Classification of CIFAR-100
An exploration towards optimal model architecture

[redacted]

***Abstract*—The CIFAR-100 is widely used as a benchmark dataset to assess the performance and robustness of deep learning models in fine-grained image classification tasks. This report presents different experiments undertaken with the aim of developing a neural network architecture capable of accurately classifying images into 100 distinct categories. We explore different architectural designs, training strategies, and regularisation techniques to improve classification performance. The methodology behind model selection, training, and evaluation is detailed and the results are analysed through quantitative metrics and visualisations. Through our testing, we were able to build an optimal architecture that was able to achieve a testing accuracy of 75% in classifying the CIFAR-100.**

I. Introduction

THIS paper presents the methodology and exploratory process followed to identify an optimal deep neural network architecture, able to achieve high precision in the CIFAR-100[1] fine classification task. The CIFAR-100 dataset comprises 60,000 RGB images, each measuring at 32×32 pixels, evenly distributed across 100 classes. Each class is composed of 500 images for training and 100 images for testing. The primary challenge in this dataset comes from the low resolution of the images, which limits the amount of available spatial information. Additionally, the relatively small number of training examples per class can impede effective learning and generalisation. In this paper, we describe the experimental procedures, architectures evaluated, and rationale behind our decisions to effectively address these challenges and produce an adequate classification performance.

II. Literature Review / Background

A. Previous Work

Notable previous work on the CIFAR-100 classification challenge includes studies such as *mixup*[2] and *Big Transfer*[3]. Both of these studies introduce methodologies that heavily focus on robust augmentation techniques. *Mixup*, in particular, relies on generating new training examples by taking convex combinations of pairs of original examples and their labels. This means that for two training examples (x_i, y_i) and (x_j, y_j), a new sample is created:

$$\tilde{x} = \lambda x_i + (1 - \lambda)x_j, \quad \tilde{y} = \lambda y_i + (1 - \lambda)y_j, \qquad (1)$$

where $\lambda \in [0, 1]$, and y_i & y_j are one-hot encoded layers. The value of λ is controlled by the *mixup* hyper-parameter α. This hyperparameter controls the strength of interpolation between feature-target pairs. Its value is tuned to the specific task.

Other studies have shown that traditional augmentation has also been a solid performer when it comes to the CIFAR-100 challenge. To put this to the test, both *mixup* and traditional augmentation techniques were compared head to head under the same network architecture.

III. Methodology

A. Dataset Preprocessing

The following is an illustrative pipeline of the preprocessing that was undertaken in the data loading pipeline: one-hot encoding, normalization, and then data splitting for the multi-classification tasks.

Tensorflow was selected as the framework of choice for the preprocessing, model architecture and training of the model. The imported labels were then converted to one-hot encoding, a method chosen for several reasons. Firstly, one-hot encoding prevents the introduction of unintended ordinarily into categorical data, ensuring the model does not misinterpret categorical features as ordered or hierarchical. Additionally, it enhances model performance by providing a more expressive representation of labels, thereby allowing the capture of complex relationships within the data. Finally, one-hot encoding ensures compatibility with numerical-based algorithms such as gradient descent, which forms the foundation of the Adam optimiser used in our experiments.

Though it is important to mention that there have been studies [4] which condemn the use of one hot encoding. The reasoning behind this stems from the additional computational resources needed to apply one hot encoding However, this was no a problem in this study.

To split the labels into one hot encoding the function below was used:

```
y_train = utils.to_categorical(y_train, 100)
y_test = utils.to_categorical(y_test, 100)
```

Since the CIFAR-100 dataset did not offer a validation dataset, one had to be created. The validation dataset is crucial as it is used for learning. During training, the model trains on the training dataset and the produces predictions which are compared against the validation dataset.

To created a validation dataset the training dataset was split into two further datasets using a 80/20 split.

