**Abstract**

# **Microsoft Word Author Guidelines for CVPR Proceedings**

Aryan Puri

University Of Adelaide

[a1914021@adelaide.edu.au](mailto:a1914021@adelaide.edu.au)

*CNNs are gaining popularity in the field of computer vision due to their feature capturing ability, resulting in high accuracy image processing.*

*This paper tries to investigate how they can be applied to image classification tasks. In this paper, the focus is evaluation and comparison of ResNet-18, MobileNetV2, and VGG architectures within a standardized experimental framework.*

*By implementation and systematic testing, assessment of performance in terms of accuracy, efficiency, and generalization was the focus.*

*Results reveal each model’s strengths and limitations in practical image classification applications and highlights factors influencing model selection for practical applications.*

*Broadly, the paper aims to deepen general understanding of CNN-based to promote wider use in real-world visual recognition problems.*

# **Introduction**

CNNs enable learning features directly from image data, enabling effective feature extraction and improved accuracy on challenging tasks.

This paper explored its capabilities for image classification. It aims at comparing three CNN architectures: ResNet-18, MobileNetV2, and Dense Net.

* ResNet-18 selected for residual connections. This solves vanishing gradient problem, enabling networks to maintain accuracy across layers.
* MobileNetV2 was selected because it employs depth wise separable convolutions to reduce computational demands. This helps in constrained applications.
* Dense Net was selected as it links each layer to every subsequent layer in the network. This enhances feature reuse and improves gradient flow. This helps us learn more robust features with fewer parameters.

Each model was trained and tested on the dataset. I employed a standardized experimental setup to assess each model's accuracy, computational efficiency, and generalization ability on unseen data.

To maximize performance, I used preprocessing like resizing, data augmentation, normalization and techniques like dropout and adaptive learning rate scheduling.

The results show distinct advantages and disadvantages associated with each architecture.

* ResNet-18, demonstrated strong accuracy and stability, making it well-suited for applications that benefit from deeper feature extraction.
* MobileNetV2 achieved competitive accuracy while significantly reducing resource requirements, underscoring its utility in scenarios where computational efficiency is critical.
* DenseNet leveraged dense connections and effectively captured features. It achieved high accuracy with a streamlined parameter set, excelling in both efficiency and performance.

Through this study, the aim was providing guidance for selecting and optimizing CNN models in diverse applications.

The findings would support more informed decision-making in applying CNNs to real-world visual recognition tasks, from high-performance settings to environments with limited computational resources.

# **Evaluating CNNs for image classification**

## **2.1 Working**

Once an image is fed into the network with the intention of classifying it, we begin by feeding input to the convolution layer which identifies and learns spatial hierarchies of features through a series of transformations. Filters in these layers are initialized with random values, updated while training via backpropagation.

The convolution operation is essentially a dot product between the filter and image patches produce feature maps which highlight feature presence. The maps created are stacked depth-wise. Key parameters include stride, controls movement across the image, and padding, allows filters to cover edge pixels.

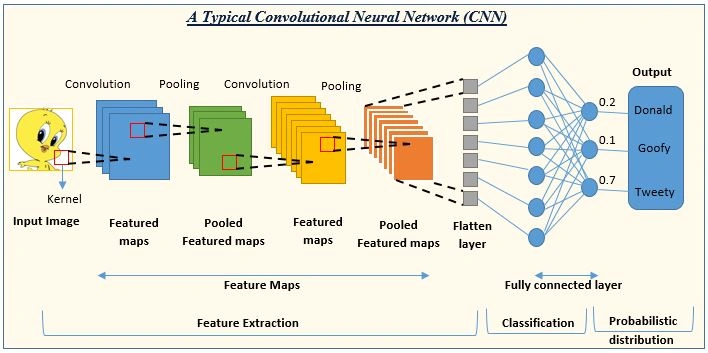
The pooling layers reduce the spatial resolution of feature maps. It decreases the computational load in subsequent layers. It also results in translational invariance, which means minor shifts in the input image have less impact on feature detection. At this layer, the feature maps are compressed into pooled feature maps, while retaining preserving information about the image structure.

