Simple CNN Model Analysis

Farid Hajizada
hajizada1farid@kaist.ac.kr

January 10, 2025

Introduction

In this report, we will classify images using our own CNN architecture. By mimicking the visual processing system of the human brain, CNNs utilize convolutional layers to extract features such as edges and textures, which are then used to make predictions. Since learning hierarchical patterns from raw images is one of its main objectives, CNN performed impressive results in many areas.

Dataset

Our dataset is 10 classes, 15 images of people per class for training, and 15 other images per class for evaluation. Additionally, we normalised sizes and utilized zero mean centring.

Artitechture

Our architecture design is given in the Figure 1. The basic convolutional block structure consists of a convolutional layer with $k \times k$ kernel, followed by a normalization layer and ReLU activation. This pattern is consistently used throughout the network to maintain effective feature processing and provide non-linearity.

In the skip connections, 1×1 convolutions are employed when `skipping=True`. These serve two main purposes: matching channel dimensions when the main path changes feature dimensions, and enabling efficient information flow through skip connections with minimal additional parameters.

The network follows an expansive channel strategy ($3\rightarrow32\rightarrow64\rightarrow128$) to capture increasingly complex features while balancing computational cost with representation capacity. This gradual expansion allows the network to build up its feature hierarchy efficiently. The final layer is classification layer which classifies one of the 10 classes.

HyperParameters and Their Effect

General setup

First we set up the number of layers to be 3(i.e. there is no repeated convolutional block). Base settings are no skipping, and no dropout and we set batch normalisation and kernel size 3. The batch size is 32 and the optimizer is SGD. The loss function is Cross Entropy loss and the learning rate is 0.001. Throughout all experiments, the number of epochs is 50. We will do a greedy search, which means after finding the best resulting hyperparameter we will set it for further experiments.

Effect of Number of Layers

Number of Convolutional Layers	Evaluation Accuracy	Training Accuracy	Training Time (s)
3	66.00%	98.67	15.02
4	66.00%	99.33	15.24
5	68.67%	98.67	15.43
6	64.00%	98.67	15.66
7	66.67%	99.67	15.89
8	65.33%	98.67	16.20
9	65.33%	99.33	16.37

Table 1: Training and Evaluation Accuracy for Different Numbers of Convolution Layers

From Table 1, we observe that the best evaluation accuracy (68.67%) is achieved with 5 convolutional layers, corresponding to 2 repeated convolutional blocks. When the number of layers is small (3 or 4), the network is too shallow to capture complex features, leading to lower accuracy. Increasing the number of layers beyond this point results in a slight drop in evaluation accuracy, likely due to overfitting caused by an excessive number of parameters. This overfitting is evident as the training accuracy approaches 100% for most configurations.

Additionally, the training time increases with the number of layers, which is expected due to the growing number of parameters.

In summary, while increasing the number of layers can improve the model's ability to capture complex patterns, it also risks overfitting and increases computational cost. The optimal configuration in this case appears to be **5 layers**.

Effect of Convolutional filter sizes

Kernel Size	Evaluation Accuracy
3	68.67%
5	62.67%
7	62.00%

Table 2: Evaluation Accuracy for Different Kernel Sizes

From Table 2, we can observe that the best accuracy is achieved with a **kernel size of 3**. This suggests that a 3×3 kernel is better at capturing local features. One possible reason for the lower accuracy with larger kernels

