Multiclassification of CIFAR-10: comparison of a custom built Convolutional Neural Network against ResNet-18, AlexNet and MobileNet

Anonymous CVPR submission

Paper ID *****

Abstract

Convolutional Neural Networks (CNNs) are the neural networks of choice when it comes to tasks such as image classification, a division of computer vision. There are various well-known architectures widely available on the internet that has achieved impressive performance across different datasets. This paper explores the use of CNNs in classifying images from the CIFAR-10 dataset which is a well-known dataset comprising 60,000 images with a resolution of 32x32 pixels used for training CNNs for image classification.

This will be done by first implementing a very simple baseline CNN model from scratch which will then be compared to three other existing state-of-the-art deep learning architectures: ResNet-18, AlexNet, and MobileNet. The three models were developed for deep learning and has made strides in their application with their architectural designs, some of which were specific to their use case. Some notable differences in architecture are the usage of residual blocks/connections, number of layers and filter sizes, and size of the input images.

Each of the 4 models will be trained from scratch on the CIFAR-10 dataset to preserve the fairness of test, and to eliminate any pretrained weights imported from the model. After the training and testing was conducted, the results show that the baseline model performed better than the 3 other imported models which suggests that for specific tasks, in this case being training a CNN to classify an image from a smaller sized dataset, a CNN that is custom-built for the task might outperform more complex, deeper networks which were originally developed for different use cases.

The underperformance is understandable if we take into consideration the size difference between CIFAR-10 and what the networks were originally trained on, the architectural complexity, and the different input image size. These factors tend to lead to overfitting as the complexity and depth of large-scale models differ from small-scale model. This paper therefore outlines highlight the importance of designing neural network architectures that align with specific task requirements and provides insight on the outcome of trying to apply large-scale models directly to small-scale tasks.

1. Introduction

Image classification is one of the most common uses of Convolutional Neural Networks (CNNs) as they are very well suited for the use case (LeCun, Bengio & Hinton 2015, pp. 436-444; Goodfellow, Bengio& Courville 2016). The CIFAR-10 dataset is chosen as the dataset for this paper. CIFAR-10 contains 60,000 32x32 pixel images across 10 different classes and is commonly used as a benchmark for evaluating image classification on CNNs (Krizhevsky & Hinton 2009; Geron 2019). This paper explores the performance of various CNN architectures on the CIFAR-10 dataset, starting with a baseline CNN model and comparing the baseline against three state-of-the-art architectures: ResNet-18, AlexNet, and MobileNet.

The goal of this paper is to illustrate the performance of these three imported model's accuracies on classifying low-resolution images, as the models were originally trained and tested on high-resolution datasets. The baseline model will be specifically designed for CIFAR-10's size and characteristics to achieve an initial baseline performance, while the other three models will be minimally modified to CIFAR-10's input size. This is required to ensure that the testing is done on a fair basis, since the three models were designed to accept a larger input size of 224x224 pixels (high-resolution) in contrast with CIFAR-10's 32x32 pixels (low-resolution).

Models will be systematically trained from scratch on the CIFAR-10 to maintain consistency and ensure that differences in performance will be largely impacted by architecture instead of pretrained weights. By having all models undergo the same training protocols, a fair comparison can be made to provide insights into the trade-offs between each model's architectural complexities and design philosophies when complex architectures are applied to smaller datasets.

The results found in this study will show the significance of designing and implementing a CNN based on the complexity of the use case and the trade-off in performance when blindly adapting well-known models with no consideration of the suitability of dataset.

This paper imports the pretrained models without the weights in order to achieve a fair comparison.

2. Methodology

2.1. Dataset

CIFAR-10 was the assigned dataset for this experiment. It has 60,000 colored images of low-resolution 32x32 pixels, equally distributed among 10 different classes of animals and vehicles. The equal distribution ensures a fair balance during training. The dataset will be split and stratified into training, validation and test sets with a 4:1:1 ratio (40,000 training, 10,000 validation, 10,000 testing).