This process is repeated for multiple layers, resulting in transition to fully connected layers, where neurons are connected to all activations from the previous layer. The additional layers serve to extract increasingly complex features like patterns and parts of objects.

Once the final pooling layer is finished, the resulting maps are flattened into a single vector, to feed into the interconnected layer. This dense layer learns to interpret the high-level features extracted by the convolutional and pooling layers and maps them to specific classes.

Lastly, the output layer uses a probabilistic distribution to generate the probabilities for each class.

We broadly divide the process into feature extraction and classification. Feature extraction is performed by the convolution and pooling layers. Classification is done by the fully connected layers.



## **Advantages**

CNN offers key advantages in the computer vision domain. Learning from raw images, they eliminate the need for manual feature engineering by learning through their convolutional layers.

Parameter sharing makes them computationally efficient, by reducing the number of parameters significantly.

As mentioned in the working, they exhibit translation invariance, making them robust to small translations and shifts.

Furthermore, local connectivity structure allows CNNs to

exploit the spatial structure of images efficiently. Using small filters enables learning local features without a prohibitive number of connections, which scales well with high-resolution images.

Additionally, they handle high-dimensional data effectively.

Finally, CNNs have shown great results in The ability to learn complex patterns and hierarchies has led to exceptional performance in image classification.

## **Disadvantages**

## CNNs also have limitations. A high number of parameters and multiplications introduce a lot of computation. High amounts of data augmentation and regularization is required to enable deeper layers to learn complex features, while avoiding overfitting. This can be addressed through specialized architectures but is a problem. Their architecture allows limited global context awareness and makes understanding larger, contextual relationships challenging. CNNs work in a black-box, making accountability a challenge due to the opaque transformations. They struggle to provide interpretable decision processes, especially in high-stakes domains where accountability is crucial.

# **Models Specification**

# **ResNet-18**

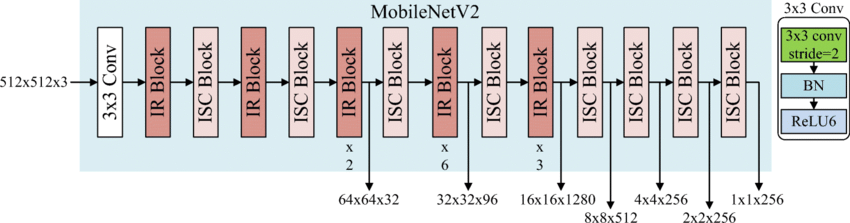
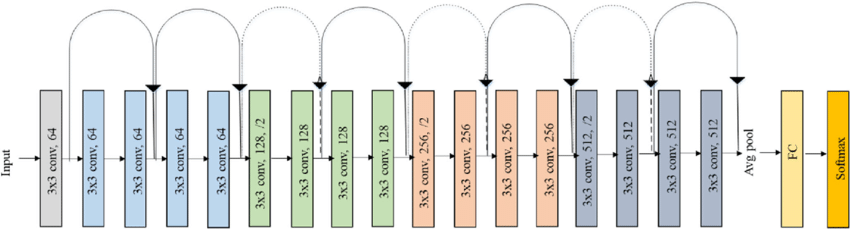
ResNet-18 is designed with residual connections, allowing it to effectively train deeper networks by circumventing the vanishing gradient problem.

The skip connections in each residual block enable gradients to flow directly across layers, preserving information from earlier stages.

This architecture excels in capturing complex hierarchical features without redundant learning, making it highly effective for tasks like CIFAR-10 classification.

By leveraging data augmentation (random flips, crops, and normalization) and feeding 224x224 images, ResNet-18 achieved strong generalization, converging quickly and maintaining high accuracy even on unseen data.

The design of residual blocks allowed ResNet-18 to balance depth and performance without a massive increase in computational cost.



