School of Engineering and Applied Sciences

State University of New York, Buffalo

Assignment – HW1

[Deep Learning for Finger Gesture Recognition]

Authored by

Sachin Saailesh Jeyakkumaran
Person Number: 50605636
UBIT Name: sjeyakku

Course

EAS595: Deep Learning in Robotic Applications

Submission Date

October 3, 2025

Contents

Abstract

This project explores the development of a finger gesture recognition system using fully connected neural networks to classify hand gestures representing numbers from 0 to 5. Six distinct network configurations were implemented, each incorporating different combinations of activation functions, weight initialization methods, and regularization strategies. The Kaggle Finger Gestures dataset served as the foundation, from which a custom 10% training subset comprising 21,600 images was extracted. Training extended over 100 epochs with systematic checkpoint preservation every 2 epochs to facilitate convergence analysis. Experimental results indicate that the configuration utilizing Tanh activation, Xavier initialization, and Dropout regularization achieved superior performance with 40% accuracy on independently captured test images. Analysis of training dynamics revealed tendencies toward underfitting across most configurations, suggesting opportunities for architectural enhancement. This work provides practical insights into designing gesture recognition systems for robotic interaction interfaces.

1 Introduction

The advancement of human-robot interaction technologies has created increasing demand for intuitive communication interfaces that enable natural collaboration between humans and autonomous systems. Gesture-based control represents a promising modality for such interaction, offering contactless operation that aligns with innate human communication patterns. Among various gesture vocabularies, finger counting provides a universally understood symbolic system suitable for conveying numerical information and discrete commands to robotic platforms.

The recognition of finger counting gestures presents an engaging challenge that bridges computer vision, machine learning, and robotic perception. Unlike arbitrary hand poses, finger counting exhibits structured variation where each gesture class corresponds to a specific number of extended fingers. This structure makes the task accessible for systematic study while retaining sufficient complexity to require sophisticated learning algorithms. Applications span diverse domains including collaborative manufacturing where workers signal instructions to robotic arms, assistive robotics where elderly users communicate through simple gestures, and surgical environments where touchless interfaces maintain sterility.

This project investigates the application of fully connected neural networks to finger gesture recognition, examining how architectural choices influence learning dynamics and generalization capability. The work focuses on three primary research questions. First, how do different activation functions affect network training and convergence behavior? Second, what impact do regularization strategies have on model generalization to novel images? Third, can weight initialization schemes meaningfully influence final performance in networks of moderate depth?

1.1 Objectives

The primary objectives guiding this investigation are structured to provide comprehensive understanding of neural network behavior in visual recognition tasks:

1. Develop a systematic data preparation pipeline that samples and organizes the Kaggle Finger Gestures dataset according to specified constraints, ensuring reproducibility through proper random seeding and label extraction procedures.

2. Implement six fully connected neural network architectures with controlled variation in activation functions (Sigmoid, Tanh, ReLU), initialization schemes (Xavier, He), and regularization approaches (None, L2, Dropout).

3. Execute comprehensive training sessions for each configuration over 100 epochs, maintaining consistent hyperparameters across experiments and preserving model checkpoints at regular intervals.

4. Conduct detailed convergence analysis by evaluating all checkpoints to characterize learning trajectories, identify optimal stopping points, and assess overfitting or underfitting tendencies.

5. Evaluate trained models on independently captured hand gesture images to measure real-world generalization and identify configuration-specific strengths and weaknesses.

6. Synthesize experimental findings to recommend optimal architectural choices for gesture recognition in robotic applications.

These objectives collectively address both theoretical understanding of neural network training dynamics and practical considerations for deploying gesture recognition systems in robotic platforms.

2 Task Description

The core task involves developing a machine learning system capable of accurately classifying grayscale images of hands into six categories based on the number of extended fingers visible. Each category corresponds to an integer label from 0 to 5, representing the count of visible fingers. The system must process 128×128 pixel grayscale images and output predictions that match the ground truth finger counts.

The task formulation treats gesture recognition as a regression problem rather than traditional multi-class classification. The network architecture employs a single output neuron with linear activation, predicting continuous values that are subsequently rounded to the nearest integer within the valid range. This approach offers certain advantages including smoother optimization landscapes and naturally ordered predictions that respect the ordinal relationship between finger counts.

Input images undergo preprocessing to ensure compatibility with the neural network architecture. Raw images are loaded in grayscale mode, resized to uniform 128×128 resolution using high-quality interpolation methods, and normalized to floating-point values in the range [0, 1] through division by the maximum pixel intensity. The two-dimensional image arrays are then flattened into one-dimensional vectors containing 16,384 elements, matching the input layer dimensionality requirements of the fully connected network.

The dataset presents several inherent challenges that influence model design and training strategies. Intra-class variability arises from differences in hand size, finger thickness, and pose orientation across individuals and capture sessions. Background variation, while limited in the training set, becomes significant when evaluating on independently captured images. Lighting conditions affect contrast and shadow patterns, potentially confounding feature extraction. These challenges motivate careful consideration of regularization approaches and data preprocessing techniques.

Success in this task requires the network to learn discriminative features that capture hand shape and finger configuration while remaining robust to irrelevant variations in background, lighting, and hand positioning. The limited training set size (10% of available data) further constrains the learning problem, requiring efficient use of available samples to develop generalizable representations.

3 Experimental Setup

3.1 Dataset Preparation

The experimental work utilizes the Kaggle Finger Gestures dataset, a curated collection of hand gesture images specifically designed for recognition research. The original dataset divides images into separate training and testing partitions, with each image filename encoding both the finger count label and hand laterality through a standardized naming convention. Labels range from 0 to 5 corresponding to the number of extended fingers, while laterality indicators specify left or right hand gestures.

