CNNs for Image Classification

Samridhi Gupta
The University of Adelaide
samridhi.gupta@adelaide.edu.au

Abstract

Convolutional Neural Networks (CNNs) have transformed computer vision, driving progress in image recognition, object detection, and more. CNNs excel by learning hierarchical features from data, and various architectures have been developed to tackle unique challenges and applications in the field (Nabil 2023).

In this paper, we explore and evaluate Convolutional Neural Networks (CNNs) for image classification by experimenting with different architectures, including ResNet-18, AlexNet, and MobileNet. We aim to establish baseline performance for these models and apply hyperparameter tuning to optimize them for classifying images in the CIFAR-10 dataset. This systematic approach provides insights into CNN performance and highlights best practices for applying deep learning to image classification.

1. Introduction and background

A ConvNet's architecture resembles the neural connectivity patterns in the human brain and draws inspiration from the structure of the visual cortex. Neurons in this system respond exclusively to stimuli within a limited part of the visual field, called the receptive field. These fields overlap to collectively cover the entire visual space (Saha 2018).

Furthermore, the pooling layers in CNNs play an important part in reconstructing the feature dimensions, which improves computational speed and the ability to distinguish. CNNs are exceptionally good at extracting a plethora of features and generalizing the patterns learned in such a manner across fresh data (Mathworks n.d.).

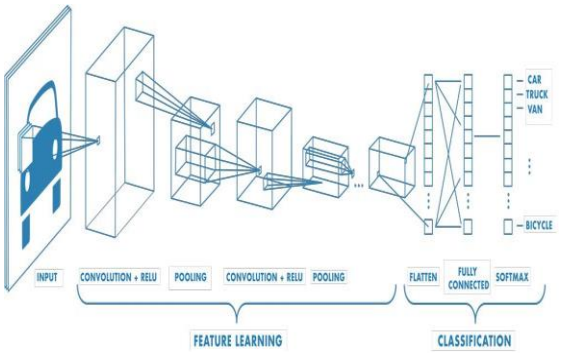

Fig 1. The architecture of a CNN

For this project, we focus on exploring and comparing the effectiveness of Convolutional Neural Networks (CNNs) in image classification tasks, specifically using the CIFAR-10 dataset, a widely used benchmark for assessing performance on visual learning problems. Given that image classification remains fundamental in fields such as autonomous driving, medical imaging, and visual search engines (IBM n.d.), selecting the right model architecture is crucial to achieving high accuracy and efficiency.

To establish a comprehensive understanding, we experiment with multiple CNN architectures, including ResNet-18, AlexNet, and MobileNet, each known for different strengths in handling complex image data. ResNet-18, a simplified version of the deeper ResNet models, incorporates residual connections to mitigate issues like vanishing gradients (Nabil 2023), allowing effective training on relatively shallow models. AlexNet, a pioneering CNN, laid the groundwork for modern deep learning, though its simpler structure contrasts with more advanced, efficient models (Nabil 2023). MobileNet, designed for lightweight, high-performance applications, offers a compact and computationally efficient approach, ideal for environments with limited processing power (Nabil 2023).

These three models together represent a variety of design philosophies in CNN development. They provide insights into traditional, residual, and efficiency-focused architectures, making them ideal for a comprehensive analysis of CNN performance on image classification tasks. Other architectures, like VGG or deeper ResNets, were not selected due to their higher computational requirements and lack of specific advantages for this dataset size and project scope.

This project involves training these models from scratch and conducting hyperparameter tuning to optimize performance for each architecture. By comparing baseline models and evaluating tuned versions, we identify which configurations, and architectural features yield the best accuracy and computational efficiency. This systematic analysis not only highlights the potential and limitations of different CNNs but also provides practical insights

into model selection for image classification tasks across varied applications.

2. Method description

The project systematically builds, trains, and evaluates multiple deep learning models on CIFAR-10, a dataset containing 60,000 color images across 10 classes. Starting with a foundational CNN model, the implementation advances to progressively more sophisticated architectures, including ResNet-18, AlexNet, and MobileNet, each designed from scratch to better capture complex patterns and improve classification accuracy.

The convolutional neural network is made of four main parts (Datacamp 2023).