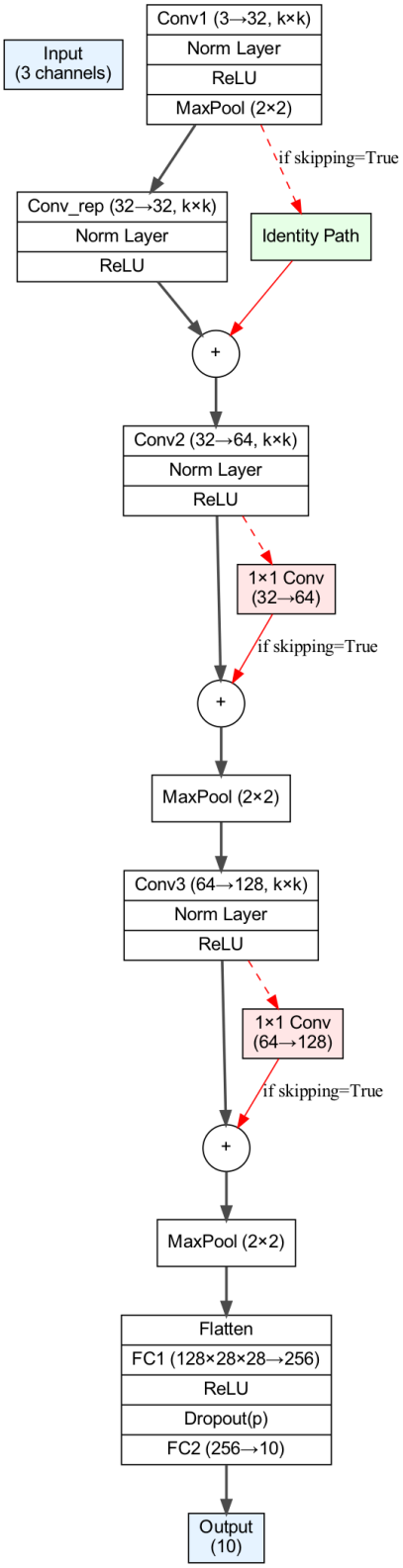

Figure 1: CNN Architecture. Parameters: Kernel size (k), Normalisation (batch/group/instance/layer), Dropout(Optional), Linear layer dropout (p), Number of convolution layers(N), Repeated block applied(N-3) times

.

is that they may capture unnecessary information due to their size. Additionally, in most CNN models, a kernel size of 3 is commonly used.

Effect of Skipping Connections

With the skipping connection, our evaluation accuracy dropped to 66.67%, while the training accuracy remains at 99.33%. The main reason for this is that the skipping connection is not well-suited to our architecture. Skipping connections are typically beneficial in deeper networks, where they help mitigate the vanishing gradient problem. However, in our case, since the architecture is not very deep, the skipping connection has little impact.

Effect of Normalizations

In our base setting, we used batch normalization. We then analyzed the effect of other normalization methods, including GroupNorm, LayerNorm, and InstanceNorm.

From Figure 2 in the Appendix, we observe that **batch normalization** achieves the best evaluation accuracy, slightly outperforming Instance and Group Normalization. The worst performance is seen with Layer Normalization. All normalization methods reached near 100% accuracy for the reasons mentioned earlier.

A similar trend is observed in the evaluation losses. After 50 epochs, the losses for BatchNorm, GroupNorm, and InstanceNorm are nearly identical, converging around 1.17, while Layer Normalization converged to 1.5. For the training losses, all methods reached values close to 0, again due to the reasons previously discussed.

Effect of Dropout and Regularization Term

To enhance the generalization capabilities of our Convolutional Neural Network (CNN) and mitigate overfitting, we incorporate dropout and weight regularization techniques into our training process. Dropout randomly deactivates a subset of neurons during each training iteration, preventing the model from becoming overly reliant on specific pathways and promoting the development of robust features. Additionally, weight decay introduces a regularization term that penalizes large weights, encouraging simpler models that perform better on unseen data.

	0.00001	**0.0001**	**0.001**	**0.01**
0.1	68.67%	62.67%	66.00%	69.33%
0.2	63.33%	63.33%	67.33%	67.33%
0.3	62.00%	63.33%	64.00%	66.00%
0.4	67.33%	66.67%	66.67%	64.67%
0.5	69.33%	66.00%	65.33%	62.67%

Table 3: Evaluation Accuracy for Different Dropout and Weight Decay Configurations (Columns: Weight Decay, Rows: Dropout)

The analysis of the dropout and weight decay configurations reveals that a lower dropout value combined with

moderate weight decay generally yields the best performance. Specifically, the combination of **dropout 0.1** and **weight decay 0.01** achieved the highest evaluation accuracy of `69.33%`, indicating that this configuration strikes the best balance between regularization and model performance. Higher dropout values (such as `0.3` and `0.5`) tend to result in lower evaluation accuracy, especially when paired with smaller weight decay values, which suggests that excessive regularization can lead to underfitting. On the other hand, smaller weight decay values like `0.00001` work well with lower dropout values (e.g., `0.1` and `0.5`), but when used with higher dropout, they lead to performance degradation. Overall, these results highlight the importance of fine-tuning both dropout and weight decay to avoid overfitting or underfitting, with the optimal configuration being a lower dropout rate and moderate weight decay. This setting increased our so far best performer also(`68.67%`).

Effect of Squared Hinge Loss

In our experiment with Squared Hinge Loss, we achieved an accuracy of 52.00%. This lower performance is likely due to the fact that squared hinge loss is more suited for binary classification tasks, where the margin between two classes is clearer. In multiclass problems, it may not capture the complexities of distinguishing multiple classes, leading to reduced accuracy. The detailed information about loss functions can be found in the Appendix .