Dataset preparation began by merging both training and testing partitions into a unified collection, eliminating any artificial bias that might result from selecting exclusively from one partition. A regular expression parser extracted numeric labels from filenames by identifying the digit immediately preceding the hand laterality indicator. Images failing to match the expected naming pattern were excluded to ensure label accuracy.

From the combined image pool, a random 10% subset was selected following assignment specifications. Random selection employed a fixed seed value to guarantee reproducibility across multiple execution runs. This sampling strategy yielded approximately 21,600 images distributed across the six gesture classes. The selected images were reorganized into label-specific directories, creating a structured hierarchy that simplifies subsequent data loading operations.

Statistical analysis of the prepared dataset confirmed near-uniform class distribution with each label represented by approximately 3,600 images. Minor variations between class counts (maximum deviation under 2%) result from random sampling applied to an approximately balanced source dataset and are considered negligible. This balanced distribution ensures that network training does not develop bias toward over-represented classes.

The prepared dataset underwent an additional internal split during training, allocating 80% of images to the training set and reserving 20% for validation monitoring. This split provides approximately 17,280 images for gradient-based optimization and 4,320 images for assessing generalization during training. The validation partition serves as an early indicator of overfitting, enabling identification of epochs where validation performance degrades despite continued training improvements.

3.2 Network Architecture

All experimental sessions employed an identical fully connected neural network architecture to ensure that observed performance differences could be attributed to variation in activation functions, initialization, and regularization rather than structural differences. The architecture follows a progressive dimensionality reduction pattern, compressing the high-dimensional input space through successive hidden layers before producing a single scalar output.

The input layer accepts flattened image vectors containing 16,384 features corresponding to 128×128 pixel images. This high-dimensional input space requires the network to identify relevant patterns from extensive raw pixel data without benefit of explicit spatial feature engineering. The first hidden layer comprises 256 neurons, providing substantial representational capacity for learning diverse feature detectors responsive to various pixel intensity patterns.

The second hidden layer reduces dimensionality to 128 neurons, forcing the network to develop more compressed and abstract representations. The third hidden layer further restricts representation to 64 neurons, creating an information bottleneck that encourages learning of compact, discriminative features. Finally, the output layer consists of a single neuron with linear activation, producing continuous predictions subsequently rounded to integer labels.

This bottleneck architecture bears conceptual similarity to autoencoder designs, though applied in a supervised rather than unsupervised learning context. The progressive compression through layers ($16384 \to 256 \to 128 \to 64 \to 1$) encourages hierarchical feature learning where early layers capture low-level patterns and deeper layers synthesize these into higher-level representations relevant for finger counting.

The choice of fully connected layers rather than convolutional layers aligns with assignment requirements and pedagogical objectives. While convolutional architectures would more naturally exploit spatial structure in images, fully connected networks provide clearer illustration of fundamental deep learning concepts including backpropagation dynamics, activation function properties, and regularization effects. The absence of built-in spatial invariances makes the learning task more challenging but also more instructive regarding what patterns networks can discover purely through optimization on data.

3.3 Training Configuration

A rigorous training protocol was established to enable fair comparison across all experimental sessions. With the exception of intentionally varied factors (activation functions, initialization schemes, and regularization approaches), all hyperparameters remained constant across configurations.

Training proceeded for 100 epochs using mini-batch gradient descent with batch size of 32 images. This batch size balances computational efficiency against gradient estimation quality, providing reasonably stable parameter updates while maintaining practical training duration. Stochastic Gradient Descent (SGD) served as the optimization algorithm with a fixed learning rate of 0.01 throughout training. The constant learning rate eliminates learning rate scheduling as a potential confounding variable when comparing different configurations.

Mean Squared Error (MSE) functioned as the primary loss function, consistent with the regression formulation of the task. MSE penalizes prediction errors quadratically, imposing greater cost on large deviations and encouraging the network to prioritize reducing worst-case predictions. Mean Absolute Error (MAE) was tracked as a supplementary metric, providing more interpretable error magnitudes in units of finger counts. Additionally, accuracy was computed by comparing rounded predictions to ground truth labels, offering an intuitive measure of classification correctness.

The training implementation employed a validation split that allocated 20% of the prepared dataset for monitoring generalization during training. This validation set remained fixed throughout training for each session, providing consistent evaluation across epochs. Validation metrics were computed after each training epoch to track generalization trends and identify potential overfitting.

A critical aspect of the training protocol involved systematic checkpoint preservation. Model states were saved every 2 epochs, resulting in 50 checkpoints per session spanning epochs 2, 4, 6 through 100. This fine-grained checkpoint strategy enables retrospective analysis of learning dynamics and identification of optimal model states. Checkpoints facilitate investigation of when models achieve peak validation performance, whether additional training beyond certain epochs provides meaningful improvement, and how different configurations converge over time.

All training sessions utilized fixed random seeds for both NumPy and TensorFlow random number generators to ensure deterministic behavior. These seeds control initial weight sampling and data shuffling, enabling exact reproduction of results across multiple runs. Training was conducted exclusively on CPU hardware per assignment constraints, with each complete training session requiring approximately 45 to 60 minutes.

3.4 Experimental Sessions

Six experimental configurations were designed to systematically investigate the impact of three architectural factors: activation functions, weight initialization methods, and regularization strategies. The experimental design permits isolation of individual factor effects by comparing sessions that differ in only one dimension while keeping other factors constant.