The datatypes will also be converted to float32 format for faster computing. All datasets are then standardized for each color channel to achieve zero centering and ensure equal treatment of all color channels.

Finally, data augmentation was also applied to all datasets to introduce variability in training images and improve generalizability (Shorten & Khoshgoftaar 2019, p.60; Perez & Wang 2017). Data augmentations include horizontal flipping, width and height shifts.

2.2. Model Architectures

This paper examines 4 different architectures' performance on classifying the CIFAR-10 images. To ensure a fair comparison is made, each models were trained from scratch on the CIFAR-10 dataset without their pretrained weights so that any difference in performance would be attributed to architectural suitability for the task. The models include:

1. **Baseline CNN model** – A basic CNN designed for this experiment specifically tailored for use on CIFAR-10. It will act as the baseline. This model employs the use of multiple convolutional layers with small filter sizes, and uses max-pooling, dropout layers and batch normalization. The final fully connected layer uses a softmax function for classification.

2. **ResNet-18** – ResNet-18 is a deep residual network that uses "residual blocks" to propagate the learned gradients through many layers. For this study, this model had to be modified to fit the 32x32x3 inputs for CIFAR-10 in the initial layer, but maintaining the residual block structure.

3. **AlexNet** – AlexNet was designed for high-resolution images (224x224) and uses larger filter sizes. For this study, the filter size and stride needed to be reduced to fit CIFAR-10's smaller size, but the core architecture was maintained.

4. **MobileNet** – MobileNet is an efficient CNN which uses "depthwise separable convolutions". Similarly, the input layer was changed to 32x32 to fit CIFAR-10, with the core architecture untouched.

2.3. Training Strategy

All four models were trained under the same protocols to ensure fairness:

· **Data Augmentation** – Applied to introduce variations in data and help models generalize better and prevent overfitting. Augmentations include random horizontal flipping, width and height shifts, and slight zooming.

· **Batch Size and Epochs** – Batch size of 64 and 50 epochs for all models were implemented, with early stopping based on validation loss.

· **Learning Rate and Optimizer** – Adam Optimizer was initiated with a learning rate of 0.0001.

· **Regularization** – Dropout and Batch Normalizations were applied as part of best practice and helps prevent overfitting.

2.4. Evaluation Metrics

All four models were evaluated on their **accuracy** and **cross-entropy loss** on the CIFAR-10 test set which is typical for a multi-classification task. The training and validation loss curves were also visualized to check for any overfitting or underfitting patterns.

2.4.1 Note on Computational Limitations:

It is worth clarifying that due to computational constraints, this study was unable to fully reach the theoretical optimal performance for all the models. This was due to the study being done on a personal MacBook with an M2 chip, parallelisation enabled. On-campus computers also could not run the tests with much better speed, and Google Colab requires payment for projects of this size.

Training deep learning models to their upper limits requires both substantial time and computational resources with regards to hyperparameter tuning and it wasn't feasible. However, the experiments performed in this study was set up and aims to achieve the fairest evaluation possible within a reasonable time frame.

Using a consistent training protocol and applying fixed training parameters with a reasonable batch size and epochs, this setup emphasizes that differences in performance were primarily from architectural differences instead of variation in training conditions. Therefore, while models are not at their optimal performance, it can be observed how different architectures affect the performance depending on suitability to the task.

2.5. Model Parameters and Hyperparameters

The performance of CNNS are highly influenced by both their model parameters and hyperparameters, and understanding the distinctions of the two are essential:

Model parameters: Model parameters are the values that are learned over training, which are the weights and biases in each layer of the CNN (Goodfellow, Bengio & Courville 2016).

· **Hyperparameters**: Hyperparameters are the settings that are predefined before training of the model (Geron 2019). Hyperparameters control the model's behaviour and structure which influences the model's ability to learn during training.