B. Data Augmentation

Data augmentation involves artificial augmentation, increasing the size of the training set by generating realistic variants of each training instance. The process reduces the problem of overfitting, making it a regularization technique. The generated instances should be as realistic as possible: ideally, given an image from the augmented training set, a human should

not be able to tell whether it was augmented or not. Simply adding white noise will not help; the modifications should be learnable. The data augmentation techniques applied included **rotation**, **width and height shifting**, **shearing**, **zooming**, and **horizontal flipping**. After conducting several experiments, it was observed that using a lower rotation range, while maintaining all other augmentation parameters constant, resulted in improved model performance. **Tab.I**.

TABLE I: Testing Accuracy for CIFAR-100 with different rotation ranges (all other variables were kept constant).

Rotation Range	Testing Accuracy	Epochs
$\pm 45°$	0.4193	50
$\pm 25°$	0.4535	50
$\pm 15°$	0.4784	50
$\pm 10°$	0.5354	50

C. Design Methodology of the Convolutional Neural Network Architecture

First, a simple Convolutional Neural Network (CNN) architecture was constructed. The architecture may be seen here **Fig.1**. The first model served as a baseline prototype to quickly evaluate performance. By analysing initial results (whether it is overfitting or underfitting), it was then iteratively refined the architecture by adding or removing layers, adjusting regularization (e.g. L1, L2, dropout, data augmentation, Early Sopping, Batch Normalization) or tuning other hyperparameters. The convolutional layer is able to **detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels**. Compared to fully connected layers it can store fewer parameters, which reduces the memory requirements of the model and improves its statistical efficiency.

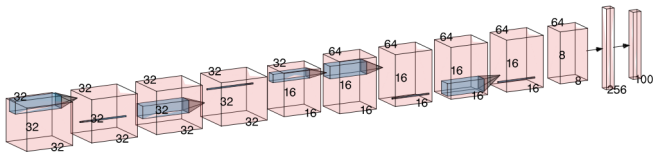

Fig. 1: The CNN model comprised of four convolutional layers: two with 32 output channels and two with 64. These were followed by two batch normalisation layers and one max pooling layer. After flattening, two dense layers with 256 and 100 units were added.

Upon testing the model, the resulting testing accuracy was 40.53%. The little complexity of the architecture was considered to be the main problem. Since there are 100 classes but only 4 convolution layers, there was a high chance that the model was falling to extract and learn important features. Therefore, a more complicated and well-structured model was considered.

The architecture of the second CNN model is shown in **Fig.2**. In the second iteration of the model the units of the output channel were increased from 128 to 256. Though due to pooling, the image size decreased from 8 x 8 to 4 x 4. Since, fewer operations were required to compute the output, there was a large improvement in computational efficiency, less time was required to train the model.

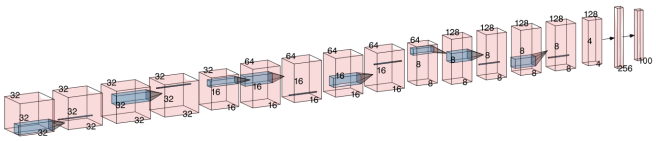

Fig. 2: Then the second tried version CNN model was constructed, in order to increase its complexity two more convolution layers was added in the size of 256 output channels output.

Looking at**Tab.II**, there can be seen a trend of increased performance with the increase of model depth and vice versa. When the number of layers is limited, the ability to classify coloured images is poor. Therefore, increasing the number of layers of the network model is an important part of improving the performance of the model [5]. Subsequently, a new question was posed: **whether the performance could be further enhanced by simply stacking additional layers?** 5 additional layers were added to form the 3rd version of CNN (CNN 3) and another 5 more layers to form the 4th (CNN 4). Concurrently, hyperparameter tuning operations such as dropout, L1, L2 regularization, and adjustments to the number of dense layers and the error rate (loss on validation compared with loss on accuracy) were undertaken to overcome performance bottleneck in model level. The selection and combination of hyperparameters were done by manual tuning. This resulted in the selection seen in **Tab.III**. The primary goal of manual hyperparameter search was to adjust the effective capacity of the model to match the complexity of the task. To reduce overfitting, dropout and L1_L2 regularisation were added in combination but did not significant impact on model accuracy. On the other hand, data augmentation significantly improved the model accuracy from 31% to 50% shown in the results section. Overall, the results indicated a saturation point beyond which increasing model complexity yielded minimal improvement in accuracy. Detailed analysis of the CNN architectures can be seen in **Tab.II**.

