MNIST Digit Classification

Model Performance Improvement Report

Introduction

MNIST is a famously used dataset for hand-written digit classification. It includes 70000 images of hand-written digits. The size of these images is 28 X 28 pixels. The training set consists of 60000 images and the test set consists of 10000 images. There are 10 classes of digits for all of these images. This report represents a comprehensive analysis of various methods for improving the accuracy of the baseline neural network model that was built for classifying the handwritten digits from the MNIST dataset. I have mainly used tensorflow and keras for completing tasks and experiments. The experimental procedures mainly include increasing the size and depth of the inner layers, experimenting with different activation functions in the inner layers, combining the activation function choices with different network size and depth, experimenting with different optimizers and learning rates, and finally performing the previous tests with various batch sizes and epochs. The main goal is to increase and improve the performance of the classifiers for different cases and evaluating the performance of the classifiers correctly for those cases.

Experimental Framework

Model Architecture

I. Baseline Model Architecture

The baseline model includes convolutional layers, max pooling layers, flatten layers and dense layers. According to the instruction, a neural network with atleast one convolutional layer and one or more hidden layers is constructed.

Specifically, for the architecture described, the hidden layers are:

Conv2D Layer: The first layer after the input, which applies 32 convolutional filters of size (3, 3) to the input images. Although it's the first layer and directly processes

the input, in the context of the entire network, it acts as a hidden layer by extracting spatial features from the input images through trainable filters.

MaxPooling2D Layer: Follows the Conv2D layer and performs max pooling with a (2, 2) window. This layer reduces the spatial dimensions (height and width) of the feature maps outputted by the Conv2D layer, effectively downsampling the feature maps and making the representation smaller and more manageable.

Flatten Layer: This layer doesn't perform any learning itself but transforms the 2D feature maps from the preceding max pooling layer into a 1D tensor. This is necessary because the following dense layers require 1D input. While primarily a structural layer to enable transitioning from convolutional layers to dense layers, it's considered a part of the hidden structure.

First Dense Layer: A fully connected layer with 100 neurons, receiving input from the Flatten layer. Each neuron in this layer is connected to all neurons in the previous layer. It applies a ReLU activation function to introduce non-linearity, enabling the network to learn complex patterns.

These layers are termed "hidden" because they do not directly interact with the input or output but instead work on representations derived from the input. These representations are transformed and passed through the network until reaching the output layer.

The output layer in this model is the:

Second Dense Layer: A fully connected layer with 10 neurons (one for each class, assuming a 10-class classification problem) that uses a softmax activation function. This layer outputs the probability distribution over the 10 classes for each input image, which is not considered a hidden layer because it directly produces the model's output.

So, the hidden layers of the baseline model are the first Conv2D layer, the MaxPooling2D layer, the Flatten layer, and the first Dense layer.

Performance of Baseline Model:

The test accuracy is 0.9875 for this baseline model. The precision score and the recall score are 0.988 and 0.988. The precision and recall score are quite high. Both of

these scores are 0.988. This indicates the model performs very well in identifying the digits from the MNIST dataset accurately.

Precision measures the ratio of correctly predicted positive observations to the total predicted positives. A precision of 0.988 means that when the model predicts a certain digit, it is correct 98.8% of the time.

Recall, on the other hand, measures the ratio of correctly predicted positive observations in the actual class. A recall of 0.988 in this context means that the model successfully identifies 98.8% of all actual digits correctly.

Having both high precision and recall suggests that the model is both highly accurate and covers the majority of the relevant cases without many false positives or false negatives.

Experiments to

Experiment Categories

1. **Layer Size and Depth**
2. **Activation Functions**
3. **Optimizers and Learning Rates**
4. **Batch Sizes and Epochs**
5. **Exclusion of CNN layer**

Hyperparameters Explored

Activation Functions: gelu, elu, relu, sigmoid, tanh, softplus, exponential, hard_sigmoid, linear, softsign

Layer Configurations: (1,1), (2,2), (3,3)