- Convolutional layers
- Rectified Linear Unit (ReLU for short)
- Pooling layers
- Fully connected layers

CNNs are handy in image classification because they can extract relative hierarchies of images depicting edges, textures, and other shapes of images in subsequent layers (Mathworks n.d.). As compared to fully connected networks, CNNs make the use of convolutional layers, which work with small filters across the image in the same spatial dimensions, so that there are significantly fewer parameters and computational effort than fully connected ones.

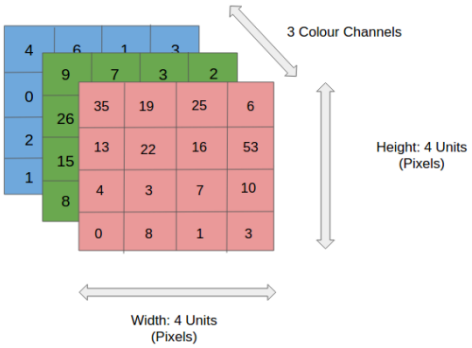

Fig 2. A 4x4x3 RGB image

The purpose of a CNN is to simplify images into a more manageable form, preserving essential features necessary for accurate predictions (Saha 2018).

2.1 Convolutional Layers

The convolution layer serves as the first essential component of a CNN, where a sliding window operation called convolution is applied to an image represented by a pixel matrix. This operation uses small grids, known as filters or kernels, which move over the image to detect specific patterns, such as edges, curves, or shapes. Multiple filters of the same size are applied within the layer, each focused on capturing a different visual feature, like the contours or overall shape of objects in the image. As these filters slide across the image, they create a new grid that highlights the locations of the detected patterns, forming a structured representation of the image's key characteristics (Datacamp 2023).

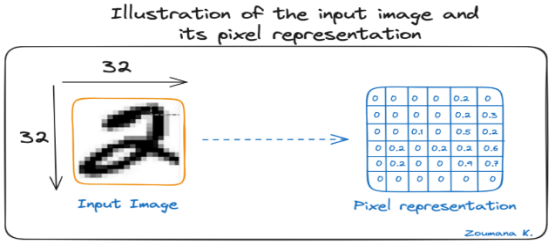

Fig 3. An input image and it's pixel presentation

2.2 ReLU function

A ReLU activation follows each convolution, allowing the network to capture non-linear feature relationships and reducing vanishing gradient issues, enhancing pattern recognition.

2.3 Pooling Layers

The pooling layer extracts key features from the convoluted matrix, using aggregation operations to reduce its dimensions. This process conserves memory during training and helps prevent overfitting (Datacamp 2023).

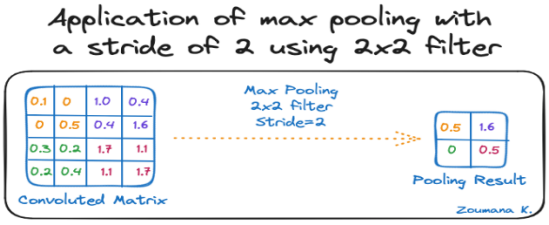

Fig 4. Max pooling demonstration

2.4 Fully connected layers

In the final layers of a CNN, inputs are taken from the flattened matrix produced by the last pooling layer, with ReLU activations applied for non-linearity. A softmax layer then generates probability values for each output label, with the highest probability indicating the predicted label (Datacamp 2023).

2.5 ResNet-18

ResNets enable deep neural networks to be trained effectively without the performance degradation commonly seen in other fully-connected networks (Azeem 2023).

2.5.1 Residual block:

Residual blocks are key to ResNets. Unlike traditional networks, where inputs pass through convolution and activation layers, ResNets add the block's input to its output, creating a residual connection (Azeem 2023).

This output is represented as H(x) = F(x) + x

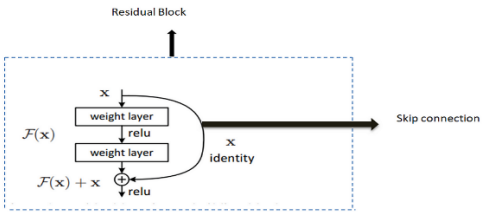

Fig 5. A residual block of ResNet

2.5.2 Skip connection:

A skip connection helps create residual blocks by bypassing the input of the residual block over the convolutional layer and adding it to the block's output (Azeem 2023).

2.5.3 Stacked layers:

ResNet architectures are built by stacking several residual blocks, allowing the network to reach considerable depth (Azeem 2023).