CNN 3 was selected as the optimal architecture from the ones stated in **Tab.II**. Considering the coptutational resources needed to train each model and the respective performance, CNN 3 had the highest testing accuracy at 52.91%.

1) Activation functions: For the CNN models mentioned above, ReLu was chosen as the preferred activation function for the convolutional layers. The reasoning can be seen below:

- Avoids saturation and vanishing gradient issues; prevents gradient dispersion.
- Promotes sparsity in activations.
- No complex exponential operations, leading to faster and more efficient computation.
- Converges significantly faster than Sigmoid or Tanh.

TABLE II: CNN Architectures Performance Comparison

Model	Trainable	Non-Trainable	Total	Dense Layers + Dropout	Neurons	Epochs	Acc. (%)
CNN 1 (32-64)	1,140,484	384	1,140,868	256-dropout(0.2)-100	356	30	40.53
	1,140,484	384	1,140,868	256-dropout(0.2)-100	356	50	46.80
CNN 2 (32-64-128)	838,148	896	839,044	256-dropout(0.2)-100	356	30	48.07
	838,148	896	839,044	256-dropout(0.2)-100	356	50	49.82
	2,962,180	896	2,963,076	1024-512-dropout(0.2)-100	1,636	50	50.58
CNN 3 (32-64-128-256)	1,462,276	1,920	1,464,196	256-dropout(0.2)-100	356	50	49.82
	2,326,276	1,920	2,328,196	1024-dropout(0.2)-100	1,124	50	52.17
	2,799,876	1,920	2,801,796	1024-512-dropout(0.2)-100	1,636	50	50.17
	2,799,876	1,920	2,801,796	1024-512-dropout(0.2)-100	1,636	50	50.99
	2,799,876	1,920	2,801,796	1024-dropout(0.2)-512-dropout(0.2)-100	1,636	50	50.61
	2,799,876	1,920	2,801,796	1024-dropout(0.3)-512-dropout(0.3)-100	1,636	50	51.08
CNN 4 (32-64-128-256-512)	5,817,604	3,968	5,821,572	1024-512-dropout(0.2)-100	1,636	50	50.59
	5,817,604	3,968	5,821,572	1024-512-dropout(0.2)-100	1,636	50	42.94

Note: All parameter counts are exact values. The last row in CNNs 4 the only variable is used by 'tahn' optimizer.

TABLE III: Model Hyperparameter Configurations

Activation	Dense Layer	Dropout	Reg. (L1/L2)	LR($\times 10^{-6}$)
ReLU	1024	0	0	1.00
Tanh	1024-512	0.2, 0.3	0.01, 0.001	1.00

Note: [x]Learning rates: LR are scaled by 10^{-6} baseline. Reg: Regularization

- More biologically plausible than Sigmoid-based activations.

The activation chosen for the last fully connect layer was the softmax activation. The softmax activation function converts logits into probabilities that sum to 1. The class with the highest probability (argmax) is chosen as the predicted output, making it suitable for multi-class classification with one-hot encoded labels. Thus, making a perfect fit for a multi-class problem.

2) Optimiser and Loss function: The approach specifed above employed an Adaptive Moment Estimation (Adam) optimizer to handle the overfitting problem, which was introduced by BN (Batch normalization) and DO(dropout) layers. By empirical theory, the loss function selected was the categorical cross-entropy (CCE). CCE measures the difference between two probability distributions; the predicted and the ground-truth distribution, which are represented by one-hot encoded vectors.

$$[h]L(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{2}$$

In a one-hot encoded vector, the correct class is represented as "1" and all other classes as "0." Categorical cross-entropy penalizes predictions based on how confident the model is about the correct class.