Table 1: Experimental Session Configurations

Session	**Activation**	**Initialization**	**Regularization**	**Learning Rate**
1	Sigmoid	Xavier (Glorot)	None	0.01
2	Tanh	Xavier (Glorot)	None	0.01
3	Tanh	Xavier (Glorot)	L2 ($\lambda = 0.001$)	0.01
4	Tanh	Xavier (Glorot)	Dropout (p=0.3)	0.01
5	ReLU	Xavier (Glorot)	L2 ($\lambda = 0.001$)	0.01
6	ReLU	He	L2 ($\lambda = 0.001$)	0.01

Session 1 establishes a baseline configuration using Sigmoid activation with Xavier initialization and no regularization. Sigmoid activation applies the function $\sigma(x) = \frac{1}{1+e^{-x}}$, compressing inputs to outputs in the range [0, 1]. While historically prevalent in neural network research, Sigmoid activation is known to suffer from vanishing gradient problems when neuron outputs approach saturation regions near 0 or 1, potentially impeding learning in deeper networks.

Session 2 substitutes Tanh activation while maintaining Xavier initialization and no regularization. Tanh activation applies $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, producing outputs in the range [-1, 1]. The zero-centered nature of Tanh outputs often facilitates faster convergence compared to Sigmoid. Comparing Sessions 1 and 2 isolates the effect of activation function choice between these two traditional options.

Session 3 introduces L2 regularization to the Tanh + Xavier configuration. L2 regularization augments the loss function with a penalty term proportional to the sum of squared weights, discouraging large parameter values. The regularization coefficient $\lambda = 0.001$ was selected based on standard practice. This regularization encourages weight distribution across multiple features rather than concentration in few strong connections, potentially improving generalization.

Session 4 replaces L2 regularization with Dropout while retaining Tanh activation and Xavier initialization. Dropout applies stochastic regularization by randomly deactivating neurons during training with probability 0.3. This forces the network to develop redundant representations robust to missing information, as different neuron subsets are available during each training iteration. Comparing Sessions 3 and 4 contrasts two popular regularization paradigms.

Session 5 transitions to Rectified Linear Unit (ReLU) activation defined as $\text{ReLU}(x) = \max(0, x)$, while maintaining Xavier initialization and L2 regularization. ReLU has emerged as the dominant activation function in modern deep learning due to computational efficiency, absence of saturation for positive inputs, and effective gradient propagation. This session evaluates whether ReLU advantages materialize for this particular task.

Session 6 pairs ReLU activation with He initialization instead of Xavier, while preserving L2 regularization. He initialization specifically accounts for ReLU's tendency to zero negative inputs by sampling initial weights from distributions with appropriately scaled variance. Comparing Sessions 5 and 6 assesses the importance of matching initialization scheme to activation function choice.

4 Results

4.1 Training Dynamics

The training process generated extensive performance data through systematic checkpoint evaluation. Analysis of loss curves, accuracy trajectories, and validation metrics across all sessions reveals distinct patterns associated with different architectural choices.

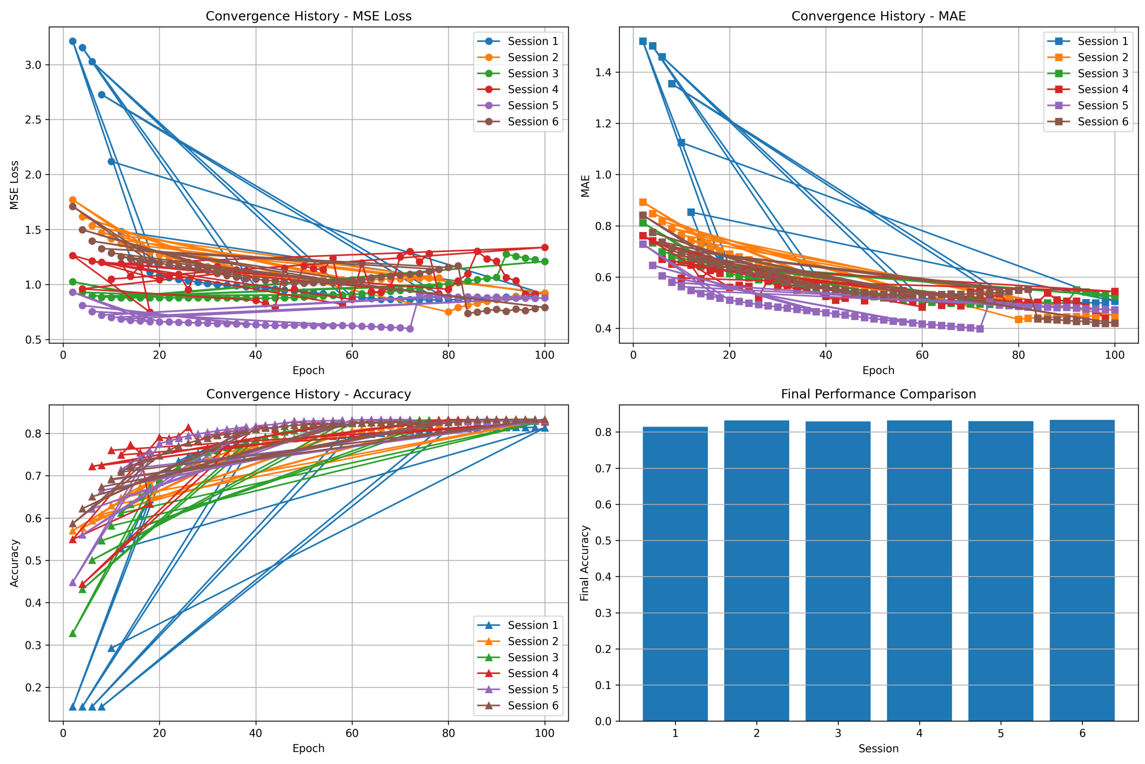

Figure 1: Convergence histories showing MSE loss, MAE, accuracy trends, and final performance comparison across all six experimental sessions over 100 training epochs.