MobileNetV2

ResNet-18

# **MobileNetV2**

MobileNetV2 focuses on efficiency through its unique inverted residual blocks and depthwise separable convolutions.

This design dramatically reduces the number of parameters and computation required per layer, making MobileNetV2 ideal for deployment on mobile or resource-constrained devices.

The inverted residuals work by first expanding the feature space, applying efficient depthwise convolutions, and then reducing the dimensionality—effectively capturing essential spatial information without excessive computation.

When trained with 224x224 augmented images, MobileNetV2 demonstrated competitive accuracy by learning robust features from the augmented data.

The model’s efficient architecture proved effective in learning CIFAR-10 features while maintaining a lightweight structure.

# **DenseNet**

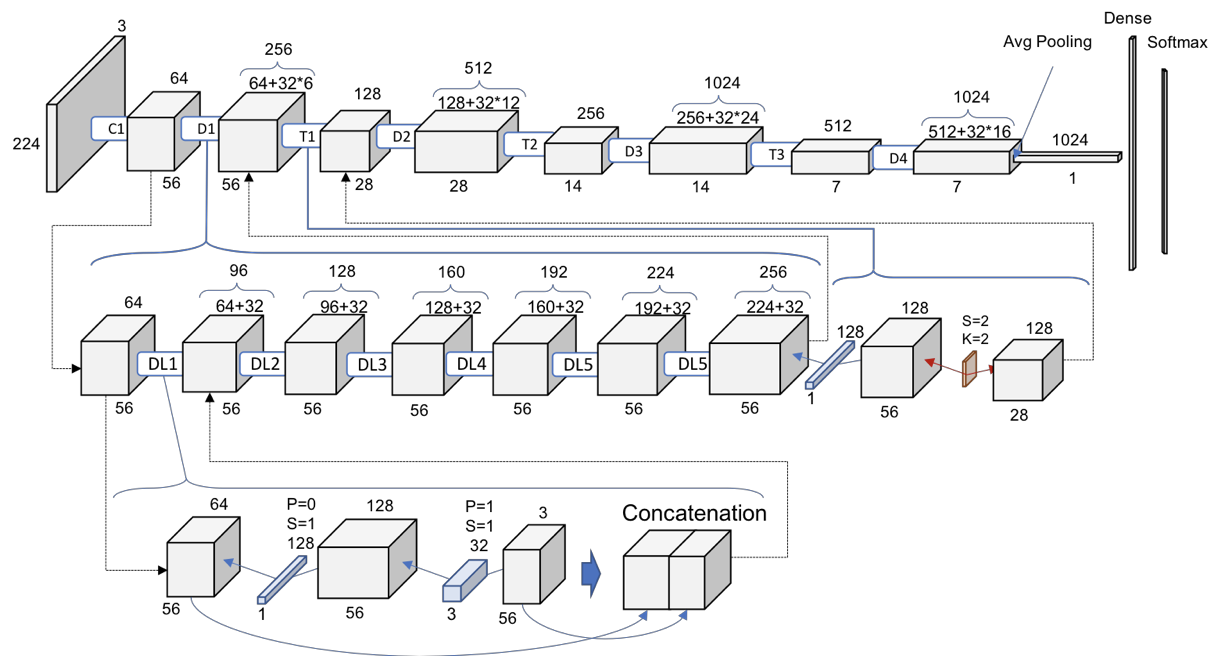
DenseNet is built on a densely connected architecture where each layer receives the feature maps from all preceding layers, resulting in efficient gradient flow and feature reuse.

This dense connectivity structure promotes a more compact and parameter-efficient network since each layer does not need to relearn redundant information.

DenseNet’s design reduces the number of parameters and enhances model generalization, even with fewer filters per layer compared to other architectures.

Feeding the model augmented 224x224 images enabled DenseNet to capture fine-grained details in CIFAR-10, and its compact architecture allowed it to converge effectively while maintaining high accuracy.

The dense connections improved training stability and model robustness, making DenseNet well-suited for high-performance classification tasks.