2.5.4 Global Average Pooling

ResNet architectures often use Global Average Pooling (GAP) as the final layer before the fully connected layer. GAP condenses each feature map to a single value, creating a compact summary of the feature map (Azeem 2023).

2.6 Alexnet

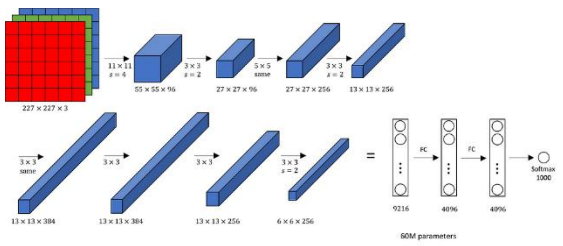

Fig 6. Architecture of AlexNet

AlexNet was the first architecture to leverage GPUs to enhance training performance. It comprises of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and a SoftMax layer. Each convolutional layer has a convolution filter with ReLU as the activation function. The pooling layers handle max-pooling, and the fixed input size is primarily due to the fully connected layers. AlexNet contains over 60 million parameters (Bangar 2022). It's key features are:

- ReLU activation function instead of tanh
- Batch size of 128
- Uses Stochastic Gradient Descent with Momentum for learning

2.7 MobileNet

MobileNet, developed by TensorFlow, is an efficient mobile-focused computer vision model. It employs depthwise separable convolutions to drastically reduce parameter count compared to traditional convolutional networks of similar depth, resulting in a lightweight model (Pujara 2023). Depthwise separable convolution consists of two main steps:

- Depthwise convolution
- Pointwise convolution

The key distinction between MobileNet and traditional CNN architectures is that MobileNet separates the convolution into two parts: a 3x3 depthwise convolution and a 1x1 pointwise convolution, whereas traditional CNNs typically use a single 3x3 convolution layer followed by batch normalization and a ReLU activation (Pujara 2023).

3. Method Implementation

Link:- https://github.com/samridhi20-hub/Deep-Learning_a1901641/blob/main/Samridhi_Gupta_CNN_Cifar10.ipynb

The implementation provides a comparative analysis across multiple architectures, highlighting the performance trade-offs between model complexity, computational cost, and accuracy.

3.1 Loading data and making it suitable for analysis

The first code snippet shows all the imported libraries that are needed to accomplish this task. After that, the next cell is responsible for extracting and loading the CIFAR-10 dataset into a Google Colab environment, preparing it for further analysis or model training. CIFAR-10, a dataset commonly used for image classification tasks, contains 60,000 32x32 color images across 10 classes, with 50,000 training images and 10,000 test images (Kaggle n.d.). Each class has an equal number of images.

The data is loaded from individual batch files, and organized into training and test datasets. After defining file paths, the dataset archive is decompressed. The code then uses a function to load each batch, reshaping and transposing image data for compatibility with deep learning models. The training data, stored across five separate files, is loaded and combined into unified arrays, while the test data is loaded separately. This prepares the CIFAR-10 data for efficient use in training and evaluating image classification models.

3.2 Data Pre-processing

Normalization scales pixel values to a consistent range, typically [0, 1], which helps the model learn more effectively by preventing large input values from overwhelming gradients and causing instability (Saxena 2024). CIFAR-10 contains small 32x32 images with pixel intensities ranging from 0 to 255. Without normalization, these values would be quite large, causing the model to learn slower or possibly suffer from numerical instability. Normalizing them to [0,1] improves model performance by ensuring that each feature contributes proportionately.

One-hot encoding transforms class labels into binary vectors, where each class is represented as a unique vector with a 1 at the index corresponding to the class and 0s elsewhere (Brownlee 2020). This format is compatible with categorical data, the standard loss function for multi-class classification, as it allows the model to calculate the probability distribution over multiple classes and learn which class best fits each input. CIFAR-10 has 10 distinct classes, making it a multi-class classification problem. Representing each label as a one-hot encoded vector enables the model to treat the classes as separate categories.