3) Pooling: Different activation functions, data augmentation, and network depth can have a significant impact on pooling layer performance[6]. Experimental results generally show that Max Pooling outperforms Average Pooling on the CIFAR-100 dataset [6]. Pooling is defined as a non-linear process that enables outputs at a particular location to be aggregated into a single value. This unique value is computed from the statistics of subsequent outputs, improving the accuracy and sensitivity of feature translation for smaller input data. Therefore, it improves efficiency by reducing computational usage. Additionally, it has also been showed to help with reducing overfitting[7][8].

4) Regularization: Another way in which overfitting can be reduced is through regularization. Regularization increases generalization and helps the model converge rapidly, reducing training time. Dropout, BatchNormalization, and L2 regularizers were applied in this model, as they were in CNN 3. A dropout layer with a rate of 0.2 was introduced to prevent overfitting by randomly deactivating input units during each training step.

D. Transfer Learning for Greater Accuracy

So far it was seen that training accuracy saturated at 52.91%. To yield better results, transfer learning [9] was opted as a viable technique to effectively improve the performance of the model and reduce computational resources. Transfer learning involved the use of pre-trained models. These models have been trained and extensively tested on big datasets, like 'ImageNet'. Additionally, since these networks are already trained, the number of training parameters reduces greatly, reducing the training time and saving computational resources. The time gained can be used further fine-tuning the model to achieve a greater accuracy. For the scope of this study, ResNet50 was selected for the feature extraction part of the model. The reasoning behind this stemmed from its wide popularity for the CIFAR-100 challenge [3], [10], [11].

1) Preprocessing: Using a pre-trained CNN entailed following a specific preprocessing technique for the input images. The technique varies from model to model, though due to Tensorflow, it has been possible to streamline it. To preprocess the input images for the ResNet50 the following function was used shown in the **appendix** below:

This function converted the input images from RGB to BGR and then zero-centred each colour channel with respect to the ImageNet dataset, without scaling.

2) Data Augmentation: As explained previously, image augmentation was performed to prevent overfitting and ultimately improve performance. For the implementation of transfer-learning *mixup* [2] was implemented. Compared to the previous implementation of augmentation through the *ImageDataGenerator*, *mixup* only has one hyper parameter, the alpha value α which determines the strength of the augmentation. To find the appropriate strength of augmentation, the model was run with different values of α (0,0.3,0.5 and 0.8) and the results were recorded. Additioanlly, the augmentation technique mentioned in was also implemented to asses its performance on the new model iteration.

3) Model Architecture: Setting up the architecture of the model followed a similar process as the method specified in the sections above. A sequential model was created whose input were images of size 32×32. In order to use the pre-trained model to its full potential the images had to be resized to 224×224, as the model was originally trained with images of that size on the ImageNet dataset.

To upscale the images to a size of 224×224 the Resizing layer was added. The Resizing layer offered the option to select between different interpolation modes. The bilinear option was selected due to being the only operational option at the time.

To test the performance effect of resizing the model's input images, the model was trained with and without the resizing layer. The output testing accuracy was then compared as seen in **TABLE IV**.

TABLE IV: Testing Accuracy for CIFAR-100 with and without image upscaling

Image Upscaling	Testing Accuracy	Epochs
No	0.4020	18
Yes	0.7310	23

Note: The epochs are indicative of the epoch whose weights were the best, not when training was stopped.

The ResNet50 model was imported as a layer, making sure to include only its backbone. A two dimensional average pooling layer was added next, to reduce the dimensions of the tensor so they can be fed to fully connected layers. Two fully connected layers were added. These performed the classification process. Lastly, a final fully connected layer was added (100 units) - same as the number of classes - with a softmax activation function.

To determine the optimal number of units for the two fully connected layers and the appropriate activation functions, Keras Tuner was used—a scalable framework for hyperparameter optimisation. It automates the training process by testing various combinations of parameters and activation functions across multiple runs. After each cycle, it records the validation accuracy and retains the configuration that achieves the highest performance.