# **Code Repository**

<https://github.com/u4DHzVmSHVQELmd/Deep-Learning-Assignment-2.git>

The code provided will download the dataset when run.

# **Dataset**

For my experiment I used the CIFAR-10 dataset. The dataset consists 60,000, 32x32 color images in 10 distinct classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Every class has 6,000 images, split into 50,000 training images and 10,000 test images. The dataset has low resolution images of diverse objects. This makes it ideal to evaluate our chosen networks.

# **Experimental Setup and Method**

The objective of this experiment is to evaluate the performance of the CNN networks on the CIFAR-10 image classification task by systematically exploring different optimization techniques.

Specifically, the experiment aims to:

* **Assess the impact of various optimizers** on training and model accuracy, starting with Stochastic Gradient Descent (SGD) with Momentum, transitioning to Adam, and finally applying Adam with a Cosine Annealing learning rate scheduler.
* **Improve model generalization and accuracy** through data augmentation techniques, including random cropping, horizontal flipping, and normalization, to create a more robust model capable of handling unseen data.
* **Investigate the effectiveness of adaptive learning rate scheduling** in combination with the Adam optimizer to determine if this strategy provides finer control over convergence and leads to higher accuracy.
* **Identify the optimal training setup** for ResNet-18 on CIFAR-10 in terms of accuracy, stability, and generalization, comparing the performance metrics (training loss, validation loss, and accuracy) across different optimization configurations.

# **Optimizers Used**

### **Stochastic Gradient Descent (SGD) with Momentum**

SGD is a popular optimizer for training deep learning models. By updating weights based on mini-batches of data, it introduces randomness that helps the model escape local minima and potentially find better solutions. Adding momentum improves SGD by accumulating past gradients, allowing it to move faster in directions with consistent gradients and reducing fluctuations in less stable directions. This leads to smoother convergence, especially useful in scenarios where gradient updates are noisy. Despite its simplicity, SGD with momentum remains highly effective, particularly for large, complex models like ResNet, as it encourages steady convergence while helping to avoid suboptimal minima.

### 

### **Adam (Adaptive Moment Estimation)**

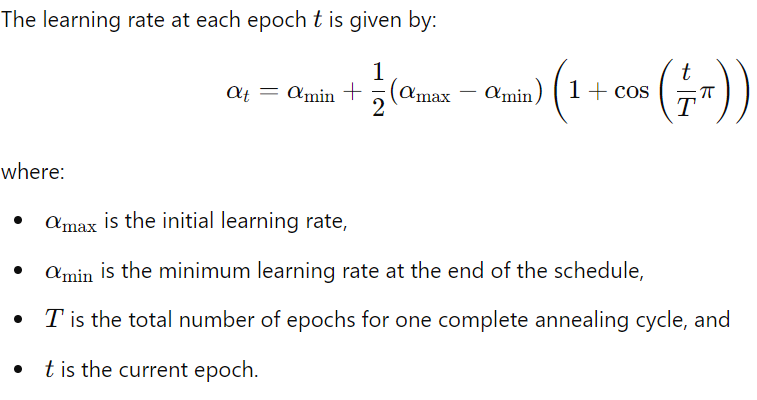
Adam combines the benefits of two other optimizers Momentum and RMSprop by computing adaptive learning rates for each parameter. It maintains both the moving average of gradients and the squared gradients, enabling fast convergence without extensive tuning. Adam is particularly effective in handling sparse gradients and diverse datasets, making it popular for a wide range of deep learning applications. The adaptive nature of Adam’s learning rate makes it suitable for models that require rapid convergence and stability, such as those trained on highly variable data or with complex architectures

### **7.3 Step Learning Rate Scheduler:**

While not an optimizer in itself, the Step Learning Rate Scheduler is a strategy used in conjunction with optimizers (like SGD) to adapt the learning rate dynamically. By reducing the learning rate at fixed intervals (e.g., every 10 epochs), it allows the model to make larger updates during the early stages of training and finer adjustments as it nears convergence. This helps prevent oscillations in loss values and enables the model to settle into an optimal minimum, enhancing the final accuracy and stability. The Step LR Scheduler is particularly useful when paired with SGD, as it helps the model transition from fast exploration to focused convergence over time.