Neuron Counts: 64, 128, 256

Optimizers: Adam, AdamW, Adam, Adadelta, RMSprop, Nadam, Adadelta

Batch Sizes: 64, 128, 256, 16

Epochs: 5, 15, 20

Learning Rates: 0.1, 0.01, 0.001

Observation and Outcomes

1. Impacts of Improving the Size and Depth of the Model Architecture

It is possible to improve the model architecture by increasing the size and depth of the inner layers of the model. If we add more convolutional layers or dense layers to make the model deeper, each additional layer can extract more complex features from the dataset. If we increase the number of neurons in the dense layers or the number of filters in the convolutional layers, our model learns complex representations of the dataset. The test accuracy for the improved model is 0.99. Hence, the accuracy increases as we add more convolutional layers and more neurons to the network. Therefore, the test accuracy increases as the width and the depth of the network grows. From the confusion matrix, we calculated that the number of the total true positive classifications is 9920 and the number of total misclassifications is 80. Moreover, the aggregate sum of the main diagonal is 9920. So, the model achieves a high level of model accuracy.

The recall score is 0.99 and the precision score is 0.99. Therefore, the model correctly classifies positive classifications 99% of the time from the total positive cases. It can also classify actual positive classifications 99% of the time from total actual positive cases.

Therefore, the precision and the recall score increase if we add more layers and increase the depth and width of the network.

Hence, a more complex model characterized by a greater depth and breadth, has the potential to enhance the model´s predictive power, possibly due to its ability to capture more intricate patterns in the data.

Generally, increasing a model´s complexity does not invariably lead to improved performance and can result in overfitting, where model performs well on the training data and performs poorly on testing dataset.

2. Impacts of Different Activation Functions in the Inner Layers

In this experiment, we have combined different activation functions in the baseline model. These are gelu, elu, relu, sigmoid, tanh, softplus, exponential, softmax, softsign, hard_sigmoid, and linear. The experiments revealed a range of outcomes, with most activation functions achieving high precision and recall scores, indicating strong model performance.

Gelu: Precision: 0.9890 and Recall: 0.9890

Elu: Precision: 0.9873, Recall: 0.9873

Relu: Precision: 0.9880 , Recall: 0.9880

Sigmoid: Precision:0.9743 , Recall: 0.9743

Tanh: Precision: 0.9884 , Recall: 0.9884

Softplus: Precision: 0.9807, Recall: 0.9806

Exponential Activation: Precision: 0.0096, Recall: 0.0980, No classification

Softmax Activation: Precision: 0.4276, Recall: 0.4781

Softsign: Precision : 0.9849, Recall: 0.9849

Hard sigmoid: Precision: 0.9729, Recall: 0.9726

Linear: Precision: 0.9767, Recall: 0.9766

Analysis

Notably, GELU, ReLU, and Tanh functions led to the highest levels of precision and recall, closely followed by ELU and Softsign. Sigmoid and Hard Sigmoid, while still performing well, trailed slightly behind the leaders. Softplus showed a moderate decrease in performance. In contrast, Exponential Activation and Softmax Activation within the inner layers significantly underperformed, with the Exponential Activation function demonstrating almost no classification capability in this context.

High Performers: GELU, ReLU, and Tanh

GELU (Gaussian Error Linear Unit), **ReLU (Rectified Linear Unit)**, and **Tanh (Hyperbolic Tangent)** functions exhibited the highest precision and recall,

suggesting these functions' ability to maintain and propagate useful gradients during training, thus enhancing the model's learning capacity.

GELU and ReLU are known for their non-saturating nature, allowing models to learn faster and more effectively. Their performance underscores their suitability for deep neural networks by mitigating the vanishing gradient problem.

Tanh's performance, closely mirroring that of GELU and ReLU, suggests that in certain architectures, its symmetric output range can effectively contribute to model convergence and classification accuracy.

Moderate Performers: Sigmoid, Hard Sigmoid, Linear and Softplus

Sigmoid and Hard Sigmoid showed respectable, albeit slightly lower, performance levels. Their tendency to saturate at the extremes of their output range likely impeded the model's learning efficiency, leading to a minor reduction in classification precision and recall.

Softplus, a smooth approximation of the ReLU function, demonstrated reduced effectiveness compared to its non-smooth counterpart. This might be attributed to its less aggressive activation for positive inputs, possibly dampening the model's ability to capitalize on strong signal features.