By using Keras Tuner, it was found that the most optimal activation function for the two Dense layers was Tanh [5], with 1900 followed by 1600 units. The resulting network architecture can be seen in **TABLE V**

Based on empirical theory it was attempted to add dropout layers between the dense layers during the classification process but the model performance significantly reduced and thus they were removed. Moreover, it noted that if the rate of the dropout layer was increased passed 0.3, the validation accuracy would drop down to 50%, same as the validation accuracy before the implementation of transfer learning.

TABLE V: ResNet Model Archotecture

Layer	Shape	Parameters
Resizing	(128, 224, 224, 3)	0
ResNet50	(128, 7, 7, 2048)	23,587,712
GlobalAvgPool2D	(128, 2048)	0
Dense 1	(128, 1900)	3,893,100
Dense 2	(128, 1600)	3,041,600
Output	(128, 100)	160,100

Total parameters: 30,682,512. Trainable parameters: 7,094,800. Learning Rate = $1e-5$.

Adam was selected as the optimizer of choice. The reasoning behind this was stated previously in III-C2.

Callbacks such are *ReduceLROnPlatue* and *EarlyStopping* were used to make the training process more efficient and faster. The callback *ReduceLROnPlatue* was used to reduce the learning rate of the network during training. During training the model might get stuck on a local minima. By using *ReduceLROnPlatue* it was possible to monitor the convergence of the loss function and reduce the learning when it was identified that the convergence process has gotten *'stuck'*, thus giving the model the opportunity to escape local minima.

EarlyStopping works in a similar manner but its purpose it to prevent overfitting. By monitoring the convergence of the model it can stop the training when the model has started to overfit. Additionally it has the ability to restore the best weights of the network from the last best performing epoch, thus making sure that model will be able as accurate as possible.

IV. Results

A. Hyperparametes & Training Strategy

1) Results and analysis for CNN 3: The selected model with the best performance is shown here:

Based on experimental resutls it was found increasing the kernel size from 3×3 to 5×5 or 7×7 does not significantly improve performance. Given the computational trade-off, using larger kernels may not be worthwhile. Additionally, since the dataset images are 32×32 pixels, a 3×3 kernel is sufficient to localise and capture key features. **It was also observed that simply stacking additional layers or increasing the number of hyperparameters does not necessarily enhance performance.** This is likely due to limitations in the dataset's size and resolution, which restrict the benefits of added model complexity. The number of training epochs was increased from 30 to 50, resulting in improved performance, with accuracy

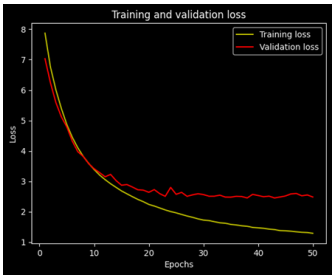

Fig. 3: Training and Validation Loss.

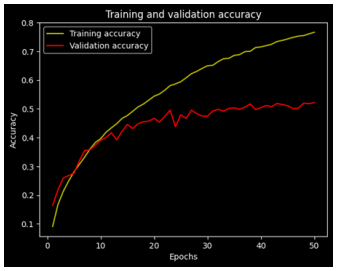

Fig. 4: Training and Validation Accuracy.

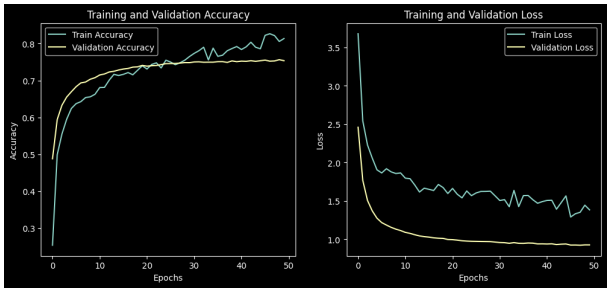

Fig. 5: Loss and Training Curves of *mixup* with $\alpha = 0.3$.

rising from a minimum of 40% to a maximum of 51%. In modern overparameterised models, training loss alone offers limited insight into generalisation performance and it is true for the CNN model proven by the results. As slight overfitting was observed, regularisation techniques were introduced, as shown in **Tab. II**. After multiple trials, dropout and L2 regularisation were applied; however, these techniques yielded only a modest improvement of approximately 2% in overall model performance.