2.6. Model parameters

Weights

Each layer in a CNN has filters and kernels with **weights** assigned to them. These filters will 'slide' over the input image and capture specific features such as edges, textrues, or colours. The weights determine the strength of the features which are then continually adjusted during training for the CNN to recognize these patterns in the data.

Biases

Biases are additional parameters that help the model shift the output of each layer to better fit the training data.

Weights and biases work together as model parameters and are continuously updated during training through a process called backpropagation. These learned parameters are what models are trained to learn and are directly correlated to the performance of the model. The more effective the model can learn the patterns with the weights and biases from training, the better the model will perform.

2.7. Hyperparameters:

Hyperparameters on the other hands are the settings that need to be defined before the training process begins. They are not learned by the model but instead setup while compiling the model. Hyperparameters control the behaviour of the model during training and decides how the model learns from training (Geron 2019).

Some key hyperparameters include:

· **Learning Rate**

The **learning rate** is the step size that the model updates the learned weight through each iteration of training. A high learning rate updates the weight by a higher ratio through each iteration and a lower one reduces the weight update. An overly high learning rate may cause the model to miss the global minima and never converge on the optimum weight; while a learning rate too low means the model learns very slowly and cause it to get stuck in suboptimal weights. A well-tuned learning rate that is adaptive is desirable in most cases to reduce training time while improving accuracy.

· **Batch Size**

Batch size refers to the number of images that the model processes in one pass before a weight update occurs. A small batch size provides is more detailed but includes noise; a larger batch size offers less noise and smoother updates but takes up more memory space. A balanced batch size is necessary to enable efficient learning without taking too much memory space.

· **Epochs**

Epochs are the number of complete passes performed through the training set. A higher number of epochs allows the model more runs to learn the patterns of the data but an excessively high epoch will cause the model to overfit the data and memorize the dataset instead of general patterns. Early stopping is used to stop training when the performance doesn't improve over a number of epochs.

· **Filter Number and Size**

Convolutional Layers in CNNs serves as filters/kernels that detect specific patterns in images. Filters have square dimensions usually in 3x3 or 5x5 that slides along the image in strides to pick up these patterns. A large filter size captures broader features while small filters capture fine details.

· **Stride and Padding**

The **kernel stride** is the number of pixel the filter moves after each slide. Large strides reduce spatial dimensions and lose general information while small strides capture fine details.

The **kernel padding** is how the kernel treats extra pixels that are outside the image and will affect the output size of the image. "Same" padding preserves the input size and "valid" padding reduces it.

· **Dropout Rate**

Dropout is a regularization technique where neurons are randomly deactivated during the training, Dropout helps prevent overfitting by allowing the network to use multiple "paths" for feature extraction instead of the same few neurons. The rate of dropout is the strength of how much neurons to deactivate during training.

2.8. Hyperparameter Tuning for Optimal Performance

Finding the right combination of these hyper parameters to gain the most optimum performance out of the model is done through hyperparameter tuning (Bergstra & Bengio 2012, pp.281-305; Li et al. 2018, pp.6765-6816). This combination allows the model to be trained in the best way and leads to faster training, better generalization and higher accuracy. The most common techniques to perform hyperparameter tuning are:

o **Grid Search**: a thorough systematic way of iterating through every combination of settings and comparing performance and saving the best setting for the model to be trained on.

o **Random Search**: this tuning method randomly chooses different hyperparameter settings to try and is much quicker but is less thorough than Grid Search.

For this paper, computational restraints do not allow for an extensive hyperparameter tuning to be done. However, standard hyperparameter values that tend to perform the generally well as a baseline were chosen and in essence, an educated guess was made.

Table 1.0 below shows the chosen hyperparameters:

Table 1. Hyperparameters used across all models

Hyperparameter	Value
Learning Rate	1e-4 (with ReduceLROnPlateau for some models)
Batch Size	64
Epochs	50 (with early stopping)
Dropout Rate	0.25, 0.5
Data Augmentation	Yes (flip, shift, zoom)
Optimizer	Adam
Activation Function	ReLU (hidden), Softmax (output)
Pooling	MaxPooling2D or Global Average Pooling
Regularization	Batch Normalization, Dropout

The architectural characteristics of the three imported models were left untouched and modifications were made as minimal as possible. All models followed the hyperparameters wherever possible.