The **Linear** activation's effectiveness could be attributed to the specific characteristics of the dataset and the problem. For datasets where the relationships between features and classes are inherently linear or close to linear, the linear activation function can directly model these relationships without the need for non-linear transformations.

Underperformers: Exponential Activation and Softmax Activation

The **Exponential Activation** function's drastic underperformance is attributed to its rapid escalation for positive inputs, likely exacerbating the exploding gradient problem and destabilizing the learning process.

The use of **Softmax Activation** in inner layers, typically reserved for output layers in multi-class classification scenarios due to its probabilistic output, appears to confuse the training process when used internally. This misapplication significantly impairs model accuracy.

ELU (Exponential Linear Unit) and **Softsign** provide noteworthy alternatives, with performances approaching that of the top-performing functions. ELU, with its positive mean, helps combat the vanishing gradient problem, while Softsign's bounded output range offers stability similar to Tanh.

3. Impacts of Combining the Activation Function Choices with Different Network Sizes and Depths

In this experiment, we have combined different activation functions, such as relu, sigmoid, tanh, SoftMax, and exponential with different network sizes and depths. We have changed the number of layers and neurons of convolution and dense layers to evaluate the model performance in different scenarios.

Activation	Layers	Precision	Recall
relu	(1,1)	0.9880	0.9880
relu	(1,1)	0.9890	0.9890
relu	(1,1)	0.9886	0.9886
relu	(2,2)	0.9901	0.9901
relu	(2,2)	0.9933	0.9933
relu	(2,2)	0.9925	0.9925
relu	(3,3)	0.9859	0.9858
relu	(3,3)	0.9845	0.9844
relu	(3,3)	0.9847	0.9846
sigmoid	(1,1)	0.9740	0.9736
sigmoid	(1,1)	0.9732	0.9728
sigmoid	(1,1)	0.9785	0.9783

sigmoid	(2,2)	0.9831	0.9830
sigmoid	(2,2)	0.9833	0.9832
sigmoid	(2,2)	0.9819	0.9818
sigmoid	(3,3)	0.0129	0.1135
sigmoid	(3,3)	0.0635	0.1831
sigmoid	(3,3)	0.0129	0.1135
tanh	(1,1)	0.9867	0.9867
tanh	(1,1)	0.9873	0.9873
tanh	(1,1)	0.9872	0.9872
tanh	(2,2)	0.9909	0.9909
tanh	(2,2)	0.9904	0.9904
tanh	(2,2)	0.9913	0.9913
tanh	(3,3)	0.9808	0.9807
tanh	(3,3)	0.9791	0.9791
tanh	(3,3)	0.9792	0.9789
Softmax	(1,1)	0.7582	0.7451
Softmax	(1,1)	0.7052	0.7029
Softmax	(1,1)	0.5055	0.4956
Softmax	(2,2)	0.0129	0.1135
Softmax	(2,2)	0.0129	0.1135
Softmax	(2,2)	0.0129	0.1135
Softmax	(3,3)	0.0129	0.0129
Softmax	(3,3)	0.0129	0.1135
softmax	(3,3)	0.0129	0.1135
exponential	(1,1)	0.0096	0.0980
exponential	(1,1)	0.0096	0.0980
exponential	(1,1)	0.0096	0.0980
exponential	(2,2)	0.0096	0.0980
exponential	(2,2)	0.0096	0.0980
exponential	(2,2)	0.0096	0.0980
exponential	(3,3)	0.0096	0.0980
exponential	(3,3)	0.0096	0.0980
exponential	(3,3)	0.0096	0.098

Key Observations:

ReLU Performance:

Exhibited high precision and recall across all network depths (1, 2, and 3 layers), with peak performance in deeper networks, indicating robustness and efficiency in learning complex patterns.

Sigmoid Activation:

Showed relatively lower precision and recall compared to ReLU, especially in shallower networks, but maintained consistent performance as network depth increased. This suggests a degree of saturation and slower learning, which is characteristic of sigmoid functions.

Tanh Activation:

Demonstrated competitive precision and recall, closely rivaling and sometimes surpassing ReLU, particularly in deeper networks. This highlights Tanh's capability in capturing and processing information efficiently, possibly due to its symmetric output range.

