# Performance Analysis of Convolutional Neural Networks on the CIFAR-10 Dataset

Evam Kaushik
Student ID: a1909167
The University of Adelaide
evam.kaushik@adelaide.edu.au

## Abstract

*This report compares the performance of several Convolutional Neural Network (CNN) architectures for performing classification on the CIFAR-10 dataset. I first develop a CNN from scratch and incrementally introduce enhancements such as Dropout, Batch Normalisation etc. and record its performance. I then explore several preexisting architectures with both, their native weights and from randomly initiated weights. The development process then explores various hyperparameters, such as different optimizers, learning rates, and regularisation methods.*

*"The purpose of computing is insight, not numbers."*

*– Richard Hamming*

## 1. Introduction

There are many prevalent Neural Network architectures[4–6, 8] that have been used for the task of image classification, of which, Convolutional Neural Networks (CNNs)[7] are the state of the art. I use the CIFAR-10 dataset[1] as a testing dataset to assess the performance of each model due to its complexity and diversity, capturing various visual features.

I utilise pretrained models like the MobileNetV2 [9], or the architecture of predefined models such as AlexNet [7] in order to aid training. I also bring aboard the massive ResNet152V2 model[2] with more than 58 Million parameters for comparision. I also train a custom CNN in order to first study effects of various hyperparameters and training methods.

## 2. Problem Background and Comparison of Approaches

Convolutional Neural Networks (CNNs) are adept for image classification tasks [10] owing to their ability to analyse an image in successive patches. This gives them the ability to preserve context for parts of the image with respect to each other. The CNN architectures I examine in this report are:

- **Custom CNN Model**: CNN built incrementally introducing dropout [16], batch normalization [17], and data augmentation techniques [19]. The model architecture is provided at the end of the report A.
- **MobileNetV2** [9]: Adept for mobile and edge devices, MobileNetV2 has depthwise separable convolutions as well as inverted residuals. The model is fine-tuned to accept 32x32 images in 3 colour channels.

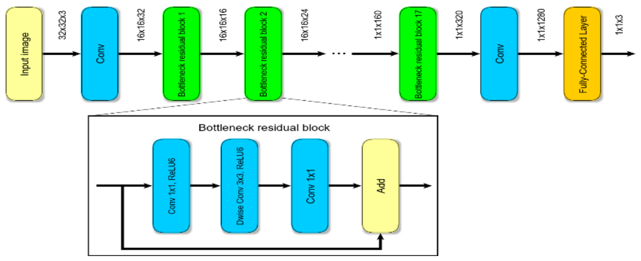


Figure 1. Architecture of MobileNetV2 Model

- **AlexNet** [7]: AlexNet was the first deep CNN that utilised GPUs for the process of model training. It is still used as a benchmark model for testing several CNNs. I trained a modified AlexNet to take a custom input of 32x32 pixels over 3 colour channels on CIFAR-10 from scratch.The detailed architecture for AlexNet can be found in the Appendix sectionA[1]

[1]Note, that the original AlexNet model takes an input of 227x227x3. This variant is modified to take as input the CIFAR-10 dataset which is of dimensions 32x32x3.

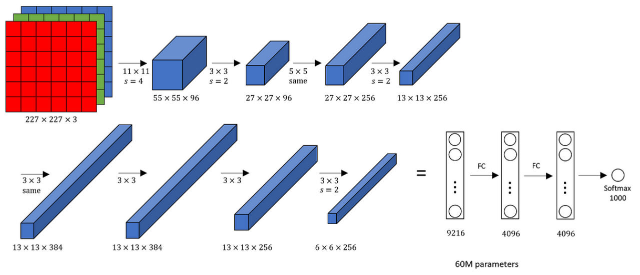


Figure 2. Architecture of the Original AlexNet

- **ResNet152V2**: A model very deep Convolutional Neural Network with multiple skip connections implemented via residual blocks[2, 8].