3.3 Building a CNN

Once the dataset is pre-processed, a convolutional neural network (CNN) model is built using Keras' Sequential API. This code defines and compiles a convolutional neural network (CNN) for classifying images in the CIFAR-10 dataset. The model consists of three convolutional layers (with 32, 64, and 128 filters), each followed by max-pooling to reduce spatial dimensions, enhancing feature extraction. After flattening, it includes a dense layer with 64 units, a dropout layer to prevent overfitting, and a final dense layer with 10 units and softmax activation for outputting probabilities across the 10 CIFAR-10 classes. The model is compiled with the Adam optimizer, categorical cross-entropy loss (suitable for multi-class classification), and accuracy as the evaluation metric. After training this model on train data, for 20 epochs in batches of size 32, an accuracy of **71.08%** is obtained on the test dataset.

To understand how well the model is fitting the data and whether it is overfitting or underfitting, two graphs are plotted. The Accuracy vs. Epochs graph suggests that the model is learning well on the training data. The validation accuracy improves in the early epochs and then stabilizes around the 10th epoch, showing that the model has learned enough patterns to generalize well on validation data without further significant improvement.

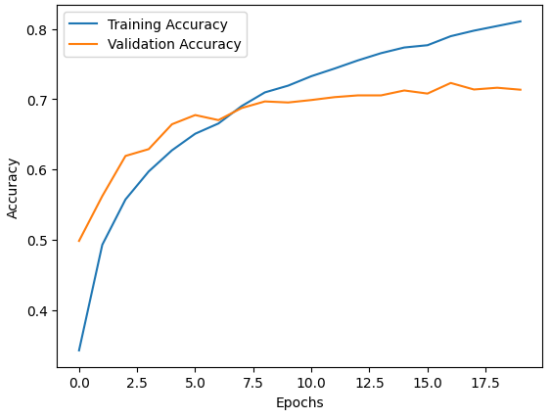

Fig 7. Accuracy vs. Epochs

In the Loss vs. Epochs graph, the training loss reduces significantly and improves at reducing the error on the training set. The validation loss also decreases, showing that the model is improving on unseen data. However, around the 8th to 10th epoch, it starts to stabilize and then slightly increases, which may indicate some overfitting as the model begins to perform slightly worse on the validation data.

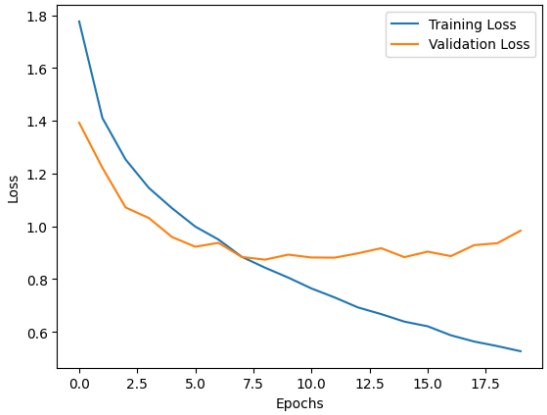

Fig 8. Loss vs. Epochs

3.4 Experimenting further with the CNN

The new model is more regularized with L2 regularization and batch normalization, potentially improving generalization and training speed compared to the previous CNN, which lacks these techniques. The main differences between this CNN and the previous one are:

L2 Regularization: The new model uses L2 regularization (kernel_regularizer=l2(0.001)) in the convolutional and dense layers. This helps reduce overfitting by penalizing large weights during training. The previous model does not have any regularization.

Batch Normalization: After each convolutional layer and its corresponding max-pooling layer, the new model includes a BatchNormalization() layer. This layer normalizes the activations, which can speed up training and help the model generalize better. The previous model does not include batch normalization.

Layer Size and Structure:

- **Convolutional Layers**: The new model uses more filters in the convolutional layers—64 and 128 filters compared to the 32, 64, and 128 filters in the previous model. This allows the model to learn more complex features.
- **Dense Layer Size**: The new model includes a dense layer with 128 units, whereas the previous model had 64 units. This increases the model's capacity to learn from the features extracted by the convolutional layers.

Dropout: Both models have a dropout layer, but in this model, it comes after the dense layer with 128 units. Dropout is used to prevent overfitting by randomly setting a fraction of input units to zero during training.

Furthermore, learning rate scheduling is introduced during training using the 'ReduceLROnPlateau' callback, which dynamically adjusts the learning rate based on the model's performance on the validation set. By reducing the learning rate by a factor of 2-10 once learning stagnates, models often benefit (Keras n.d.). Along with these changes, the new model is trained and tested, this time on 30 epochs (10 more than the last model). An accuracy of **77.75%** is obtained.