Effect of Changing fully connected layer with truncated SVD

Compression Ratio	Parameters	Parameter Reduction (%)	Training Acc (%)	Evaluation Acc (%)	Accuracy Change (%)	Relative Accuracy Change (%)
1.0 (Baseline)	25,805,258	0.00	100.00	69.33	0.00	0.00
0.9	23,254,793	9.88	100.00	63.33	-6.00	-8.67
0.8	20,638,692	20.02	100.00	60.00	-9.33	-13.47
0.7	18,123,200	29.77	98.67	58.67	-10.66	-15.47
0.6	15,507,099	39.91	92.00	52.67	-16.66	-23.96
0.5	12,991,607	49.66	80.67	46.67	-22.66	-32.66
0.4	10,375,506	59.79	70.00	42.67	-26.66	-38.36
0.3	7,759,405	69.93	59.33	40.67	-28.66	-41.34
0.2	5,243,913	79.68	42.67	29.33	-40.00	-57.66
0.1	2,627,812	89.82	20.00	12.00	-57.33	-82.72

Table 4: SVD Compression Analysis Results with Relative Accuracy Change

Truncated SVD approximates a matrix A by decomposing it into three matrices: $A = U\Sigma V^T$ where U and V are orthogonal matrices, and Σ is a diagonal matrix of singular values. By truncating the singular values, we retain only the largest k singular values and corresponding vectors, reducing the dimensionality. The compressed matrix A_k is given by: $A_k = U_k \Sigma_k V_k^T$ where U_k, Σ_k, and V_k represent the truncated matrices corresponding to the top k singular values. The compression ratio is determined by $\frac{k}{n}$,where n is the original dimension of the matrix A.

The results in Table 4 show the trade-off between compression ratio and accuracy. As the compression ratio decreases, the number of parameters is reduced, but the evaluation accuracy also drops. For example, at compression 1.0 (baseline), the model achieves an accuracy of 69.33%. As compression increases, accuracy decreases, with the most aggressive compression (0.1) leading to a drastic accuracy drop to 12%. The relative accuracy change compared to the baseline model shows that compression 0.9 offers the best balance, with an 8.67% reduction in accuracy and a 9.88% reduction in parameters. This configuration achieves the highest accuracy (63.33%) while still significantly reducing the number of parameters. Therefore, compression 0.9 represents the best trade-off between model size and performance, making it the most efficient model for deployment with acceptable accuracy loss.

Effect of Learning Rates

Learning Rate	Best Training Accuracy (%)	Best Evaluation Accuracy (%)
0.1	100.00	50.66
0.01	100.00	68.00
0.001	100.00	69.33
0.0001	98.00	58.66

Table 5: Learning Rate Analysis for Base Model

The table 5 in the Appendix, presents the model performance with varying learning rates. As the learning rate decreases from 0.1 to 0.0001, the training accuracy remains at 100% for rates of 0.1, 0.01, and 0.001, but the evaluation accuracy shows a notable improvement, peaking at `69.33%` with a learning rate of 0.001. A learning rate of 0.1 results in a lower evaluation accuracy (`50.66%`), likely due to overshooting during training. The 0.0001 learning rate causes both a drop in training accuracy (`98%`) and evaluation accuracy (`58.66%`), suggesting it is too small for effective convergence. Thus, a learning rate of **0.001** provides the best balance between training performance and generalization, yielding the highest evaluation accuracy.

At higher learning rates (e.g., 0.1), the model initially converges rapidly but may overshoot, leading to poor evaluation accuracy, indicating instability. As the learning rate decreases (e.g., 0.001), the convergence becomes smoother, and the model achieves better generalization, as reflected in the improved evaluation accuracy. However, very small learning rates (e.g., 0.0001) result in slower convergence and a slight drop in both training and evaluation accuracy, suggesting that the rate is too small for effective learning. Overall, a learning rate of 0.001 strikes an optimal balance between fast convergence and generalization, offering the best performance.

Effect of Batch Sizes

I ran the experiment with different batch sizes: 16, 32, 64, 128 and the results did not change. This is expected since dataset is simple and small(150 images). The model is likely able to achieve near-optimal performance without needing much fine-tuning of the batch size.

Effect of Optimizers

RMSprop (Root Mean Square Propagation) is an adaptive learning rate optimization algorithm. It maintains per-parameter learning rates by dividing by an exponentially decaying average of squared gradients, which helps prevent the learning rates from decreasing too quickly and allows the algorithm to continue learning effectively even after seeing many examples.

Optimizer	Best Training Accuracy (%)	Best Evaluation Accuracy (%)
SGD	100.00	69.33
SGD Momentum(0.9)	100.00	72.66
Adam	100.00	59.33
RMSprop	100.00	56.66

Table 6: Optimizer Analysis for Base Model

The table 6 shows that **SGD with momentum** provides the highest evaluation accuracy (`72.66%`), outperforming other optimizers. While SGD achieves the same training accuracy of `100%`, its evaluation accuracy is lower (`69.33%`), suggesting better generalization with momentum. Both Adam and RMSprop result in lower evaluation accuracies (`59.33%` and `56.66%`, respectively), indicating that these optimizers may not be as well-suited for this particular model and dataset despite achieving `100%` training accuracy.

