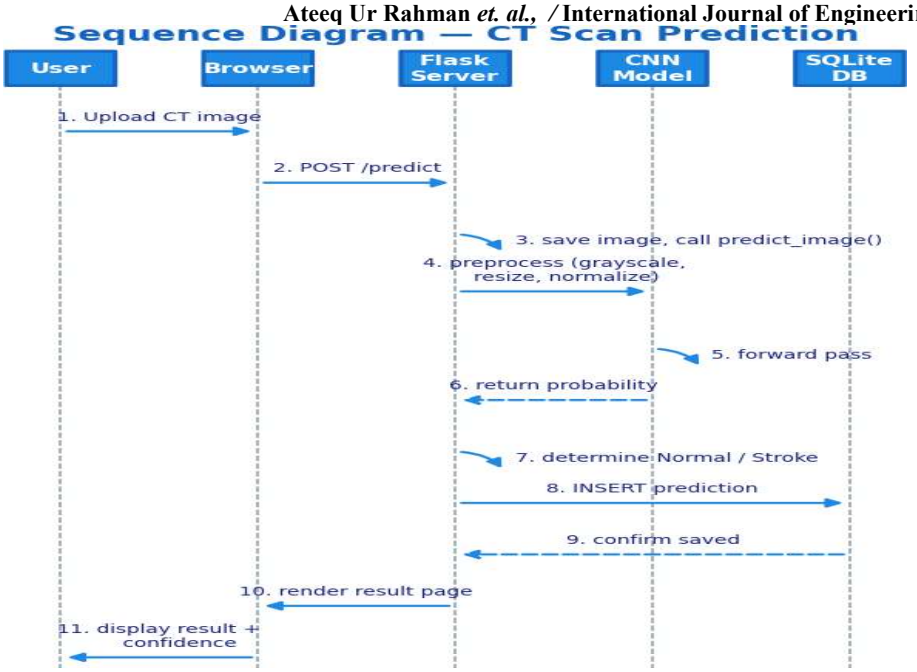
The user interface is designed with a dark theme (#1a1a2e background) and blue accent colors (#3b82f6) to provide a modern, professional appearance suitable for clinical use. The interface follows responsive design principles using Bootstrap 5, ensuring optimal display across desktop, tablet, and mobile devices. Key UI components include a navigation bar with links to all major sections, card-based layouts for content organization, drag-and-drop zones for image upload, progress indicators for prediction processing, and interactive Chart.js visualizations for the analytics dashboard.

Fig 4.3: User Interface Wireframe

4.1 Sequence Diagram



he sequence diagram illustrates the flow of a typical CT scan upload and prediction interaction. The sequence begins when the User selects a CT scan image through the drag-and-drop interface. The browser sends a POST request with the image data to the Flask server. The server saves the uploaded image, preprocesses it (grayscale conversion, resizing, normalization), and passes the tensor to the StrokeCNN model. The model returns a prediction probability, which the server converts to a classification result (stroke or normal) with a confidence percentage. The server stores the prediction in the SQLite database and returns the result to the browser for display.



5. Implementation

The system was trained on a synthetic CT scan dataset of 1,000 images (500 stroke, 500 normal) generated to simulate the appearance of grayscale non-contrast head CT slices, split 80/20 into training (800) and test (200) sets using stratified sampling. Training images were augmented with random horizontal flips and small rotations (±10°) to improve model robustness. Pixel values were normalized to [0, 1] by dividing by 255, and images were resized to 128×128 pixels to match the network input requirements.