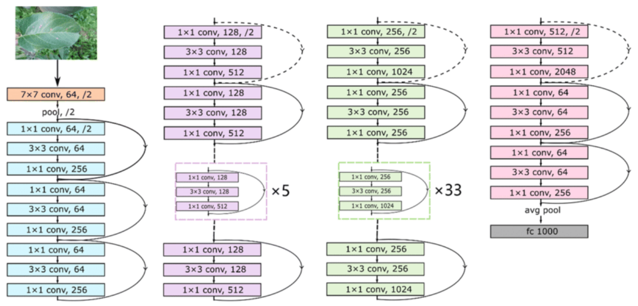


Figure 3. Architecture of the Resnet152V2 Model

## 3. Methodology and Implementation

I used TensorFlow [11] as the deep learning framework. Standardisation and normalisation were applied to CIFAR-10 as preprocessing techniques [18].

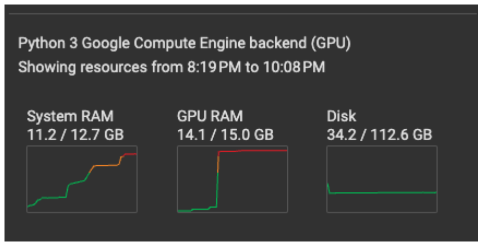
### 3.1. Data Splitting and Validation

The Data was split into a train/test set of 80/20. The training data was then further split into training and validation sets, also in an 80/20 split, using the ImageDataGernerator library.

### 3.2. Hyperparameter Tuning

The following hyperparameters were used to optimise model performance:

- **The Optimisers**: The Stochastic Gradient Descent (SGD) with momentum and Adam [14] was used.
- **The Learning Rates**: The learning rates of 0.1, 0.01 and 0.001 were used.
- **Regularisation Methods**: L2 regularisation was used to apply weight decay with the values 0, 0.0005, and 0.001.
- **Data Augmentation**:The following image synthesizing strategies were implemented and combined image augmentation with width and height shift and horizontal flip [15].


Python 3 Google Compute Engine backend (GPU)
Showing resources from 8:19 PM to 10:08 PM

| System RAM | GPU RAM | Disk |
|---|---|---|
| 11.2 / 12.7 GB | 14.1 / 15.0 GB | 34.2 / 112.6 GB |

Figure 4. Computational Resource consumption visualisation on Google Colab

- **Incremental Improvements**: For the custom CNN model dropout, batch normalization and data augmentation were applied as new techniques and tested.[12].

### 3.3. Implementation Details

For each architecture, I implemented the following:

- **MobileNetV2**: Loaded pretrained weights from ImageNet, adjusted input size to 32x32, replaced the classification head, and fine-tuned the model.
- **AlexNet**: Implemented from scratch, following the original architecture [7], adapted for CIFAR-10 input size.
- **Custom CNN Model**: Started with a simple CNN architecture and incrementally added dropout layers, batch normalization layers, and data augmentation techniques.
- **Batch Size**: Set to 128 for all experiments.
- **Number of Epochs**: Varied based on model convergence; typically between 50 and 100 epochs.
- **Learning Rate Scheduling**: Used a step decay schedule, reducing the learning rate at specified epochs.

### 3.4. Training and Validation Splits

The data was split into an 80/20 Train/Test split of which, a further 80/20 Train/Validation split was created. Model checkpoints were created for those with the best validation accuracy.

## 4. Experimental Analysis and Results

### 4.1. System Information

```
CPU Information:
  CPU Name: 2000.148
  CPU Cores: 2

Memory Information:
  Total Memory: 12.67 GB
  Available Memory: 11.49 GB

GPU Information:
  GPU Name: Tesla T4
  GPU Memory Total: 15360.00 MB
```

### 4.2. Results

The best performing models for each architecture are presented in Table 1. Training and validation curves are shown in Figures 6, 7, and 5.

Table 1. Performance comparison of different models on CIFAR-10.

| Model | Test Accuracy | Loss |
|---|---|---|
| Custom CNN (Final) | 90.5% | 0.3646 |
| MobileNetV2 (Fine-tuned) | 70.53% | 0.8097 |
| AlexNet (From Scratch) | 60.11% | 1.3035 |
| ResNet152V2 (From Scratch) | 62.35% | 1.0441 |
| ResNet152V2 (Imagenet Weights) | 92.31% | 0.1159 |