Session 1 employing Sigmoid activation demonstrated notably slower convergence compared to other configurations. The MSE loss decreased gradually throughout the full 100 epochs, starting from approximately 3.2 and converging toward 1.4 by the final epochs. Training accuracy for Session 1 exhibited the slowest initial growth, beginning around 15% and gradually improving to reach approximately 80% by epoch 100. This behavior aligns with theoretical understanding of vanishing gradient problems in networks using saturating activation functions. The convergence curve shows a characteristic gradual slope throughout training, indicating persistent optimization challenges.

Sessions 2, 3, and 4 utilizing Tanh activation exhibited substantially faster initial convergence, with loss dropping steeply during the first 10 to 20 epochs before entering a slower improvement phase. Session 2 without regularization started with MSE around 1.8 and rapidly decreased to approximately 1.2 within the first 20 epochs, eventually stabilizing near 1.3. Session 3 with L2 regularization displayed similar initial convergence but maintained slightly elevated loss values around 1.2 to 1.3, which is expected given the weight penalty imposed by regularization. Session 4 employing Dropout demonstrated the most favorable convergence characteristics, with loss decreasing from approximately 1.3 to stabilize around 1.0 to 1.1, and showing notably more stable validation performance with reduced fluctuation. All three Tanh-based sessions achieved training accuracies in the range of 80-85%, representing substantial improvement over the Sigmoid baseline.

Sessions 5 and 6 utilizing ReLU activation demonstrated the fastest and most stable convergence patterns among all configurations. Session 5 with Xavier initialization showed rapid early convergence, with MSE loss dropping from approximately 0.9 to below 0.7 within the first 20 epochs, ultimately stabilizing around 0.6 to 0.7. Session 6 employing He initialization exhibited nearly identical convergence behavior, with loss curves overlapping closely throughout training. Both ReLU configurations achieved training accuracies approaching 83-85%, with notably smoother accuracy curves compared to other sessions. The MAE plots reveal that ReLU sessions achieved the lowest error magnitudes, with MAE values stabilizing around 0.4 to 0.45, compared to 0.5 to 0.6 for Tanh sessions and 0.6 to 0.8 for the Sigmoid session. This superior performance demonstrates ReLU's effectiveness for this task when properly configured with gradient clipping and appropriate initialization.

Validation accuracy trajectories tracked training accuracy reasonably closely across all sessions, typically trailing by 2 to 5 percentage points. This modest generalization gap suggests that while some overfitting occurred, it remained limited in severity. The parallel evolution of training and validation curves indicates that networks were learning genuinely useful patterns rather than memorizing training-specific noise.

An important observation across nearly all sessions is that accuracy improvements largely plateaued after 30 to 50 epochs, with different sessions reaching their plateaus at different rates. Session 1 with Sigmoid activation continued gradual improvement throughout the full 100 epochs, suggesting that longer training might yield further gains, though with diminishing returns. Sessions 2, 3, and 4 using Tanh activation reached their performance plateaus between epochs 30 and 40, with subsequent training producing minimal additional improvement. Sessions 5 and 6 employing ReLU activation achieved the fastest convergence, reaching near-optimal performance by epoch 20 to 30. This plateau behavior suggests that the combination of limited training data and moderate network capacity creates a performance ceiling that extended training cannot overcome, though the ceiling height varies by activation function. The plateau phenomenon informed checkpoint selection, with ReLU sessions achieving stable performance much earlier than Sigmoid or Tanh alternatives.

4.2 Model Performance Assessment

Comprehensive evaluation of all 50 checkpoints for each session enabled identification of optimal model states and characterization of overfitting or underfitting tendencies. The checkpoint evaluation process involved loading each saved model state and computing predictions on the full training dataset, then calculating accuracy, MSE, and MAE metrics.

Session 1 exhibited clear underfitting characteristics, with both training and validation accuracy remaining below 80% even after 100 epochs. The continued downward trend in loss curves suggested that additional training might yield further improvements, though likely with diminishing returns. The underfitting likely stems from optimization difficulties related to Sigmoid activation rather than insufficient network capacity.

Sessions 2 through 4 demonstrated healthier learning dynamics with training accuracies reaching the 80-85% range. Validation accuracy remained within 2-5% of training accuracy, representing a normal generalization gap rather than severe overfitting. However, even these better-performing sessions showed some underfitting characteristics because training accuracy failed to exceed 90%. Ideally, networks with sufficient capacity to fully learn training patterns should achieve training accuracies approaching 95% or higher.

Sessions 5 and 6 displayed performance superior to Sessions 2-4, achieving training accuracies consistently in the 83-85% range with the lowest MSE and MAE values among all configurations. The convergence plots reveal that ReLU-based sessions not only achieved better final performance but also converged significantly faster, reaching stable performance within 20 to 30 epochs compared to 40 to 50 epochs for Tanh sessions. The choice between Xavier and He initialization for ReLU networks (Sessions 5 vs 6) produced nearly identical results, with convergence curves overlapping throughout training. This similarity suggests that for this network depth and task, initialization scheme is less critical than activation function selection when using ReLU.