From Figure 4 in the Appendix, SGD with Momentum shows the best performance, achieving the fastest initial convergence and a stable final evaluation accuracy of 72.66%. Adam converges second-fastest but settles at a lower accuracy of 56.66%. Standard SGD has slower initial learning but eventually matches Momentum's performance. Rmsprop, despite rapid loss reduction, exhibits the most instability, with significant fluctuations in both training and evaluation metrics.

Difference Between Random Weight Initialization and Pre-rained Weight

Using pre-trained weights from the **ResNet-18** model, we achieved an accuracy of 98.67% with the SGD-momentum optimizer. In contrast, the model with random weight initialization reached a maximum accuracy of only 72.66%. Both experiments ultimately saw the training accuracy peak at 100%, highlighting the crucial impact of weight initialization on model performance. One of the main factors affecting this result is the simplicity of our training and testing datasets. The datasets likely contained relatively easy patterns for the model to learn, reducing the complexity of the problem and allowing both models to quickly converge to high accuracy. This could explain why both models achieved 100% training accuracy, but the difference in initialization significantly influenced the generalization performance on unseen data.

Appendix

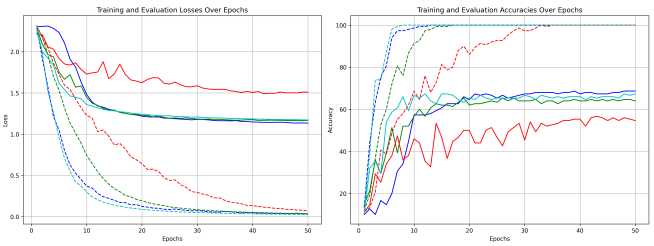

Figure 2: a)Training and Evaluation loss with different normalization techniques b)Training and Evaluation accuracy with different normalization techniques The dashed lines show training results while solid lines represent evaluation. Blue-BatchNorm, Green-GroupNorm, Red-Layer Normalisation, Cyan-Instance Normalization

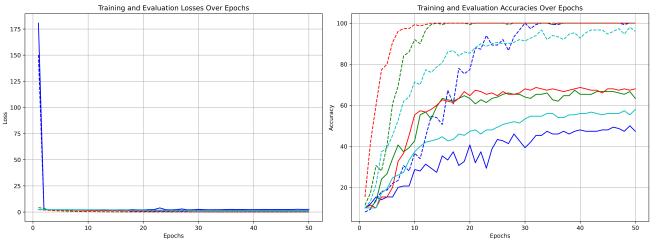

Figure 3: a) Training and Evaluation loss with different learning rates b)Training and Evaluation accuracy with different learning rates. The dashed lines show training results while solid lines represent evaluation. Blue-0.1, Green-0.01, Red-0.001, Cyan-0.0001

Loss functions

For a dataset with N examples, where each example i has a true label y_i and the model produces predicted scores f_j for each class j, the squared hinge loss for a single training example is $L_i = \sum_{j \neq y_i} (\max(0, f_j - f_{y_i} + 1))^2$.

This loss penalizes the model when the predicted score for an incorrect class j is too close to or greater than the score for the true class y_i, with larger margin violations receiving a heavier penalty. This encourages the model to maximize the margin between the true class and incorrect ones.

For a dataset with N examples and true labels y_i (one-hot encoded), where the model produces predicted probabilities $\hat{p}_j$ for each class j, the cross-entropy loss for a single example is $L_i = -\sum_{j=1}^{K} y_{i,j} \log(\hat{p}_{i,j})$.

This loss quantifies the difference between the true and predicted class distributions, penalizing incorrect predictions more heavily, especially when the predicted probability for the true class is low.

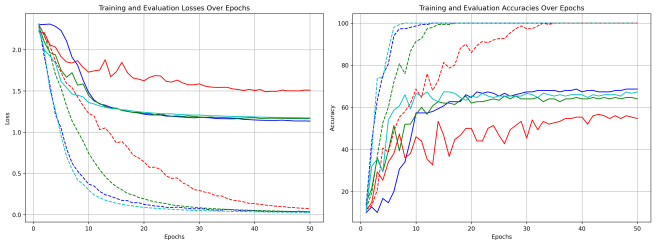

Figure 4: a) Training and Evaluation loss with different optimizers b)Training and Evaluation accuracy with different optimizers. The dashed lines show training results while solid lines represent evaluation. Blue-SGD, Green-SGD-momentum, Red-Adam, Cyan-RMSprop