Softmax Activation:

Resulted in significantly lower precision and recall in most configurations, with a notable decrease in networks beyond a single layer. This is an expected outcome since softmax is primarily suited for the output layer in classification tasks, not as an activation function in inner layers.

Exponential Activation:

Yielded the lowest precision and recall, indicating poor performance across all configurations. The exponential function likely led to exploding gradients, severely hindering the learning process.

Variation with Network Depth:

Increasing the network depth generally improved performance for ReLU and Tanh activations but had mixed impacts on Sigmoid and drastically negative effects on Softmax and Exponential functions. The poor performance of exponential activation

across all depths underscores its unsuitability for internal layers due to instability in gradient propagation.

Analysis:

The **ReLU and Tanh** activations emerge as the most effective across varying network depths, highlighting their versatility and capacity for deep learning tasks.

The **Sigmoid** function's moderate performance might limit its application to specific contexts where other benefits (e.g., output range) outweigh the drawbacks.

The use of **Softmax** in inner layers is evidently inappropriate, aligning with its conventional role in the output layer for multi-class classification problems.

The **Exponential** activation's universal failure across experiments starkly illustrates the challenges of managing gradient stability with this function.

This experiment reinforces the critical role of activation function selection in neural network design. **ReLU** and **Tanh** stand out for their robust performance, especially in deeper networks, affirming their widespread adoption in practice. Conversely, **Softmax** and **Exponential** activations demonstrate significant limitations when misplaced in network architectures. These findings contribute to a nuanced understanding of activation functions' impacts, guiding practitioners in architecting more effective neural models.

4. Impacts of Various Optimizers and Learning Rates

In this experiment, we have combined various optimizer choices such as adam, nadam, adamw, adadelta, and rmsprop. The learning rates are 0.1,0.01,0.001,0.001. We have used the baseline model that we have built earlier and we have combined the choices in the model.

Optimizer	Learning Rate	Precision	Recall
adam	0.1	0.0102	0.1010
adam	0.01	0.9811	0.9811

adam	0.001	0.9876	0.9876
adam	0.0001	0.9858	0.9857
nadam	0.1	0.0095	0.0974
nadam	0.01	0.9795	0.9792
nadam	0.001	0.9863	0.9862
nadam	0.0001	0.9852	0.9852
adamw	0.1	0.0096	0.0980
adamw	0.01	0.9454	0.9415
adamw	0.001	0.9210	0.9178
adamw	0.0001	0.5242	0.6118
adadelta	0.1	0.9846	0.9846
adadelta	0.01	0.9616	0.9615
adadelta	0.001	0.9143	0.9141
adadelta	0.0001	0.7240	0.7082
rmsprop	0.1	0.0106	0.1028
rmsprop	0.01	0.0095	0.0974
rmsprop	0.001	0.9834	0.9833
rmsprop	0.0001	0.9881	0.9881

By analyzing precision and recall as the primary metrics, we can derive several key insights regarding the interplay between these factors:

Observations from the Data:

Impact of Learning Rate:

A learning rate of 0.001 tends to yield optimal results across different optimizers, striking a balance between convergence speed and stability.

Higher learning rates generate lower performance values when we use adadelta and rmsprop optimizers in the model. Conversely, if we decrease the learning rates, then adam optimizer shows high classification performance of the model.

Adamw shows poor performance when the learning rate is too low and too high. When we gradually decrease the learning rate, the performance sharply increases and decreases further.

Optimizer Performance:

Adam and Nadam optimizers consistently provide high precision and recall across most configurations, suggesting their effectiveness in adapting the learning rate during training.

AdamW also shows promising results, particularly at lower learning rates, indicating its potential in regularization and preventing overfitting.

Adadelta and RMSprop exhibit variable performance, with instances of very low precision and recall at certain learning rates, pointing to their sensitivity to learning rate settings and possible difficulties in navigating the loss landscape.

Learning Rate Optimization: Fine-tuning the learning rate is essential, with lower rates (e.g., 0.001, 0.0001) often yielding better performance, particularly in combination with adaptive optimizers like Adam and Nadam.

Optimizer Efficacy: The choice of optimizer significantly affects model outcomes, where Adam, Nadam, and AdamW emerge as versatile and effective across a broad spectrum of configurations.