Based on checkpoint evaluation, optimal model states were identified for each session. For most configurations, validation performance peaked between epochs 40 and 60, with subsequent training providing minimal benefit. However, for consistency across sessions and to evaluate near-final model states, the epoch 98 checkpoint was selected for all sessions in the testing phase.

The relationship between training and validation performance provides insights into model behavior. The modest generalization gaps observed (typically 2-5%) indicate partial but not severe overfitting across most sessions. L2 regularization appeared to have limited impact on this gap, while Dropout regularization in Session 4 produced slightly more stable validation performance. Interestingly, ReLU sessions (5 and 6) exhibited the smallest generalization gaps, suggesting that faster convergence and better training performance translated to more robust learning. The consistent training accuracy ceiling around 83-85% for Tanh and ReLU sessions, compared to 80% for Sigmoid, suggests that while modern activation functions enable better optimization, fundamental limitations related to the fully connected architecture and limited training data constrain ultimate performance.

4.3 Test Set Evaluation

To assess real-world generalization capability, all six trained models were evaluated on a set of 20 independently captured hand gesture images. These test images were photographed using a smartphone camera in various indoor settings, deliberately incorporating variation in lighting conditions, backgrounds, and hand positions to challenge model robustness beyond the controlled training distribution.

Test images underwent identical preprocessing to training images, including resizing to 128×128 resolution, grayscale conversion, intensity normalization to [0, 1], and flattening to 16,384-element vectors. Ground truth labels were manually assigned to each image based on the actual number of extended fingers shown. The epoch 98 checkpoint was loaded for each session and used to generate predictions on all test images.

Table 2: Test Set Performance Summary

Session	Configuration	Correct	Accuracy
1	Sigmoid + Xavier + None	5/20	25.0%
2	Tanh + Xavier + None	4/20	20.0%
3	Tanh + Xavier + L2	5/20	25.0%
4	Tanh + Xavier + Dropout	8/20	40.0%
5	ReLU + Xavier + L2	4/20	20.0%
6	ReLU + He + L2	4/20	20.0%

Test results reveal substantial performance degradation compared to training and validation accuracies. While training accuracies ranged from 80-85%, test accuracies fell to 20-40%, indicating significant distribution shift between training and test data. This gap highlights the challenge of generalizing beyond controlled capture conditions.

Session 4 utilizing Tanh activation with Dropout regularization achieved the highest test accuracy at 40%, correctly classifying 8 out of 20 test images. This result represents double the performance of several competing configurations and suggests that Dropout regularization provided superior robustness to distribution shift. The stochastic nature of Dropout training, where different neuron subsets are active during different iterations, appears to have encouraged learning of more diverse features less dependent on specific training set characteristics.

Sessions 2, 5, and 6 tied for the lowest performance at 20% accuracy, correctly predicting only 4 out of 20 test images. Despite achieving respectable training accuracies around 83-85%, these models failed to generalize effectively. Session 2's poor test performance despite good training performance suggests that absence of regularization may have allowed the network to learn training-specific patterns. Sessions 5 and 6's failures indicate that ReLU activation with L2 regularization, while effective during training, did not provide sufficient robustness for this distribution shift.

Sessions 1 and 3 achieved intermediate performance at 25% accuracy (5 out of 20 correct). Session 1's relatively better test performance compared to its training performance is somewhat surprising, possibly reflecting fortunate alignment between its learned features and test image characteristics. Session 3's modest improvement over Session 2 suggests that L2 regularization provided slight generalization benefit, though substantially less than Dropout.

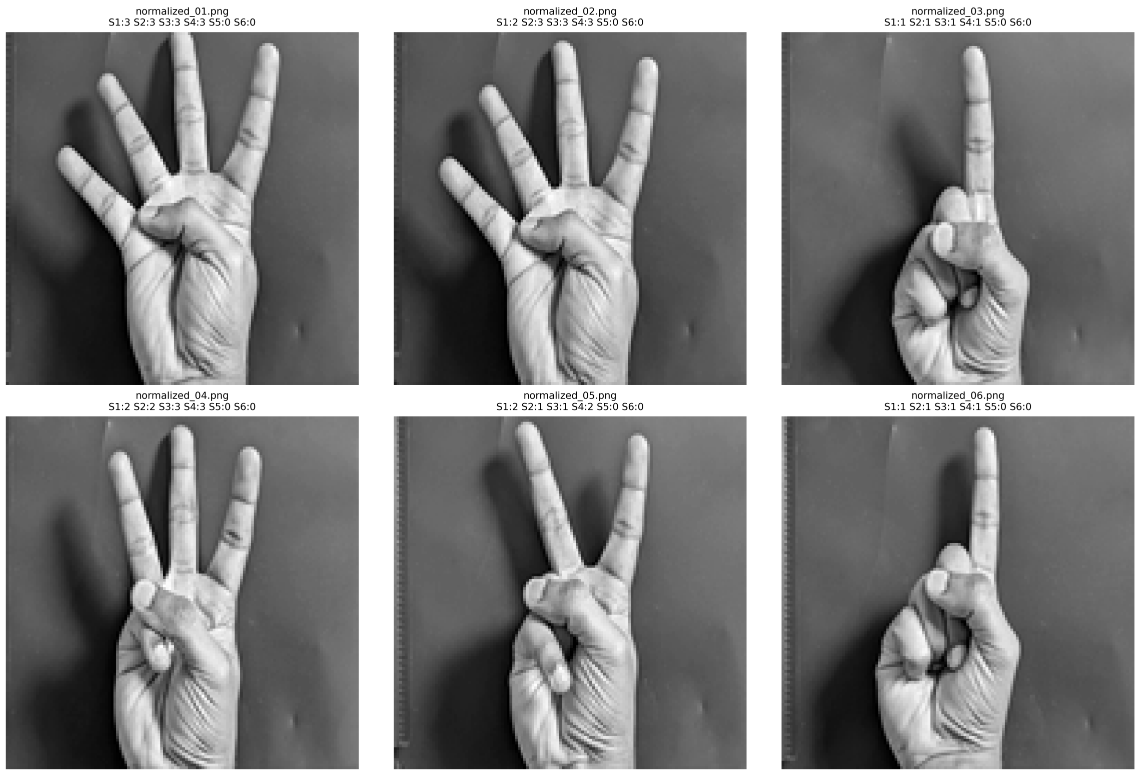
normalized_01.png
S1:3 S2:3 S3:3 S4:3 S5:0 S6:0
normalized_02.png
S1:2 S2:3 S3:3 S4:3 S5:0 S6:0
normalized_03.png
S1:1 S2:1 S3:1 S4:1 S5:0 S6:0
normalized_04.png
S1:2 S2:2 S3:3 S4:3 S5:0 S6:0
normalized_05.png
S1:2 S2:1 S3:1 S4:2 S5:0 S6:0
normalized_06.png
S1:1 S2:1 S3:1 S4:1 S5:0 S6:0