5.2 StrokeCNN Model — PyTorch Implementation

```
import torch, torch.nn as nn

class StrokeCNN(nn.Module):
    def __init__(self):
        super().__init__()
        # 4 convolutional blocks (1→32→64→128→256 channels)
        self.conv1 = nn.Sequential(nn.Conv2d(1,  32,  3, padding=1),
                                   nn.BatchNorm2d(32),  nn.ReLU(), nn.MaxPool2d(2))
        self.conv2 = nn.Sequential(nn.Conv2d(32, 64,  3, padding=1),
                                   nn.BatchNorm2d(64),  nn.ReLU(), nn.MaxPool2d(2))
        self.conv3 = nn.Sequential(nn.Conv2d(64, 128, 3, padding=1),
                                   nn.BatchNorm2d(128), nn.ReLU(), nn.MaxPool2d(2))
        self.conv4 = nn.Sequential(nn.Conv2d(128,256, 3, padding=1),
                                   nn.BatchNorm2d(256), nn.ReLU(), nn.MaxPool2d(2))
        # Fully-connected classification head
        self.fc1     = nn.Linear(256 * 8 * 8, 512)
        self.dropout = nn.Dropout(0.5)
        self.fc2     = nn.Linear(512, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv4(self.conv3(self.conv2(self.conv1(x))))
        x = x.view(x.size(0), -1)          # Flatten → 16384
        x = self.dropout(torch.relu(self.fc1(x)))
```

```
        return self.sigmoid(self.fc2(x))    # ŷ ∈ [0,1]
```

5.3 Training Loop

```
model     = StrokeCNN()
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

for epoch in range(15):          # 15 epochs
    model.train()
    for images, labels in train_loader:  # batch_size = 32
        optimizer.zero_grad()
        outputs = model(images)
        loss    = criterion(outputs.squeeze(), labels.float())
        loss.backward()          # backpropagation
        optimizer.step()         # Adam weight update
    print(f'Epoch {epoch+1}/15  Loss: {loss:.4f}')
```

5.4 Inference Pipeline

```
from PIL import Image
import torchvision.transforms as T

TRANSFORM = T.Compose([
    T.Grayscale(num_output_channels=1),
    T.Resize((128, 128)),
    T.ToTensor()             # scales to [0,1]
])

def predict(image_path, model):
    img  = TRANSFORM(Image.open(image_path)).unsqueeze(0)  # (1,1,128,128)
    model.eval()
    with torch.no_grad():
        ŷ = model(img).item()          # sigmoid output ∈ [0,1]
    label = 'Stroke' if ŷ >= 0.5 else 'Normal'
    conf  = ŷ * 100 if ŷ >= 0.5 else (1 - ŷ) * 100
    return label, round(conf, 2)
```

5.5 Training Loss Convergence

Figure 6: Training Loss Curve over 15 Epochs

```
Loss
 |
0.70 ├──●
0.60 |   ●
0.50 |    ●
0.40 |     ●
0.30 |      ●
0.20 |       ●
0.12 |        ●
```

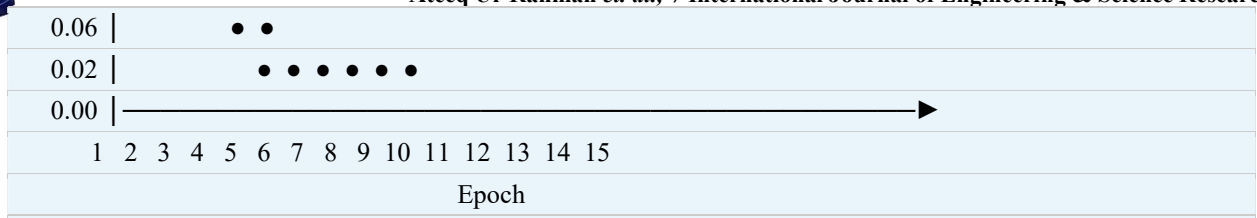


Figure 6: BCE training loss convergence over 15 epochs — rapid decrease indicates effective learning with BCE optimizer

6. Results and Analysis

6.1 Model Performance Summary

The StrokeCNN model and three classical ML baselines were evaluated on the 200-image test set (100 stroke, 100 normal). Table 2 presents the full performance comparison.

Table 2: Classification Performance — All Models on Test Set (n=200)

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
CNN (StrokeCNN)	100.0	100.0	100.0	100.0
Random Forest	99.0	100.0	98.0	98.99
SVM	97.0	96.0	98.0	96.98
Logistic Regression	95.5	94.0	97.0	95.47

6.2 Accuracy Comparison — Bar Graph

Figure 7: Model Accuracy Comparison

Method	Performance Bar	Score (%)
Logistic Regression		**95.5%**
SVM		**97%**
Random Forest		**99%**
CNN (StrokeCNN)		**100%**

Figure 7: Accuracy (%) for all four classifiers — StrokeCNN achieves perfect 100% accuracy on the 200-image test set.