5. Impact of Experimenting with Various Batch Sizes and Epochs

In this experiment, we execute all the above experiments with various batch sizes and epochs. Specifically, the experiment was designed to assess the model's behavior across an array of batch sizes—64, 128, 256, and 16—and epochs—5, 15, and 20. This methodical approach enabled a crystal-clear understanding of how these critical hyperparameters impact the learning dynamics and overall efficacy of the neural network in image classification tasks.

a. Increasing the Size and Depth of the Inner Layers with various Batch Sizes and Epochs

In this experiment, we evaluated the model performance after enhancing the model by increasing the number of convolutional layers and the number of neurons in the dense layers. We also increase the dense layers.

Furthermore, we combined this investigation with various batch sizes and epochs.

Observations from the Experiments

Impact of Increasing Batch Sizes and Epochs on the Model Performance

Batch Size	Epoch	Precision	Recall
64	5	0.9906	0.9906
64	15	0.9931	0.9931
64	20	0.9934	0.9934
128	5	0.9901	0.9901
128	15	0.9916	0.9916
128	20	0.9917	0.9917
256	5	0.9880	0.9879
256	15	0.9880	0.9879
256	20	0.9915	0.9915
16	5	0.9886	0.9884
16	15	0.9928	0.9928
16	20	0.9935	0.9935

Outcomes and Observations

When we increase batch size, and the epoch remains constant, the precision and recall decrease. In this case, the performance of the model decreases. When we

increase epochs, and the batch size remains constant then the performance of the model increases. When the batch size is low, and the epoch is high we get the highest performance which is 0.9935 accuracy and precision score. When the batch size is 16 and the epoch is 20, we get the highest performance of the classification model.

b. Experimenting with Different Activation Functions in the Inner Layers with various Batch Sizes and Epochs

In this experiment, we selected a systematic approach for selecting the activations functions in the inner layers with various batch sizes and epochs. A systematic variation in the activation functions lead to different performance outcomes for the model. We selected relu, sigmoid, softmax, exponential, and hard sigmoid functions and combined these activation functions with various batch sizes and epochs.

Activation	Batch Size	Epoch	Precision	Recall
Relu	64	5	0.9851	0.9850
Relu	64	15	0.9865	0.9865
Relu	64	20	0.9879	0.9879
Relu	128	5	0.9842	0.9842
Relu	128	15	0.9882	0.9882
Relu	128	20	0.9867	0.9867
Relu	256	5	0.9767	0.9765
Relu	256	15	0.9829	0.9828
Relu	256	20	0.9861	0.9861
Relu	16	5	0.9861	0.9861
Relu	16	15	0.9883	0.9883
Relu	16	20	0.9887	0.9887
Sigmoid	64	5	0.9466	0.9447
Sigmoid	64	15	0.9739	0.9739

Sigmoid	64	20	0.9761	0.9760
Sigmoid	128	5	0.9296	0.9286
Sigmoid	128	15	0.9606	0.9600
Sigmoid	128	20	0.9675	0.9673
Sigmoid	256	5	0.9116	0.9099

Activation	Batch	Epoch	Precision	Recall
Sigmoid	256	15	0.9423	0.9421
Sigmoid	256	20	0.9510	0.9507
Sigmoid	16	5	0.9706	0.9705
Sigmoid	16	15	0.9841	0.9841
Sigmoid	16	20	0.9844	0.9843
Softmax	64	5	0.0129	0.1135
Softmax	64	15	0.6286	0.6384
Softmax	64	20	0.7079	0.7078
Softmax	256	5	0.0129	0.1135
Softmax	256	15	0.0129	0.1135
Softmax	256	20	0.0129	0.1135
Softmax	128	5	0.0129	0.1135
Softmax	128	15	0.2820	0.3402
Softmax	128	20	0.2636	0.3447
Softmax	16	5	0.6895	0.6888
Softmax	16	15	0.7407	0.7402
Softmax	16	20	0.9193	0.9195
Exponential	64	5	0.0096	0.0980
Exponential	64	15	0.0096	0.0980
Exponential	64	20	0.0096	0.0980
Exponential	128	5	0.0096	0.0980
Exponential	128	15	0.0096	0.0980
Exponential	128	20	0.0096	0.0980
Exponential	256	5	0.0096	0.0980
Exponential	256	15	0.0096	0.0980
Exponential	256	20	0.0096	0.0980
Exponential	16	5	0.0096	0.0980
Exponential	16	15	0.0096	0.0980
Exponential	16	20	0.0096	0.0980