2) Transfer Learning Model: Upon training the model specified in Table V a testing accuracy of 74.85% was achieved, the highest one so far, when copared to previous CNN iterations. To further validated the accuracy of the model, additional metrics were produced such as precision, recall and F1-score. These yielded values of 75.04%, 74.85% and 74.81%, respectively.

Then the transfer-learning network was trained with different amount of image augmentation in an aim to increase performance. Results can be seen below.

TABLE VI: ResNet Results (Augmentation)

Type	Accuracy	Precision	Recall	F1-Score
No augmentation	0.7484	0.7507	0.7485	0.7481
mixup: $\alpha = 0.3$	0.7515	0.7528	0.7515	0.7503
mixup: $\alpha = 0.5$	0.7504	0.7523	0.7504	0.7494
mixup: $\alpha = 0.8$	0.7443	0.7469	0.7443	0.7431
ImageDataGenerator	0.7324	0.7364	0.7324	0.7313

Where Accuracy is the testing accuracy.

From the results seen in Table VI it can be deduced that the best performance overall on the CIFAR-100 classification, was produced when transfer learning was implemented along with *mixup*, at $\alpha = 0.3$. The produced the highest testing accuracy along with the highest F1-Score, at 75.03%

B. Evaluation Metrics

To asses the model performance during training, metrics such as *Loss*, *Validation Loss*, *Accuracy* and *Validation Accuracy* were used. The *Accuracy* can be defined as:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{3}$$

Loss expression differs from process to process. Due to CIFAR-100 being a multiclass, one-hot encoded problem categorical cross-entropy loss was used. This is mentioned in III-C2

To further validated the good performance of the model, additional metrics were used to evaluate results after the model had finished training. These were recall, precision and F-1 score. Respective equations can be seen below.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{4}$$

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \tag{5}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

V. Discussion

A. Critical Evaluation of Results

The customised CNN model produced relatively lower accuracy compared to the transfer learning approach, due to the following reasons and limitations:

- CNN training dataset being small. Small amount of data will result in a high difference between training and testing accuracy and thus overfitting.
- It is evident that making adjustments to the hyperparameters, such as the dropout rate, regularisers, and kernel size, or augmenting the number of layers in the CNN with additional fully connected layers, will not result in a substantial enhancement of the objectives. This is due to the fact that the interplay between these parameters will have a significant impact on the objectives.

During the evaluation of the model with the transfer learning integration, under all the different types of augmentation, one bad performer prevailed. More specifically, it was identified that the class *'camel'* was always the worst performer, judging by its F1-Score. Its worse performance was recorded when the model was trained with no augmentation, scoring 0.4080 F1-Score. Even after implementing different types and strength of image augmentation, *'camel'* classification performance did not improve insignificancy as the never scored higher than 0.4556.

As a last evaluation, it is important to note that the transfer leanring model started to overfit when there was no augmenation applied, even though it was not captured by the performance metrics. This can be attributed to the testing dataset being quite similar to the training set. Even though the

testing dataset was not explicit seeing by the model, it was recorded under the exactly same conditions and environment as the training dataset. Thus the testing dataset that was used to evaluate the model does not represent a real case scenario were the model would be employed to classify different datasets.

B. Problems encountered

For this task, CIFAR-100 original images size can be resized into a larger size, for instance: 224x224 at first then applied to the CNNs model. The 8GB GPU RAM capacity resulted in the model experiencing bottleneck.