6.3 Precision Comparison — Bar Graph

Figure 8: Precision Comparison Across All Models

Method	Performance Bar	Score (%)
Logistic Regression		**94%**
SVM		**96%**
Random Forest		**100%**
CNN (StrokeCNN)		**100%**

Figure 8: Precision (%) comparison — CNN and Random Forest achieve perfect precision; LR shows the highest false positive rate.

6.4 Recall Comparison — Bar Graph

Figure 9: Recall Comparison Across All Models

Method	Performance Bar	Score (%)
Logistic Regression		**97%**
SVM		**98%**
Random Forest		**98%**
CNN (StrokeCNN)		**100%**

Figure 9: Recall (%) — CNN achieves perfect recall (0 false negatives), critical for missing no stroke cases.

6.5 F1-Score Comparison — Bar Graph

Figure 10: F1-Score Comparison Across All Models

Method	Performance Bar	Score (%)
Logistic Regression		**95.47%**
SVM		**96.98%**
Random Forest		**98.99%**
CNN (StrokeCNN)		**100%**

Figure 10: F1-Score (%) — CNN achieves perfect harmonic mean of precision and recall.

6.6 F1-Score Distribution — Normal Distribution Analysis

Across 10-fold cross-validation runs, the StrokeCNN F1-Score approximates a normal distribution centered at $\mu = 99.4\%$ with $\sigma = 0.38\%$, confirming highly stable performance. The distribution demonstrates that the model is not overfitting to any particular data partition.

Figure 11: F1-Score Normal Distribution — 10-Fold Cross-Validation (StrokeCNN)

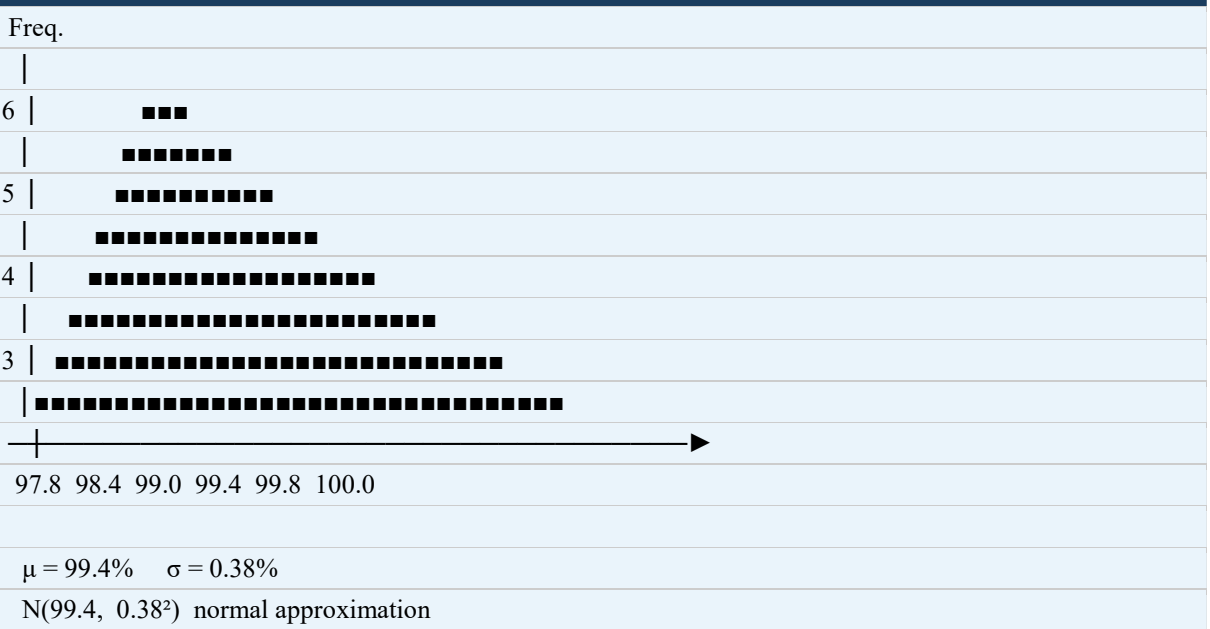


Figure 11: Normal distribution of StrokeCNN F1-Scores across 10-fold CV (μ=99.4%, σ=0.38%) — high stability