Hard Sigmoid	64	20	0.9740	0.9739
Hard Sigmoid	64	5	0.9432	0.9426
Hard Sigmoid	64	15	0.9704	0.9700
Hard Sigmoid	128	20	0.9650	0.9649
Hard Sigmoid	128	5	0.9247	0.9246
Hard Sigmoid	128	15	0.9597	0.9596
Hard Sigmoid	256	20	0.9418	0.9413
Hard Sigmoid	256	5	0.9086	0.9080
Hard Sigmoid	256	15	0.9385	0.9381
Hard Sigmoid	16	20	0.9855	0.9855
Hard Sigmoid	16	5	0.9728	0.9725
Hard sigmoid	16	15	0.9828	0.9828

Outcomes

From the above table, it is evident that softmax and exponential function have extremely low recall and precision scores for various batch sizes and epochs. In this case, the model fails to classify correctly.

Observations:

ReLU Activation: Exhibits strong and consistent performance across a range of batch sizes and epochs, with precision and recall metrics generally above 0.98, signifying excellent model accuracy. The highest performance is noted when the batch size is small (16) and epochs are high (20), indicating an optimal combination for this activation function.

Sigmoid Activation: Shows good performance, but with a slight decrease compared to ReLU. Precision and recall are generally in the high 0.90s but drop notably when batch sizes increase to 256. This activation function seems less robust against changes in batch size and epochs, suggesting a potential sensitivity to training regimen.

Softmax Activation: Demonstrates significantly lower precision and recall compared to ReLU and Sigmoid, particularly when used in internal layers. This is expected as Softmax is typically used in the output layer for classification tasks due to its probabilistic output. The decrease in performance is drastic with larger batch sizes, highlighting its inadequacy for internal layer activation in the tested configurations.

Exponential Activation: Yields the poorest performance, with precision and recall metrics significantly lower than other activation functions, often near 0.01, indicating nearly no classification capability. This result underscores the challenges with the exponential function, likely due to its contribution to exploding gradients.

Hard Sigmoid Activation: Presents varied results, with some configurations achieving precision and recall metrics in the high 0.90s, yet performance deteriorates in specific settings (notably with certain batch sizes and epochs). This suggests that while Hard Sigmoid can perform well, it may require careful tuning of batch sizes and epochs for optimal performance.

c. Experimenting with Various Activation Function Choices with Different Network, Depth, Epoch, and Batch Sizes

In this experiment, we combined various activation function choices with different network size, depth, epoch and batch sizes. We meticulously selected various activation functions such as relu, sigmoid, and softmax functions. Specifically, the experiment was designed to assess the model's behavior across an array of batch sizes—64, 128, 256—and epochs—5, 15, and 20.

In this experiment, softmax activation function generated the lowest precision and recall values despite the increment of the depth and width of the network. In this case, it showed the worst value when the batch size increased. **Conversely, Relu activation function generated the highest precision and recall score 0.9923 when the convolution and dense layers increase to (2,2) and (2,2). In this scenario, the batch size is 64 and the number of epochs is 20.** On the other hand, sigmoid function generated consistently good precision and recall scores, but it generated lowest precision and recall scores when the number of layers and batch size increases.

Activation Function Variation: The choice of activation function significantly influences model performance, with **ReLU** consistently outperforming other activation functions across various batch sizes and epochs. This underscores ReLU's capability to mitigate the vanishing gradient problem, thereby facilitating better learning. **Sigmoid** functions exhibit competitive performance but may suffer from saturation issues in deeper networks or with inappropriate initialization, potentially leading to slower convergence or stagnation. **Softmax** functions in inner layers exhibit poorer performance, highlighting their unsuitability for internal activation due to exponential growth leading to instability or their inherent nature being more suited to output layers for probability distributions.

Impact of Batch Size and Epochs: Larger batch sizes generally lead to faster computation per epoch due to parallelization but may result in reduced model generalization capability. This is evident from the diminishing precision and recall scores as batch size increases, particularly noticeable in configurations using less robust activation functions like sigmoid. Increasing the number of epochs tends to improve model accuracy up to a certain point, beyond which gains diminish. This is indicative of the model's ability to learn more from the data over time, though it also raises the risk of overfitting if not coupled with appropriate regularization techniques.