3. Architecture of Models

In this section, an overview of all the model's architecture is explored to provide context on why each models were specifically designed for their respective tasks and the performance that can be reasonably expected. A summary of all models generated and made available in the appendix for a detailed look.

3.1. Baseline CNN model

The baseline CNN model is designed specifically with CIFAR-10's dataset characteristics in mind, with simplicity and efficiency as the inspiration. It will be a balance of layer depth and regularization to maximize performance while keeping training speed fast.

3.1.1 Architectural Structure

The baseline CNN consists of three main convolutional blocks, each larger than the preceding layer designed to extract increasingly complex features from the input image. The architecture can be summarized as follows:

o **Convolutional Layers:** The model begins with the first two convolutional layers each using a filter size of 3x3, with batch normalization and max pooling to downsample feature maps. In the later layers, the number of filters increase in each block, gradually from 32 initially up to 128 to allow the model to extract increasingly complex patterns while maintaining efficiency.

o **Pooling Layers:** Max pooling layers are used in each convolutional blocks to reduce the spatial dimensions. This step helps to reduce the number of parameters for efficiency of the network while still extracting features at the resolutions.

o **Dropout Layers:** Dropout layers are applied after convolutional blocks and dense layers to prevent overfitting.

o **Fully Connected Layers:** The final layers consist of a dense layer with ReLU activation and finally a softmax output layer for multi-class classification. The output is represented by what the model predicts the image to be out of the 10 classes. The baseline CNN model has 4,239,840 total parameters.

Design Philosophy: The baseline balances depth of layers, with small filter sizes while employing batch normalisation and dropout for regularizations. For a small dataset with low-resolutions, this is a model that can generalize well on test data without the huge overhead of complex architectures.

3.2. ResNet-18

ResNet-18 is derived from a Residual Network architecture design. The introduction of residual connections allowed deep networks to be developed while bypassing the problem of vanishing/exploding gradients (He et al. 2016, pp.770-778). Typically, Residual Networks are used for complex image classifications on large datasets with high-resolution inputs (He et al. 2016, pp.770-778). As it has very deep layers, the model is effective in capturing detailed features, but for CIFAR-10, the input layer needs to be modified to accept low-resolution inputs.

3.2.1 Architectural Structure

ResNet-18 has 18 layers in total as the name suggests. It has convolutional, batch normalization and fully connected layers but the layers are organized into 4 main residual blocks, each with two or more convolutional layers. The main difference in architecture is the usage of skip connections, which enables the input of a block to bypass convolutional layers and go directly into the block's output. This allows deep networks to be developed and avoid issues with gradient vanishing/exploding. The architecture can be summarized as follows:

o **Convolutional Layers:** Each residual blocks are made up of convolutional layers of 3x3 filters with batch normalization and uses ReLU activation functions.

o **Residual Connections**: The residual connections between blocks allows some bypass of convolutional layers by allowing some layers to add the input of a block to its output.

o **Strided Convolutions:** Instead of traditional max-pool layers, ResNet-18 uses strided convolutions in specific blocks to perform spatial dimension reductions.

o **Global Average Pooling**: The final layer of ResNet-18 uses global average pooling to condense the spatial information and then uses softmax activation in the final fully connected layer.

The ResNet-18 model has 33,545,888 total parameters.

Design Philosophy: Resnet-18 was built and designed to be a deep network which excels in handling high-resolution images. The use of residual connections allow for uninterrupted flow of gradients which is essential for deep networks with large numbers of layers. However, if applied to a small dataset with low-resolution images, there is an inherent risk of overfitting simply due to the architectural design. The high number of layers will likely cause the model to overfit, and the small dataset simply isn't enough to train a model of this complexity. All in all, using Resnet-18 on CIFAR-10 does not make use of the potential of the data and will likely underperform.