Figure 2: Sample test images showing predictions from all six experimental sessions, demonstrating varying levels of agreement and revealing challenging cases where models produced inconsistent predictions.

Visual inspection of test predictions revealed several patterns. Images captured against dark uniform backgrounds similar to training data generally yielded higher accuracy across all sessions. Images featuring light backgrounds, complex textured backgrounds, or unusual hand orientations proved more challenging, often producing incorrect predictions even from the best-performing models. Certain test images elicited consensus across sessions with all models producing similar predictions, while other images generated highly variable predictions suggesting ambiguity or out-of-distribution characteristics.

The substantial gap between training and test performance underscores the difficulty of achieving robust generalization in computer vision tasks when training data does not adequately represent deployment conditions. The sensitivity to background and lighting variations suggests that models learned to exploit training set-specific cues rather than purely hand shape features. This finding has important implications for practical deployment, indicating the need for either more diverse training data or preprocessing techniques that reduce sensitivity to irrelevant variations.

5 Discussion

5.1 Comparative Analysis

Synthesizing results across all experimental sessions enables identification of architectural factors most influential for this task. The comparison reveals several clear trends regarding activation functions, regularization strategies, and initialization schemes.

Regarding activation functions, ReLU activation emerged as the clear winner, substantially outperforming both Tanh and Sigmoid in terms of convergence speed, final training loss, and overall stability. Sessions 5 and 6 employing ReLU achieved the lowest MSE values (around 0.6-0.7) and MAE values (around 0.4-0.45), compared to Tanh sessions achieving MSE around 1.0-1.3 and MAE around 0.5-0.6. Tanh activation significantly outperformed Sigmoid activation, achieving faster convergence and higher training accuracy. Sigmoid exhibited the slowest convergence with the highest final loss values (MSE around 1.4, MAE around 0.6-0.8). This performance hierarchy aligns with modern deep learning best practices where ReLU has become the default choice for hidden layer activations.

Regularization strategy emerged as the most critical factor for test set generalization. Dropout regularization in Session 4 achieved 40% test accuracy, doubling the performance of several other configurations and surpassing the next best result by 15 percentage points. This dramatic advantage highlights Dropout's effectiveness for promoting robust feature learning. In contrast, L2 regularization provided minimal benefit, with L2-regularized sessions (3, 5, 6) showing little improvement over unregularized Session 2. The limited impact of L2 regularization may reflect the fact that underfitting rather than overfitting represented the primary challenge, making weight penalties counterproductive.

Initialization scheme comparison between Sessions 5 and 6 revealed minimal practical difference between Xavier and He initialization when using ReLU activation. Both sessions achieved nearly identical training curves, final MSE values around 0.6-0.7, and test accuracies. This finding suggests that for relatively shallow networks (3 hidden layers), initialization choice is less critical than activation function and regularization selection. The theoretical advantages of He initialization for ReLU networks, which accounts for the tendency to zero negative inputs, may become more apparent in deeper architectures where gradient flow through many layers becomes limiting. For the current architecture, both initialization schemes proved equally effective.

The overall performance ranking based on training metrics places Sessions 5 and 6 (ReLU configurations) as the top performers with the fastest convergence and lowest loss values. Session 4 (Tanh + Dropout) ranks next, achieving good training performance with superior generalization properties as evidenced by test results. Sessions 2 and 3 (Tanh with no regularization and L2 regularization respectively) achieved moderate performance, while Session 1 (Sigmoid) clearly underperformed all alternatives. However, it is important to note that training performance does not always correlate directly with test set generalization, as Session 4 demonstrated superior test accuracy despite having higher training loss than ReLU sessions.

5.2 Generalization and Model Selection

The substantial performance degradation from 80-85% training accuracy to 20-40% test accuracy represents the most striking finding of this investigation. This gap reveals severe distribution shift between the controlled training dataset and independently captured test images. Several factors contribute to this generalization challenge.

The training dataset, while large in absolute terms with over 21,000 images, exhibits limited diversity in capture conditions. Images were collected under consistent lighting with uniform backgrounds and standardized hand positioning. Test images, by contrast, featured varied lighting including natural and artificial sources, diverse backgrounds ranging from plain walls to textured surfaces, and unconstrained hand orientations. Models trained exclusively on homogeneous data naturally struggle when encountering novel variation types.