### **7.4 Cosine Annealing:**

The Cosine Annealing scheduler is a learning rate decay technique designed to gradually reduce the learning rate in a non-linear fashion, following a cosine curve over a specified number of epochs. Unlike step-based decay, which reduces the learning rate at fixed intervals, Cosine Annealing provides a smooth transition from the initial learning rate to a minimal rate, allowing the model to fine-tune parameters delicately as it approaches convergence.



This cosine function produces a gradual, oscillating decay of the learning rate.

# **8. Results**

**8.1.** **ResNet-18:**

**8.1.1. SGD with Momentum:**

# The model was trained over 20 epochs using Stochastic Gradient Descent (SGD) with Momentum and an adaptive learning rate scheduler.

# The initial learning rate was set to 0.1, with a momentum of 0.9 and a weight decay of 5e-4 to mitigate overfitting.

# The learning rate was decayed by a factor of 0.1 every 10 epochs, allowing for faster convergence in the early stages of training and finer adjustments as the model approached its optimal state.

# Training loss began high value at 2.1531 and decreased consistently reaching 0.1839 in the 20th epoch.

# This suggests that the model benefited from the momentum. This helps faster convergence.

# Test loss began at 1.6585 and decreased over time, eventually reaching 0.3286 in the final epoch.

# The gradual reduction in test loss across epochs, indicates great generalization.

# The model complexity aligns well with the dataset, and the applied regularization methods, such as weight decay, prevented overfitting.

**8.1.2. Adam:**

20 epochs using the Adam optimizer, with an initial learning rate of 0.001. The results demonstrate a smooth convergence with consistent decreases in both training and test loss, alongside steady improvements in test accuracy over the epochs.

Training loss began at 1.4460, and it decreased progressively, reaching a final value of 0.1603 by the 20th epoch. Similarly, the test loss started at 1.2566 in the first epoch and decreased to 0.3728 by the end of training.

The consistent drop indicates stable learning without signs of overfitting or significant oscillations, suggesting that the Adam optimizer effectively managed gradient updates by adapting the learning rate dynamically based on individual parameters.

Test accuracy started at 54.81% in the first epoch, reflecting the model's rapid adaptation to the dataset. By epoch 10, test accuracy reached 86.05%, indicating successful feature learning and robust initial convergence. In the later stages, accuracy gains became more gradual, with the final test accuracy reaching 87.93%.

This accuracy trend reflects the Adam optimizer’s ability to balance exploration and fine-tuning, achieving high accuracy while avoiding large fluctuations in performance

**8.1.3. Adam with Cosine Annealing:**

While this gave identical as with Adam optimizer the highest accuracy it achieved was 91.13%. Also, a smoother loss curve.

**8.2.** **MobileNet V2:**

**8.2.1.** **SGD with Step Decay**

The SGD optimizer, combined with step decay, achieved consistent convergence. The training loss decreased from an initial value of 1.2408 to 0.3297 by the 20th epoch, and the test loss dropped from 0.8925 to 0.4346. Test accuracy showed steady improvement from 68.85% in the first epoch to 84.93% by the final epoch.

The model generalizes well, as evidenced by the stable test accuracy in later epochs. The step decay of the learning rate (reducing the learning rate every 10 epochs) allowed the model to fine-tune itself as training progressed, achieving high accuracy with reduced oscillations.

Thefinal test accuracy of 84.93% reflects the effectiveness of SGD with momentum for stable and controlled convergence.

**8.2.2. Adam Optimizer**

The Adam optimizer started with a relatively low training loss of 0.5900, indicating rapid early convergence due to adaptive learning rates. However, in later epochs, the model experienced slight instability, with the training loss increasing to 0.7425 by the 20th epoch, and test loss increasing to 0.6632. This was reflected in the test accuracy, which peaked at around 84% but dropped to 77.59% by the final epoch.