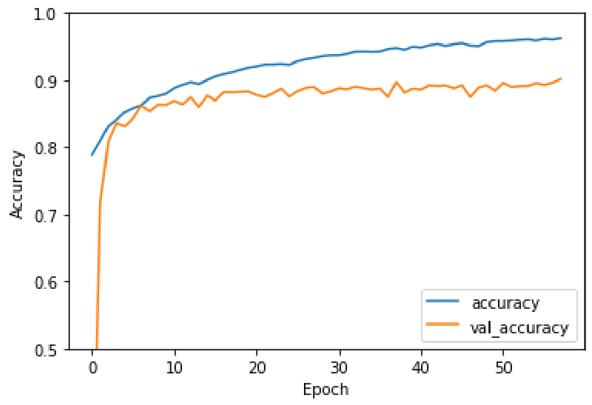


Figure 5. Training and validation accuracy for Custom CNN Model.

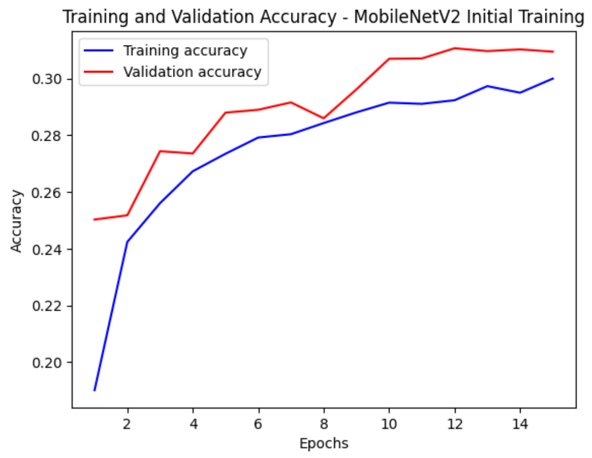


Figure 6. Training and validation accuracy for MobileNetV2 (Fine-tuned).

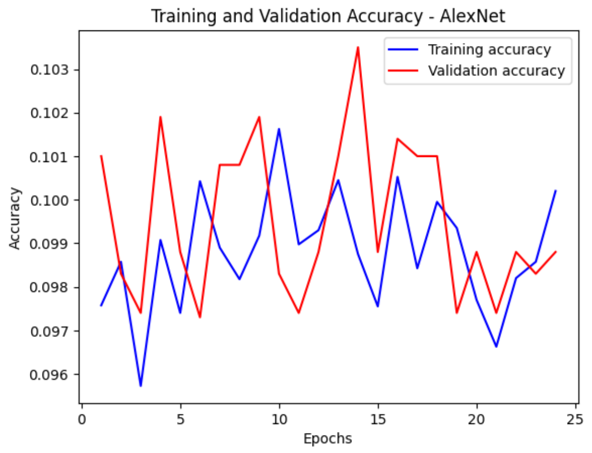


Figure 7. Training and validation accuracy for AlexNet (From Scratch).

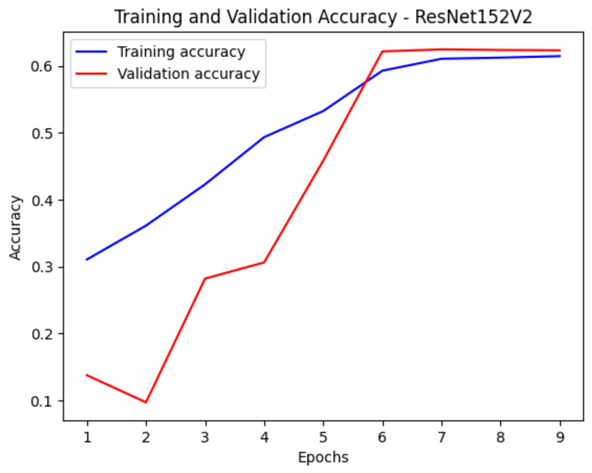


Figure 8. Training and validation accuracy for ResNet152V2 Model.

### 4.3. Analysis

#### 4.3.1 MobileNetV2 (Fine-tuned)

Due to a simple architecture more suited for edge devices MobileNetV2, performs poorly and only achieves a 70% accuracy [20].

#### 4.3.2 AlexNet (From Scratch)

The relatively lower performance of 60 % when compared to MobileNetV2 can be attributed to the lack of pre-trained weights. The model also isn't suited for training on smaller images of 32x32 when compared to its standard 227x227 [7].