3.3. AlexNet

AlexNet, when introduced in 2012 was one of the first implementations of ReLU activation function and dropout regularization, influencing how CNNs were designed (Krizhevsky, Sutskever & Hinton 2012, pp. 1097-1105). AlexNet is the most similar in architecture of all three pretrained models to our baseline CNN model, but the main difference is that AlexNet was designed for high-resolution images of 224x224 pixels. Its design allowed it to capture hierarchical visual features over its layers (Krizhevsky, Sutskever & Hinton 2012, pp. 1097-1105). For this study, the input layer will be modified to accept 32x32 low-resolution images.

3.3.1 Architectural Structure

AlexNet has a total of eight layers, including five convolutional layers and three fully connected (dense) layers. Its architecture can be summarized as follows:

- **Large Filter Sizes in Initial Layers**: The first convolutional layer in AlexNet uses a large filter size of 11x11 with a stride of 4. This choice was intended to capture large patterns early on and to reduce the spatial dimensions more quickly, which suited its original input image size of 224x224. In the context of CIFAR-10, which uses 32x32 images, this filter size is reduced to 3x3 to retain useful information from smaller inputs.

- **Convolutional and Pooling Layers**: The subsequent convolutional layers use smaller filter sizes and strides, allowing AlexNet to capture more refined spatial features as it goes deeper. Max-pooling layers are applied after certain convolutional layers to reduce spatial dimensions and retain only the most relevant features, balancing computation and model efficiency.

- **ReLU Activation**: Similar to our baseline model, AlexNet uses ReLU activation function to introduce non-linearity in a computationally efficient way. ReLU helps mitigate the vanishing gradient problem, allowing AlexNet to learn complex features effectively while accelerating convergence.

- **Dropout Layers**: Dropout was introduced in AlexNet as a regularization method to prevent overfitting in the dense layers.

- **Dense Layers and Softmax Output**: The final layers are fully connected (dense) layers, which integrate all spatial information learned in the previous layers and combine features to classify the image. The last layer uses softmax activation to produce class probabilities, making AlexNet effective for multi-class classification tasks.

The AlexNet model has 6,343,712 parameters

Design Philosophy: AlexNet's design philosophy is the inspiration of our baseline CNN model. We can draw similarities between the models such as the use of ReLU, Batch Normalization and dropout regularizations. The main difference is that the initial input size of AlexNet was designed for high-resolution images, and the number of layers compared to our baseline model is higher. There are small differences in frequency of batch normalization and dropouts as well, but the performance of AlexNet is expected to be similar to our baseline, but slightly lower.

The AlexNet model has 6,343,712 total parameters.

3.4. MobileNet

MobileNet is a deep learning architecture for smaller mobile and embedded devices. It's main architectural feature is the depthwise separable convolution, allowing it to significantly reduce the number of parameters and operations (Howard et al. 2017). This feature makes it a good choice for computationally limited devices such as smartphones while still maintaining solid performances on image classification tasks (Howard et al. 2017).

Architectural Structure

MobileNet's architecture is defined by the implementation of depthwise separable convolutions, but the remaining architecture remains similar to traditional CNNs. Its architecture can be summarized as follows:

o **Depthwise Separable Convolutions:** A convolutional layer which is made up of two parts with different functions:

o **Depthwise convolution:** Applies filters to only the input channel instead of all channels, to preserve computational power.

o **Pointwise Convolution layer:** Applies a 1x1 convolution to recombine information from all channels. This layer creates the feature maps from the depthwise convolutions and learns feature combinations across channels.

o **Width and resolution multipliers:** These are hyperparameters exclusive to MobileNet which can be tuned depending on model size and complexity.

o **Width Multiplier:** Scales the number of filters in each layer.

o **Resolution Multiplier:** Scales the input image resolution.

o **ReLU and Batch Normalization:** Each convolutional layer is followed by ReLU activation and batch normalization to stabilize and accelerate training.

o **Global Average Pooling and Fully Connected Layer:** Similar to ResNet-18, the final layer uses a global average pooling layer before the dense layer which finally uses a softmax activation for multi-classification.