While Adam initially allowed fast learning, the fluctuations in both training and test loss in later epochs suggest possible overfitting or inadequate generalization. The lack of a learning rate decay mechanism may have contributed to this instability in the later stages.

The final test accuracy of 77.59% indicates that Adam, though effective for rapid convergence, may benefit from learning rate scheduling for stability in prolonged training sessions.

**8.2.3. Adam with Cosine Annealing**

The combination of Adam with Cosine Annealing provided a smooth decay in the learning rate following a cosine curve. Training loss started at 0.6188 and decreased to 0.3384 by the final epoch. Test loss also showed a downward trend from 0.6023 to 0.4466, and test accuracy improved from 79.29% initially to a peak of 85.11%.

Cosine Annealing helped to maintain generalization by periodically reducing the learning rate in a smooth, non-linear manner, which prevents drastic drops that could lead to instability. This allowed the model to improve its accuracy incrementally while avoiding large fluctuations, especially in later epochs.

The final test accuracy of 84.77% suggests that the Cosine Annealing scheduler, when combined with Adam, provides both the rapid convergence of adaptive learning and stable generalization, closely matching the performance of SGD.

**8.3.** **DenseNet:**

**8.3.1. ADAM**

The training process showed a steady decrease in training loss from 0.4739 in the first epoch to 0.0342 by the 8th epoch, while validation loss followed a similar downward trend, reaching a minimum of 0.1271 by the 6th epoch. Training accuracy improved from 84.50% to 98.86%, with validation accuracy closely aligned, rising from 93.48% to 96.01%, indicating strong generalization and minimal overfitting. Early stopping was triggered at the 9th epoch, preventing overfitting and achieving a final test accuracy of 95.82%, consistent with validation performance. This training process reflects efficient convergence and robust generalization due to effective optimization and early stopping.

# **9. References**

**Optimizers and Learning Rate Schedulers**

1. **Adam Optimizer**  
   D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations (ICLR)*, 2015.  
   [Link: <https://arxiv.org/abs/1412.6980>]
2. **SGD with Momentum**  
   Y. Nesterov, "A Method of Solving a Convex Programming Problem with Convergence Rate O(1/k^2)," *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983.
3. **Step Learning Rate Scheduler**  
   G. Hinton, L. Deng, D. Yu, et al., "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82-97, Nov. 2012.  
   [Link: https://doi.org/10.1109/MSP.2012.2205597]
4. **Cosine Annealing and Cosine Annealing with Warm Restarts (SGDR)**  
   I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," in *International Conference on Learning Representations (ICLR)*, 2017.  
   [Link: <https://arxiv.org/abs/1608.03983>]

**Convolutional Neural Networks and Architectures**

1. **ResNet (Residual Networks)**  
   K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778.  
   [Link: <https://arxiv.org/abs/1512.03385>]
2. **MobileNetV2**  
   M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510-4520.  
   [Link: <https://arxiv.org/abs/1801.04381>]
3. **DenseNet (Densely Connected Networks)**  
   G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261-2269.  
   [Link: <https://arxiv.org/abs/1608.06993>]

**General Deep Learning Concepts**

1. **Understanding Gradient Descent and Optimization**  
   S. Ruder, "An Overview of Gradient Descent Optimization Algorithms," *arXiv preprint arXiv:1609.04747*, 2016.  
   [Link: <https://arxiv.org/abs/1609.04747>]
2. **Regularization and Hyperparameter Tuning**  
   I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.  
   [Link: <https://www.deeplearningbook.org/>]

**Hyperparameter Tuning and Learning Rate Scheduling**  
L. N. Smith, "A Disciplined Approach to Neural Network Hyper-Parameters: Part 1 - Learning Rate, Batch Size, Momentum, and Weight Decay," *arXiv preprint arXiv:1803.09820*, 2018.  
[Link: <https://arxiv.org/abs/1803.09820>]