The model performs well on the training data, as evidenced by high training accuracy and low training loss. However, the validation accuracy plateaus and validation loss does not decrease further after some epochs, showing that the model is not generalizing well to new data, likely due to overfitting.

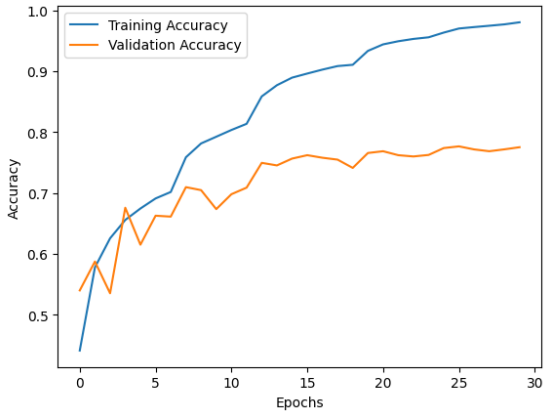

Fig 9. Accuracy Vs. Epochs for improved CNN

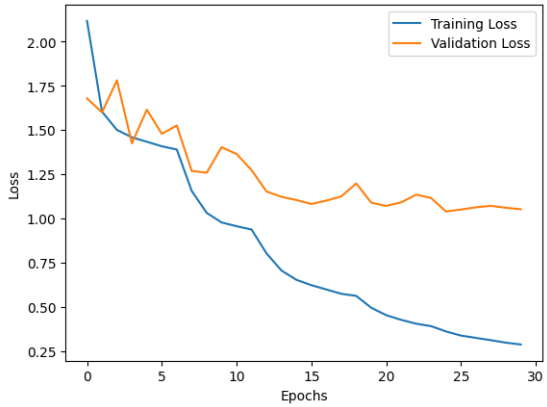

Fig 10. Loss vs. Epochs for the improved CNN

3.5 ResNet-18 architecture

As we saw from the CNN architecture, the model is overfitting. Hence, we explore new architectures. ResNet-18 is one of them. The 'resnet_block()' function defines the basic building block of ResNet, which includes two convolutional layers with batch normalization and ReLU activation, along with an optional downsampling step. The 'build_resnet_18()' function stacks these blocks in the typical ResNet-18 structure, starting with an initial convolutional and pooling layer, followed by residual blocks at increasing filter depths. Finally, a global average pooling layer and a dense output layer with 'softmax' activation are added to classify images into 10 classes. The model is compiled using the 'Adam optimizer', categorical cross-entropy loss, and accuracy as a performance metric.

Adam optimizer was chosen because Adam's default parameters (learning rate, beta1, and beta2) work well for many problems, reducing the time and effort needed for tuning. After training the

ResNet-18 model on training data and validating it for 20 epochs in batches of 64, an accuracy of **71.61%** was obtained on the test dataset.

3.6 AlexNet for CIFAR-10

Next, we use AlexNet for classification. This model consists of five convolutional layers that learn spatial features, followed by three max-pooling layers to reduce spatial dimensions. After the convolutional layers, the data is flattened and passed through two fully connected (dense) layers with dropout to reduce overfitting. The final layer uses softmax activation to classify images into one of the 10 classes in CIFAR-10. The model is compiled with the Adam optimizer and categorical cross-entropy loss for multi-class classification, with accuracy as the evaluation metric.

An accuracy of **71.36%** on the test dataset was obtained, which is almost the same as ResNet-18 model, after training the AlexNet model on training data and validating it for 20 epochs in batches of 64.

3.7 Designing MobileNet

The architecture starts with a regular convolutional layer followed by several MobileNet blocks using depthwise separable convolutions (by combining both depthwise and pointwise convolutions) to reduce the computational cost and number of parameters while still capturing features at different depths. The network gradually increases filters and reduces spatial dimensions, ending with a global average pooling layer and a fully connected layer with softmax activation for classification. The model is compiled with the Adam optimizer and categorical cross-entropy for multi-class classification, evaluating model performance with accuracy.

An accuracy of **68.31%** is obtained on the test dataset.