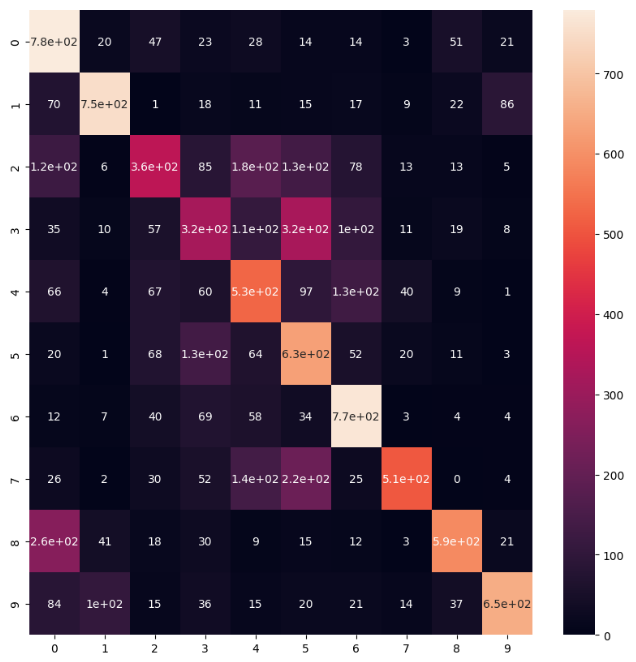


Figure 9. Confusion Matrix for the best performing model, i.e., ResNet152V2 [2]

#### 4.3.3 Custom CNN Model

The custom CNN model showed significant improvements with each incremental enhancement:

- **Baseline Model**: Test accuracy of 80.0%.
- **With Dropout**: Jumped to 83.0% [16].
- **With Batch Normalization**: Further improved to 86.0% [17].
- **With Data Augmentation**: Final test accuracy of 90.5% [19].

## 5. Reflection on Method Selection and Results

The use of pretrained models such as MobileNetV2 and ResNet152V2 with Imagenet weights has proven to be a worthwhile contender for comparing performance with standard models. The combination of retraining and fine-tuning has given the model the ability to generalise better. Hyperparameter tunings such as dynamic learning rates helped the model converge more optimally.

### 5.1. Neural Redshift at Play

The final decision to train on ResNet152V2 came about after observing the phenomenon of *Neural redshift*[13], wherein models with ¡5 Million parameters showed a low accuracy score, confirming that the architecture is not complex enough to capture the nuances of the patterns in CIFAR-10 dataset. This is further confirmed by the testing accuracy of models with ¿5 Million parameters such as ResNet152V2 and our custom CNN.

## 6. Conclusion

The experiment conludes that pretrained model can only leverage their weights when they match the original trainign image shape and number of output classes as is seen by scratch models performing better than pretrained fine-tuned ones.

The study also focuses on the importance of matching the complexity of the model architecture to that of the training data, as seen by the prevalence of the Neural Redhshift phenomenon.

## References

[1] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Technical Report*, University of Toronto, 2009.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 630–645, 2016.

[3] 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*, pages 770–778, 2016.

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[5] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, Inception-ResNet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[6] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] M. Abadi et al. TensorFlow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation*, pp. 265–283, 2016.

[12] A. Riffaud. Understanding Regularisation Techniques in Deep Learning. *Medium*, 2020. Available at: `https://medium.com/@alriffaud/understanding-regularisation-techniques-in-deep-learning-fa80185ee13e`.

[13] D. Teney, A. Nicolicioiu, V. Hartmann, and E. Abbasnejad. Neural Redshift: Random Networks are not Random Functions. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4786–4796, 2024. IEEE.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[15] T. DeVries and G. W. Taylor. Improved regularisation of convolutional neural networks with Cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, pp. 448–456, 2015.

[18] A. Riffaud. Understanding Regularization Techniques in Deep Learning. *Medium*, 2020. Available at: `https://medium.com/@alriffaud/understanding-regularization-techniques-in-deep-learning-fa80185ee13e`.