The MobileNet model has 10,041,312 total parameters.

Design Philosophy: MobileNet was created for a lightweight and fast performance on image classification, and introduced depthwise separable convolutions to achieve this. The other architectural characteristics make it desirable for CIFAR-10 multiclassification theoretically. However, the feature extraction offered by MobileNet might not be detailed enough to achieve satisfactory performance for the task.

4. Results

This section will present the final performance results on the four CNNs chosen for this study. The key metrics, accuracy and loss are analyzed to assess the model's performance and suitability for the dataset. Table 2.0 summarises the quantitative performance of each model:

Model	Test Accuracy (%)	Test Loss
Baseline CNN	**87.47**	0.38
ResNet-18	80.91	0.58
AlexNet	83.04	0.41
MobileNet	73.00	0.72

· **Baseline CNN**: The baseline model achieved the highest test accuracy of **87.47%**, with relatively low test loss. Its architecture, specifically designed for the CIFAR-10 dataset, balanced complexity and capacity effectively, enabling it to generalize well.

· **ResNet-18**: With a test accuracy of **80.91%**, ResNet-18's performance was slightly lower than the baseline model. The residual block structure made it powerful for complex feature learning but introduced overfitting, as shown in the training and validation curves. This model's depth may be excessive for CIFAR-10's lower-resolution images.

· **AlexNet**: AlexNet performed moderately well, achieving **83.04%** accuracy on the test set. However, due to its high-capacity fully connected layers, it showed signs of overfitting, as seen in the separation of training and validation curves. This overfitting is likely due to AlexNet's originally intended application to larger datasets like ImageNet.

· **MobileNet**: MobileNet achieved a test accuracy of **73.00%**, which was the lowest among the models. Its lightweight architecture, with reduced parameters via depthwise separable convolutions, limited its capacity to learn complex patterns. This underfitting is shown in the training and validation curves, with lower accuracy and higher test loss.

2. Analysis of Training and Validation Curves

The accuracy and loss plots for each model reveal important characteristics of their learning behavior of each model:

· **Baseline CNN**: The training and validation curves showed consistent improvement with minimal overfitting. The close alignment between training and validation accuracy suggests that the baseline CNN was well-suited to CIFAR-10, generalizing effectively without unnecessary complexity.

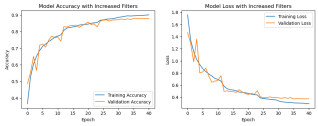

Figure 1. Baseline CNN Accuracy and Loss Curve

· **ResNet-18**: ResNet-18's training accuracy was high, but its validation performance was somewhat lower, indicating overfitting. The residual connections allowed the model to fit the training data well, yet its depth led to diminished generalization on the validation and test sets.

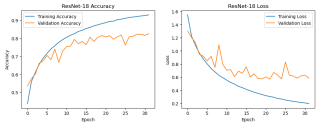

Figure 2. ResNet-18 Accuracy and Loss Curve

· **AlexNet**: AlexNet's training and validation curves show divergence, particularly in later epochs, where training accuracy improved while validation accuracy plateaued. This divergence suggests overfitting due to the large fully connected layers and model complexity, which were better suited for high-resolution data.

· **MobileNet**: MobileNet's training and validation curves indicated underfitting, as both metrics leveled off early in training. Its lightweight design, optimized for efficiency, limited the model's ability to capture essential features in CIFAR-10, leading to lower accuracy.