4. Experiments and analysis

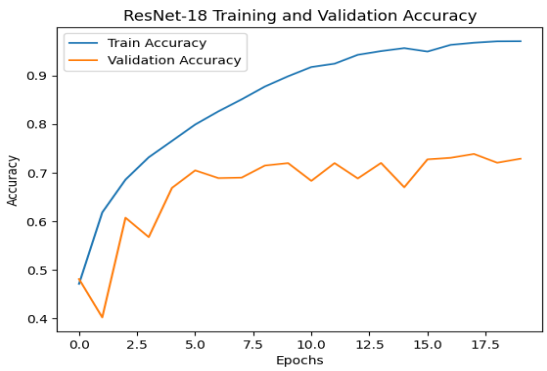

Fig 11. Accuracy vs. Epochs for ResNet-18

- ResNet-18

The training accuracy steadily increases, reaching close to 1.0 by the end of the 20 epochs. This shows that ResNet-18 fits the training data well. The validation accuracy increases initially but plateaus around 0.7, showing slight fluctuations. There is overfitting. The model achieves high accuracy on the training set, but the validation accuracy does not improve beyond a certain point.

- AlexNet

Similar to ResNet-18, the training accuracy increases over time and approaches 0.9 by the end. This indicates AlexNet fits well with the training data, though not as strongly as ResNet-18. The validation accuracy increases initially, then levels off around 0.7, with fewer fluctuations compared to ResNet-18. AlexNet overfits, but slightly less than ResNet-18 as the gap between training and validation accuracy is comparatively less.

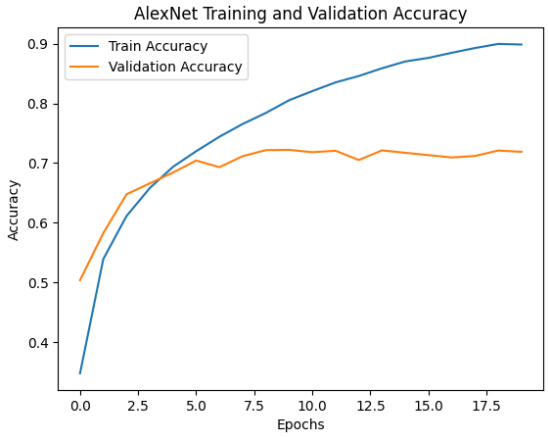

Fig 12. Accuracy vs. Epochs for AlexNet

- MobileNet

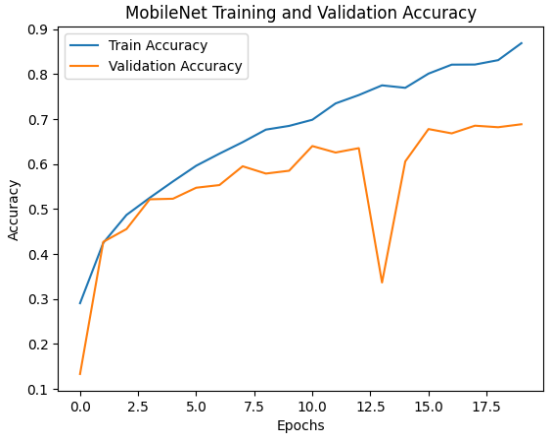

Fig 13. Accuracy vs. Epochs for MobileNet

MobileNet's training accuracy grows steadily but reaches only around 0.8 by the end of 20 epochs, which is lower than ResNet-18 and AlexNet. This could suggest that MobileNet is less prone to overfitting but may also have a lower capacity for fitting the data. The validation accuracy increases initially and stabilizes around 0.7 with some drops, and there is a sharp drop around the 12th and 14th epochs. These drops indicate potential instability in validation performance, which could result from fluctuations in model learning.

4.1 Hyperparameter Tuning

From the results obtained above, MobileNet seems like a good choice to go ahead with for the classification of CIFAR-10 images. To further refine it, I chose to perform hyperparameter tuning. To check how much the accuracy changes, I also performed hyperparameter tuning on ResNet-18 and AlexNet.

In each model, hyperparameters like learning rate, dropout rate, and certain architecture-specific settings (e.g., the width of MobileNet) are tuned to achieve better performance.

Hyperparameter tuning is performed on the ResNet-18, AlexNet and MobileNet models using Keras Tuner's 'RandomSearch' function. The goal is to find the best combination of hyperparameters to maximize the validation accuracy of the model.