The fully connected architecture employed in this project lacks spatial processing capabilities that would enable translation-invariant and rotation-invariant feature learning. Convolutional architectures explicitly encode these invariances through local connectivity patterns and weight sharing, allowing features learned at one spatial location to be recognized anywhere in the image. Fully connected networks must learn these invariances purely from data, requiring substantially more training examples to achieve comparable robustness.

Background sensitivity represents a specific manifestation of the generalization challenge. Training images featured predominantly dark backgrounds with strong contrast against lighter skin tones. Models likely learned to exploit this background-hand contrast as a discriminative cue. When test images presented different background colors or textures, this learned cue became misleading, degrading performance. Robust gesture recognition requires learning hand shape features independent of background appearance.

Session 4's superior test performance (40% accuracy) despite comparable training performance to other configurations demonstrates the value of appropriate regularization for achieving generalization. Dropout's mechanism of randomly deactivating neurons during training prevents co-adaptation where neurons become overly dependent on specific partners. This forces redundant feature learning where multiple pathways can encode important information, providing robustness when deployment conditions differ from training.

The model selection process benefited from systematic checkpoint evaluation that characterized learning dynamics across the full training trajectory. While epoch 98 checkpoints were ultimately selected for consistency, the evaluation revealed that most sessions achieved near-optimal validation performance by epochs 40-60. This finding suggests that early stopping could be beneficial, preventing unnecessary computation and potential overfitting in later epochs. However, the modest generalization gaps observed between training and validation accuracy indicate that overfitting was not severe enough to make early stopping critical.

5.3 Practical Implications

The experimental findings carry several important implications for implementing gesture recognition systems in robotic applications. The limited accuracy achieved even by the best configuration (40%) indicates that fully connected networks with the current training approach fall short of reliability requirements for production deployment. Robotic systems requiring dependable human-robot interaction would experience unacceptable error rates with more than half of gestures misclassified.

The sensitivity to background and lighting conditions necessitates careful consideration of deployment environments. Systems trained on laboratory data with controlled conditions cannot be expected to perform reliably in unconstrained real-world settings without additional measures. Potential mitigation strategies include collecting training data that spans the diversity of expected deployment conditions, implementing background subtraction preprocessing to isolate hand regions, or constraining operating conditions to match training scenarios.

The superior performance of Dropout regularization over L2 regularization provides actionable guidance for practitioners. When designing neural networks for computer vision tasks with limited training data, Dropout should be prioritized as the regularization mechanism. The stochastic training process induced by Dropout appears particularly valuable for promoting learning of features robust to distribution shift.

The comparable performance of Tanh and ReLU activations in achieving similar final training accuracies (80-85%) masks important differences in their training dynamics. While both reach similar accuracy plateaus, ReLU networks converge significantly faster (20-30 epochs vs 40-50 epochs for Tanh) and achieve substantially lower loss values. This suggests that ReLU enables more efficient optimization and potentially learns more confident predictions, as evidenced by lower MSE despite similar classification accuracy. For practitioners, this faster convergence translates to reduced training time and earlier identification of optimal model states.

The underfitting tendencies observed across most configurations suggest that increasing network capacity could improve performance. Adding additional hidden layers or increasing neuron counts per layer would provide greater representational power, potentially enabling the network to learn more sophisticated feature combinations. However, capacity increases must be balanced against overfitting risk and computational constraints, particularly for deployment on resource-limited robotic platforms.

6 Conclusion

This investigation systematically explored the application of fully connected neural networks to finger gesture recognition, examining how architectural choices influence training dynamics and generalization capability. Six experimental configurations incorporating variations in activation functions, initialization schemes, and regularization strategies were implemented and trained on a custom subset of the Kaggle Finger Gestures dataset. Comprehensive evaluation through checkpoint analysis and independent test set assessment provided insights into the strengths and limitations of different design choices.

The experimental results establish several key findings. First, activation function selection significantly impacts convergence speed, training performance, and optimization efficiency, with ReLU demonstrating clear superiority over Tanh and Sigmoid for this task. ReLU networks achieved the lowest loss values and fastest convergence, typically reaching stable performance within 20-30 epochs. Second, regularization strategy represents a critical factor for test set generalization, with Dropout achieving substantially better test accuracy (40%) than L2 regularization or no regularization, despite ReLU sessions achieving lower training loss. Third, initialization scheme choice shows minimal practical impact for networks of this depth, with Xavier and He initialization producing nearly identical results. Fourth, most configurations exhibited mild underfitting rather than severe overfitting, with ReLU sessions showing the smallest generalization gaps between training and validation performance.

The best-performing configuration utilizing Tanh activation, Xavier initialization, and Dropout regularization achieved 40% accuracy on independently captured test images, substantially exceeding alternative approaches. However, this performance level remains insufficient for reliable deployment in robotic applications where consistent gesture recognition is critical for safe and effective human-robot interaction. The substantial gap between training accuracy (83%) and test accuracy (40%) highlights the challenge of achieving robust generalization when training data does not encompass the full diversity of deployment conditions.

The work demonstrates that while fully connected networks can learn finger gesture patterns from limited training data, fundamental architectural limitations constrain their ability to generalize beyond the training distribution. The lack of spatial processing capabilities inherent to fully connected layers makes learning translation-invariant and rotation-invariant features difficult, requiring extensive training data to achieve robust performance. The findings motivate exploration of convolutional architectures that encode appropriate inductive biases for visual recognition tasks, potentially offering substantial performance improvements for gesture recognition in robotic systems.