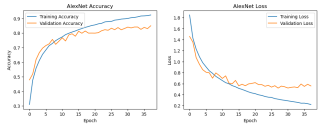

Figure 3. AlexNet Accuracy and Loss Curve

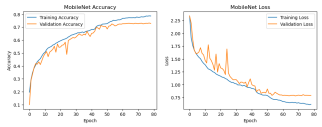

Figure 4. MobileNet Accuracy and Loss Curve

5. Discussion of Trade-offs in Model Complexity

The results demonstrate that simpler, well-tailored models can outperform complex architectures on smaller, low-resolution datasets like CIFAR-10. The baseline CNN achieved the best performance, balancing complexity and generalization effectively. More complex models such as ResNet-18 and AlexNet, originally designed for high-resolution, large-scale datasets, showed signs of overfitting and struggled to adapt to CIFAR-10's constraints. MobileNet, optimized for efficiency, provided reasonable performance but fell short due to its limited capacity.

6. Code

All code used in this experiment can be found on my github repository. A pynb file along with a full code chunk is made available publicly.

7. Conclusion

In summary, this study highlights the importance of designing CNN architectures to suit the dataset's complexity and resolution. CNN architectural planning and designs significantly impacts the resulting performance of the model. The custom baseline CNN outperformed state-of-the-art architectures by being more suitable for CIFAR-10's characteristics, showing that model generalization can be achieved with simpler structures tailored to the data, which saves time and computational power. Future work could explore ways to adapt high-capacity models to small datasets or investigate more efficient architectures optimized for diverse datasets. This study emphasizes that selecting or designing models based on the task requirements and resource constraints remains key to achieving optimal performance in deep learning applications. Further recommendations for future experimentation can be improving on the baseline CNN model implemented in this study and lightly increasing the complexity of the model with layers, batch normalisation and dropout regularizations to achieve even higher performance than baseline, followed by an extensive hyperparameter tuning.

8. References

1. Bergstra, J. & Bengio, Y. 2012, 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research*, vol. 13, pp. 281–305.

2. Geron, A. 2019, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd edn, O'Reilly Media, Sebastopol, CA, Chapter 11.

3. Geron, A. 2019, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd edn, O'Reilly Media, Sebastopol, CA, Chapter 14.

4. Goodfellow, I., Bengio, Y. & Courville, A. 2016, *Deep Learning*, MIT Press, Cambridge, MA, Chapter 6.

5. Goodfellow, I., Bengio, Y. & Courville, A. 2016, *Deep Learning*, MIT Press, Cambridge, MA, Chapter 9.

6. He, K., Zhang, X., Ren, S. & Sun, J. 2016, 'Deep residual learning for image recognition', in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, Las Vegas, NV, pp. 770–778.

7. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. & Adam, H. 2017, 'MobileNets: Efficient convolutional neural networks for mobile vision applications', *arXiv preprint*, arXiv:1704.04861, viewed 4 November 2024 https://arxiv.org/abs/1704.04861

8. Krizhevsky, A. & Hinton, G. 2009, *Learning Multiple Layers of Features from Tiny Images*, University of Toronto, viewed 4 November 2024 https://www.cs.toronto.edu/kriz/learning-features-2009-TR.pdf.

9. Krizhevsky, A., Sutskever, I. & Hinton, G.E. 2012, 'ImageNet classification with deep convolutional neural networks', in *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., Lake Tahoe, NV, pp. 1097–1105.

10. LeCun, Y., Bengio, Y. & Hinton, G. 2015, 'Deep learning', *Nature*, vol. 521, no. 7553, pp. 436–444.

11. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. 2018, 'Hyperband: A novel bandit-based approach to hyperparameter optimization', *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816.

12. Perez, L. & Wang, J. 2017, 'The effectiveness of data augmentation in image classification using deep learning', *arXiv preprint*, arXiv:1712.04621, viewed 1 November 2024 https://arxiv.org/abs/1712.04621.

13. Shorten, C. & Khoshgoftaar, T.M. 2019, 'A survey on image data augmentation for deep learning', *Journal of Big Data*, vol. 6, no. 1, p. 60.