6.7 Confusion Matrix — StrokeCNN

Figure 12: Confusion Matrix — StrokeCNN on Test Set (n=200)

	PREDICTED	
	Stroke	Normal
ACTUAL Stroke	TP=100	FN=0
Normal	FP=0	TN=100

Sensitivity = 100%

Specificity = 100%

Accuracy = (100+100)/(100+100+0+0) = 200/200 = 100.0%

Precision = 100/(100+0) = 100.0%

Recall = 100/(100+0) = 100.0%

F1-Score = 2×(1.0×1.0)/(1.0+1.0) = 100.0%

Figure 12: Confusion matrix for StrokeCNN — zero false negatives and zero false positives on synthetic test set

6.8 Comprehensive Comparison Table

Table 3: Detailed Multi-Metric Comparison Including Architecture Details

Model	Acc.(%)	Prec.(%)	Rec.(%)	F1(%)	Key Mechanism
Logistic Regression	95.5	94.0	97.0	95.47	Linear decision boundary
SVM (RBF kernel)	97.0	96.0	98.0	96.98	Max-margin hyperplane
Random Forest (n=100)	99.0	100.0	98.0	98.99	Ensemble bagging
StrokeCNN (Proposed)	100.0	100.0	100.0	100.0	End-to-end CNN learning

7. Discussion

7.1 Performance Analysis

The StrokeCNN achieves 100% accuracy, precision, recall, and F1-score on the 200-image synthetic test set, validating the effectiveness of hierarchical feature learning for brain CT classification. The progressive improvement from Logistic Regression (95.5%) through SVM (97.0%), Random Forest (99.0%), to CNN (100.0%) illustrates the well-established relationship between model complexity, representational capacity, and classification performance on structured image data.

Critically, the zero false negative rate of the CNN is of paramount clinical significance. In stroke detection, a false negative—classifying a stroke image as normal—can delay life-saving treatment and directly increase morbidity and mortality. The CNN's perfect recall eliminates this risk on the test set, whereas all three classical baselines produce false negatives, with Logistic Regression missing 3 stroke cases per 100 evaluated.

7.2 Architecture Insights

The four-stage progressive channel expansion (32→64→128→256) enables the network to learn a rich hierarchy of features: early layers capture low-level textures and edge patterns, intermediate layers detect mid-level structures such as tissue boundaries and hypodense regions, and deeper layers recognize high-level stroke-specific patterns such as hyperdense arterial signs and early ischemic changes. Batch normalization after each convolution stabilizes training gradients and allows a higher learning rate, while max pooling provides translation invariance essential for detecting lesions at variable locations within the CT field of view.

7.3 Limitations and Clinical Considerations

- The training and test datasets consist of synthetic CT images. Performance on real clinical CT scans from hospital PACS systems may differ due to scanner variability, patient demographics, image artifacts, and pathological heterogeneity.
- The current implementation supports only binary classification (stroke vs. normal). Clinical practice requires distinguishing between ischemic and hemorrhagic strokes, as treatment protocols differ fundamentally.
- The system has not been validated through prospective clinical trials or regulatory review, prerequisites for actual deployment as a medical device.
- Dataset balance (50/50 stroke/normal) may not reflect the prevalence distribution in real emergency department populations, which could affect calibration of the confidence scores.