During the process of creating the model architecture for the ResNet50, different values for the *interpolation* were tested on the Resizing layer. Sadly, due to some unknown error, it was found that only the *bilinear* option was reproduced without the kernel crashing or any error being produced.

C. Future Work

In order to enhance the model's generalization capacity and overcome the limitations of the CNN 3 model, the application of the SAM optimiser is recommended [12]. Further experimentation and investigation are recommended in order to generate more data and optimize the augmentation process.

The current implementation of the model achieves an accuracy of 75% by transfer learning. The testing accuracy could potentially be improved through discriminative fine-tuning combined with gradual unfreezing [13]. Since different layers capture different types of information, fine-tuning strategies should reflect these distinctions.

In discriminative fine-tuning, layers of the model are trained with different learning rates. Typically, deeper layers (closer to the output) are fine-tuned using higher learning rates, while shallower layers (closer to the input) employ progressively smaller rates.

Gradual unfreezing involves incrementally unfreezing the model's layers, starting from the final layers that hold task-specific feature representations. Each newly unfrozen layer (or set of layers) is fine-tuned for one epoch before moving onto lower layers. This iterative process can continue until a predefined number of layers have been fine-tuned.

Combining discriminative fine-tuning with gradual unfreezing offers several benefits. Gradual unfreezing helps preserve previously learned representations, mitigating catastrophic forgetting. Additionally, adjusting fewer parameters at a time typically enhances training stability, preventing abrupt shifts during learning and allowing for smoother convergence. Therefore, by leveraging general knowledge obtained from standard training along with task-specific insights gained through fine-tuning, superior model performance can be achieved.

Overall, CNNs are the cornerstone of deep learning in the field of computer vision, but the core of deep learning is to choose the right infrastructure according to the task requirements. It is highly recommended to conduct further hyperparameter experimentation, to achieve higher accuracy and assess the model's potential for commercial application.

REFERENCES

[1] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, May 2012.

[2] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[3] A. Kolesnikov, L. Beyer, X. Zhai, *et al.*, "Big transfer (bit): General visual representation learning," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, Springer, 2020, pp. 491–507.

[4] C. Seger, *An investigation of categorical variable encoding techniques in machine learning: Binary versus one-hot and feature hashing*, 2018.

[5] W. Hao, W. Yizhou, L. Yaqin, and S. Zhili, "The role of activation function in cnn," in *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*, 2020, pp. 429–432. DOI: 10.1109/ITCA52113.2020.00096.

[6] A. Zafar, M. Aamir, N. Mohd Nawi, *et al.*, "A comparison of pooling methods for convolutional neural networks," *Applied Sciences*, vol. 12, no. 17, 2022, ISSN: 2076-3417. DOI: 10.3390/app12178643. [Online]. Available: https://www.mdpi.com/2076-3417/12/17/8643.

[7] Y. Matsuo, Y. LeCun, M. Sahani, *et al.*, "Deep learning, reinforcement learning, and world models," *Neural Networks*, vol. 152, pp. 267–275, 2022.

[8] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *Rough Sets and Knowledge Technology: 9th International Conference, RSKT 2014, Shanghai, China, October 24-26, 2014, Proceedings 9*, Springer, 2014, pp. 364–375.

[9] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[10] T. Ridnik, G. Sharir, A. Ben-Cohen, E. B. Baruch, and A. Noy, "Ml-decoder: Scalable and versatile classification head," *CoRR*, vol. abs/2111.12933, 2021. arXiv: 2111.12933. [Online]. Available: https://arxiv.org/abs/2111.12933.

[11] R. Wightman, H. Touvron, and H. Jégou, "Resnet strikes back: An improved training procedure in timm," *CoRR*, vol. abs/2110.00476, 2021. arXiv: 2110.00476. [Online]. Available: https://arxiv.org/abs/2110.00476.

[12] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," *arXiv preprint arXiv:2010.01412*, 2020.

[13] J. Howard and S. Ruder, *Universal language model fine-tuning for text classification*, 2018. arXiv: 1801.06146 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1801.06146.