Network Size and Depth: The experimentation indicates that increasing network depth and size can enhance model performance, particularly when utilizing activation functions like ReLU. This can be attributed to the ability of deeper networks to capture more complex patterns and relationships in the data. However, the benefits of increased size and depth are contingent upon adequate data and proper regularization to prevent overfitting. Moreover, computational costs and training time escalate with network size, emphasizing the need for a balanced approach.

d. Experimenting with Various Optimizers and Learning Rates for Different Batch Sizes and Epochs

In this experiment, we systematically combined different optimizers with different learning rates for different batch sizes and epochs. The optimizers are adam, nadam, and adamw. The learning rates are 0.1,0.01, and 0.001.

Key Observations:

Optimizer Sensitivity to Learning Rate: The optimizer's performance is highly sensitive to the learning rate. For instance, with an excessively high learning rate (e.g., 0.1), the model's accuracy significantly suffers, as evident from the adam optimizer experiments. This suggests that the optimizer overshoots the minima in the loss landscape, failing to converge to a solution that generalizes well.

Effect of Batch Size and Epochs:

Larger batch sizes and higher epochs do not necessarily translate to better model performance, especially when paired with suboptimal learning rates. For instance, larger batch sizes with a high learning rate of 0.1 consistently resulted in poor model accuracy and convergence issues, indicated by precision and recall scores close to random guessing. However, when the learning rate was adjusted to a more appropriate value (e.g., 0.01), the model performance improved dramatically, underscoring the importance of a balanced learning rate for stable training dynamics. When learning rate is set to 0.01, the precision and recall scores are 0.9811.

Model Performance Across Different Configurations:

It shows a stark contrast in model performance between experiments with a learning rate of 0.1 and 0.01. With a learning rate of 0.01, the model achieves high accuracy, precision, and recall across different batch sizes and epochs. This indicates that a lower learning rate allows the optimizer to make finer adjustments to the model weights, leading to better generalization on the validation set.

Precision and Recall Analysis:

In scenarios with a high learning rate of 0.1, the model failed to predict any samples correctly, as indicated by precision and recall scores close to 0. This is a clear indicator of model divergence, where the optimizer's step size is too large to find a meaningful direction for weight updates. Conversely, with a lower learning rate of 0.01, the model's precision and recall scores significantly improve, showcasing

effective learning and generalization. This reflects the optimizer's ability to fine-tune the model weights towards optimizing the loss function more efficiently.

Optimization Challenges: The experiments highlight the challenges in optimizer configuration, particularly in selecting an appropriate learning rate. An optimally chosen learning rate, coupled with a suitable batch size and epoch number, can lead to vastly improved model performance, as seen with the adam optimizer at a learning rate of 0.01.

6. Experimenting without any Convolutional Layers

In this experiment, we focused on neural network architectures excluding the convolutional layers, and we concentrated only on the flatten and dense layers. The SGD (Stochastic Gradient Descent) algorithm was employed, configured with a learning rate of 0.01 and a momentum of 0.9. The evaluation of model accuracy was conducted through precision and recall metrics. Additionally, the confusion matrix was plotted to provide a visual representation of the model's performance across different classes. This approach allowed for an assessment of the impact of excluding convolutional layers on the model's ability to classify images accurately. The test accuracy is 0.9780 and the recall and precision scores are 0.9780.

Therefore, the performance of the neural network deteriorates when the convolutional layers are excluded from the network.

Conclusion

The experimental results underscore the critical role of the hyperparameters in neural network training. They illustrate that while the choice of optimizer, batch size, and epochs are important, the learning rate's influence on model performance is paramount. Selecting an appropriate learning rate can mitigate issues of convergence and model divergence, leading to improved accuracy, precision, and recall. From the experiment, it is evident that relu activation function generated the highest accuracy, precision and recall scores when combined with other parameters in optimal ways. Moreover, it is evident that if we remove the convolutional layers from the model,

it leads to the worst performance of the model. This emphasizes the necessity of hyperparameter tuning in achieving optimal model performance.