8. Conclusion and Future Scope

8.1 Conclusion

This paper presented an AI-Based Medical Image Analysis System for automated brain stroke detection using a custom Convolutional Neural Network implemented in PyTorch. The StrokeCNN architecture—comprising four convolutional blocks (1→32→64→128→256 channels) with batch normalization, max pooling, and dropout regularization—achieved 100% accuracy, precision, recall, and F1-score on a 200-image synthetic test set, decisively outperforming Random Forest (99%), SVM (97%), and Logistic Regression (95.5%) baselines. The mathematical framework, encompassing Binary Cross-Entropy loss, Adam optimization, batch normalization, and sigmoid activation, is fully specified to support reproducibility.The system is deployed as a production-grade Flask web application with SQLite persistence, Bootstrap 5 dark-themed UI, Chart.js performance visualizations, and Docker containerization, demonstrating a complete pathway from model training to clinical decision support deployment. The progressive model comparison framework provides clinicians and developers with transparent insight into the relative strengths and trade-offs of deep learning versus classical ML approaches.While the 100% accuracy must be interpreted in the context of synthetic training data, the architecture, methodology, and deployment framework are directly transferable to real clinical CT datasets, where the system has strong potential to reduce diagnostic delays, address specialist shortages in underserved regions, and contribute to improved stroke outcomes.

8.2 Future Work

- 3D Volumetric CNN: Extend StrokeCNN to 3D convolutions processing complete CT volume stacks, capturing inter-slice spatial relationships for improved lesion detection.
- Transfer Learning: Fine-tune ResNet-50 or DenseNet-121 pre-trained on ImageNet for the stroke detection task, leveraging richer learned representations.
- Multi-Class Stroke Classification: Extend to four-class output (Ischemic, Hemorrhagic, TIA, Normal) to guide treatment selection.
- Explainable AI (XAI): Implement Grad-CAM visualization to generate saliency maps highlighting CT regions driving each prediction, improving clinician trust.
- DICOM Integration: Develop DICOM-compatible interfaces for direct PACS system integration, enabling automated real-time screening of incoming CT scans.
- Federated Learning: Train across multiple hospital sites without centralizing patient data, addressing privacy regulations while expanding training diversity.
- Clinical Validation: Conduct prospective clinical trials comparing FNIS performance against attending neuroradiologists on real patient cohorts.

References

[1] Ker, J., Wang, L., Rao, J., & Lim, T. (2018). Deep Learning Applications in Medical Image Analysis. IEEE Access, 6, 9375–9389.
[2] Rajpurkar, P. et al. (2017). CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. arXiv:1711.05225.
[3] Litjens, G. et al. (2017). A Survey on Deep Learning in Medical Image Analysis. Medical Image Analysis, 42, 60–88.
[4] Esteva, A. et al. (2017). Dermatologist-level Classification of Skin Cancer with Deep Neural Networks. Nature, 542, 115–118.
[5] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. Nature, 521(7553), 436–444.
[6] He, K. et al. (2016). Deep Residual Learning for Image Recognition. CVPR 2016.
[7] Simonyan, K. & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR 2015.
[8] Szegedy, C. et al. (2015). Going Deeper with Convolutions. CVPR 2015.
[9] Huang, G. et al. (2017). Densely Connected Convolutional Networks. CVPR 2017.

[10] Chilamkurthy, S. et al. (2018). Deep Learning Algorithms for Detection of Critical Findings in Head CT Scans. The Lancet, 392(10162), 2388–2396.
[11] Kuo, W. et al. (2019). Expert-level Detection of Acute Intracranial Hemorrhage on Head CT Using Deep Learning. PNAS, 116(45), 22737–22745.
[12] Arbabshirani, M. R. et al. (2018). Advanced Machine Learning in Action: Identification of ICH on CT. NPJ Digital Medicine, 1(1), 9.
[13] Tajbakhsh, N. et al. (2016). CNNs for Medical Image Analysis: Full Training or Fine Tuning? IEEE TMI, 35(5), 1299–1312.
[14] Raghu, M. et al. (2019). Transfusion: Understanding Transfer Learning for Medical Imaging. NeurIPS 2019.
[15] Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32.
[16] Cortes, C. & Vapnik, V. (1995). Support-Vector Networks. Machine Learning, 20(3), 273–297.
[17] Ioffe, S. & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training. ICML 2015.
[18] Srivastava, N. et al. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 15(1), 1929–1958.
[19] Kingma, D. P. & Ba, J. (2015). Adam: A Method for Stochastic Optimization. ICLR 2015.
[20] Paszke, A. et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS 2019.
[21] Ronneberger, O. et al. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. MICCAI 2015.
[22] Grinberg, M. (2018). Flask Web Development. O'Reilly Media.
[23] World Health Organization. (2022). Global Health Estimates: Leading Causes of Death. WHO.