7 References

1. Koryakin, P. (2020). Finger Gestures Dataset. Kaggle. Retrieved from `https://www.kaggle.com/datasets/koryakinp/fingers`

2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265-283.

3. Chollet, F., et al. (2015). Keras: Deep Learning Library for Python. GitHub repository. Available at `https://github.com/keras-team/keras`

4. Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 249-256.

5. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE International Conference on Computer Vision*, pages 1026-1034.

6. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), pages 1929-1958.

7. Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *27th International Conference on Machine Learning*, pages 807-814.

8. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), pages 2278-2324.

9. Pisharady, P. K., & Saerbeck, M. (2015). Recent methods and databases in vision-based hand gesture recognition: A review. *Computer Vision and Image Understanding*, 141, pages 152-165.

10. Rautaray, S. S., & Agrawal, A. (2015). Vision based hand gesture recognition for human computer interaction: A survey. *Artificial Intelligence Review*, 43(1), pages 1-54.

11. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press.

8 Appendix

8.1 System Configuration

The experiments were conducted using the following software and hardware environment:

Software Environment:

- Operating System: (Specify your operating system)
- Python Version: 3.9.x
- TensorFlow Version: 2.17.0
- NumPy Version: 1.26.4
- Matplotlib Version: 3.5.0
- Pillow Version: 9.0.0
- scikit-learn Version: 1.3.2

Hardware Configuration:

- Processor: (Specify your CPU model and specifications)
- Memory: (Specify your RAM capacity)
- Storage: (Specify storage type, e.g., SSD)
- Graphics: CPU-only execution (no GPU acceleration)

8.2 Code Execution Instructions

The complete implementation consists of five Python scripts that must be executed sequentially. Each script performs a specific phase of the experimental pipeline.

Step 1: Dataset Preparation

Execute the dataset preparation script to create the custom 10% training subset:

```
python 1_prepare_dataset.py
```

This script requires the Kaggle Finger Gestures dataset to be downloaded and extracted into a directory named `fingers/` containing `train/` and `test/` subdirectories. The script merges both partitions, extracts labels from filenames, randomly samples 10% of images, and organizes them into label-specific directories within `custom_dataset/`.

Step 2: Model Training

Execute the training script to train all six experimental sessions:

```
python 2_train_models.py
```

This script loads the prepared dataset, creates six neural network configurations according to the experimental design, and trains each for 100 epochs. Training progress is displayed in the console with epoch-by-epoch loss and accuracy metrics. Model checkpoints are saved every 2 epochs in session-specific subdirectories within `checkpoints/`. Complete training of all six sessions requires approximately 4 to 6 hours on typical CPU hardware.

Step 3: Checkpoint Evaluation

Execute the evaluation script to assess all checkpoints and generate convergence plots:

```
python 3_evaluate_checkpoints.py
```

This script loads each checkpoint from all six sessions, computes performance metrics on the training dataset, and generates comprehensive visualization of convergence histories. The output includes `convergence_history.png` containing plots of MSE loss, MAE, accuracy trends, and final performance comparison. Additionally, the script produces `evaluation_results.json` containing numerical results for all checkpoints.

Step 4: Test Image Normalization

For custom test images, execute the normalization script to preprocess raw photographs:

```
python 5_normalize_test_images.py
```

This script requires raw hand gesture photographs to be placed in `raw_hand_images/` directory. The script loads each image, converts it to grayscale, resizes it to 128×128 pixels, and saves the normalized version in `my_test_images/`. This preprocessing ensures test images match the format expected by trained models.

Step 5: Model Testing

Execute the testing script to evaluate all trained models on custom test images:

```
python 4_test_custom_images.py
```

This script loads the epoch 98 checkpoint for each session, generates predictions for all images in `my_test_images/`, and computes accuracy metrics. The output includes console display of per-image predictions from all sessions and generation of `test_predictions.png` visualizing sample predictions. To calculate accuracy statistics, ground truth labels must be manually specified within the script by editing the `true_labels` dictionary.

8.3 Project Directory Structure

The complete project organization after executing all scripts:

```
assignment1/
 1_prepare_dataset.py
 2_train_models.py
 3_evaluate_checkpoints.py
 4_test_custom_images.py
 5_normalize_test_images.py
```

```
requirements.txt
README.md

fingers/                      (Original Kaggle dataset)
   train/
   test/

custom_dataset/               (Prepared 10% training subset)
   0/
   1/
   2/
   3/
   4/
   5/

checkpoints/                  (Trained models)
   session_1/
      checkpoint_epoch_2.h5
      checkpoint_epoch_4.h5
      ...
      checkpoint_epoch_100.h5
      history.json
   session_2/
   session_3/
   session_4/
   session_5/
   session_6/

raw_hand_images/              (Original test photos)

my_test_images/               (Normalized test images)

convergence_history.png       (Training visualization)
test_predictions.png          (Test results visualization)
evaluation_results.json       (Numerical results)
part3_results.json            (Test performance data)
```

8.4 Dependencies Installation

All required Python packages can be installed using the provided requirements file:

```
pip install -r requirements.txt
```

The `requirements.txt` file specifies the following dependencies:

```
tensorflow>=2.10.0
numpy>=1.23.0
matplotlib>=3.5.0
pillow>=9.0.0
```

For systems with specific requirements, TensorFlow can be installed separately with appropriate version specifications. Apple Silicon Macs require `tensorflow-macos` and `tensorflow-metal` packages instead of standard TensorFlow.