The test accuracy of ResNet-18 after this was 71.38%, which was not significantly different from the previous accuracy. For AlexNet, the accuracy increased from 71.36% to 78.5%.

The tuned MobileNet model shows a test accuracy of 77.39%, which is a significant increase from its previous performance and a test loss of around 0.6784. A 77.39% accuracy indicates that the model correctly classifies about three-quarters of the test images. This accuracy is reasonable. A test loss of 0.6784 indicates that, on average, the error between the model's predicted probabilities and the actual class labels is moderate. Since the test loss is below 1.0, it shows that the model's predictions are closer to the target values than if they were random guesses. This also implies the model has achieved a reasonable level of confidence in its predictions, though it may still have some room for improvement in finer details.

Model Name	**Accuracy before hyperparameter tuning**	**Accuracy after hyperparameter tuning**
ResNet-18	71.61%	71.38%
AlexNet	71.36%	78.5%
MobileNet	68.31%	77.39%

Table 1. Results before and after Hyperparameter tuning

4.2 Summary of the results

Model name	**Accuracy**	**Final model**
CNN	71.08%	No
Improved CNN	77.05%	No
ResNet-18	71.38%	No
AlexNet	78.5%	No
MobileNet	77.39%	Yes

Table 2. Final accuracies of all models

5. Conclusion/further work

By a thorough experiment and analysis, I concluded that MobileNet is the best choice for classifying images in CIFAR-10. It significantly improves after hyperparameter tuning and the low-test loss shows that the model is fitting considerably well on the test dataset. On the other hand, other models show a lot more overfitting than MobileNet.

Further work on this project could involve experimenting with additional architectures like VGG-16 and VGG-19, or Vision Transformers (ViTs), which leverage self-attention mechanisms for image classification. Additionally, applying transfer learning with pretrained models on larger datasets like ImageNet could improve performance, especially for complex image features in CIFAR-10. Lastly, hyperparameter tuning could be expanded to include advanced techniques like Bayesian optimization to find optimal configurations, potentially leading to better generalization and reduced training times.

6. References

[1] Nabil, M 2023, *Unveiling the Diversity: A Comprehensive Guide to Types of CNN Architectures*, Medium, viewed 10 November 2024, <https://medium.com/@navarai/unveiling-the-diversity-a-comprehensive-guide-to-types-of-cnn-architectures-9d70da0b4521>

[2] Mathworks n.d., *What is a Convolutional Neural Network?*, Mathworks, viewed 10 November 2024, <https://au.mathworks.com/discovery/convolutional-neural-network.html>

[3] IBM n.d., *What are convolutional neural networks?*, IBM, mviewed 10 November 2024, < https://www.ibm.com/topics/convolutional-neural-networks >

[4] Datacamp 2023, *An introduction to Convolution Neural Networks*, Datacamp, viewed 10 November 2024, <https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns>

[5] Azeem, I 2023, *Understanding ResNet Architecture: A Deep Dive into Residual Neural Network*, Medium, viewed 10 November 2024, <https://medium.com/@ibtedaazeem/understanding-resnet-architecture-a-deep-dive-into-residual-neural-network-2c792e6537a9>

[6] Bangar, S 2022, *AlexNet Architecture Explained*, Medium, viewed 10 November 2024, <https://medium.com/@siddheshb008/alexnet-architecture-explained-b6240c528bd5>

[7] Pujara, A 2023, *Image classification with MobileNet*, builtin, viewed 10 November 2024, <https://builtin.com/machine-learning/mobilenet#:~:text=MobileNet%20is%20a%20computer%20vision,a%20lightweight%20deep%20neural%20network.>

[8] Saxena, S 2024, *Introduction to Batch Normalizaion*, Analytics Vidhya, viewed 10 November 2024, <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/#:~:text=Normalization%20is%20a%20data%20pre,values%20to%20a%20balanced%20scale.>

[9] Brownlee, J 2020, *Why One-Hot Encode Data in Machine Learning?*, Machine Learning Mastery, viewed 10 November 2024, <https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>

[10] Keras n.d., *ReduceLROnPlateau*, Keras, viewed 10 November 2024, <https://keras.io/api/callbacks/reduce_lr_on_plateau/>

[11] Saha, S 2018, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, Towards Data Science, viewed 10 November 2024, < https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>