[19] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[20] M. Huh, P. Agrawal, and A. A. Efros. What makes ImageNet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[21] J. Deng et al. ImageNet: A large-scale hierarchical image database. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[22] S. Zagoruyko and N. Komodakis. Wide residual networks. *British Machine Vision Conference*, 2016.

## A. Appendix: Architectures of ResNet152V2 model and Custom CNN Model

Table 2. Architecture of ResNet152V2 model

| **Layer (type)** | **Output Shape** | **Param #** | **Connected to** |
|---|---|---|---|
| input_layer_2 (InputLayer) | (None, 32, 32, 3) | 0 | - |
| conv1_conv (Conv2D) | (None, 32, 32, 64) | 1,728 | input_layer_2[0][0] |
| conv1_bn (BatchNormalization) | (None, 32, 32, 64) | 256 | conv1_conv[0][0] |
| conv1_relu (Activation) | (None, 32, 32, 64) | 0 | conv1_bn[0][0] |
| pool1_pad (ZeroPadding2D) | (None, 34, 34, 64) | 0 | conv1_relu[0][0] |
| pool1_pool (MaxPooling2D) | (None, 16, 16, 64) | 0 | pool1_pad[0][0] |
| conv2_block1_preact_bn (BatchNormalization) | (None, 16, 16, 64) | 256 | pool1_pool[0][0] |
| conv2_block1_preact_relu (Activation) | (None, 16, 16, 64) | 0 | conv2_block1_preact_bn[0][0] |
| conv2_block1_1_conv (Conv2D) | (None, 16, 16, 64) | 4,096 | conv2_block1_preact_relu[0][0] |
| conv2_block1_1_bn (BatchNormalization) | (None, 16, 16, 64) | 256 | conv2_block1_1_conv[0][0] |
| conv2_block1_1_relu (Activation) | (None, 16, 16, 64) | 0 | conv2_block1_1_bn[0][0] |
| conv2_block1_2_pad (ZeroPadding2D) | (None, 18, 18, 64) | 0 | conv2_block1_1_relu[0][0] |
| conv2_block1_2_conv (Conv2D) | (None, 16, 16, 64) | 36,864 | conv2_block1_2_pad[0][0] |
| conv2_block1_2_bn (BatchNormalization) | (None, 16, 16, 64) | 256 | conv2_block1_2_conv[0][0] |
| conv2_block1_2_relu (Activation) | (None, 16, 16, 64) | 0 | conv2_block1_2_bn[0][0] |
| conv2_block1_0_conv (Conv2D) | (None, 16, 16, 256) | 16,384 | conv2_block1_preact_relu[0][0] |
| conv2_block1_3_conv (Conv2D) | (None, 16, 16, 256) | 16,384 | conv2_block1_2_relu[0][0] |
| conv2_block1_out (Add) | (None, 16, 16, 256) | 0 | conv2_block1_0_conv[0][0], conv2_block1_3_conv[0][0 |
| conv2_block2_preact_bn (BatchNormalization) | (None, 16, 16, 256) | 1,024 | conv2_block1_out[0][0] |
| conv2_block2_preact_relu (Activation) | (None, 16, 16, 256) | 0 | conv2_block2_preact_bn[0][0] |
| conv2_block2_1_conv (Conv2D) | (None, 16, 16, 64) | 16,384 | conv2_block2_preact_relu[0][0] |
| conv2_block2_2_conv (Conv2D) | (None, 16, 16, 64) | 36,864 | conv2_block2_1_conv[0][0] |
| conv2_block2_3_conv (Conv2D) | (None, 16, 16, 256) | 16,384 | conv2_block2_2_conv[0][0] |
| conv2_block2_out (Add) | (None, 16, 16, 256) | 0 | conv2_block1_out[0][0], conv2_block2_3_conv[0][0] |
| conv3_block1_preact_bn (BatchNormalization) | (None, 8, 8, 512) | 2,048 | conv2_block2_out[0][0] |
| conv3_block1_preact_relu (Activation) | (None, 8, 8, 512) | 0 | conv3_block1_preact_bn[0][0] |
| conv3_block1_1_conv (Conv2D) | (None, 8, 8, 128) | 65,536 | conv3_block1_preact_relu[0][0] |
| conv3_block1_2_pad (ZeroPadding2D) | (None, 10, 10, 128) | 0 | conv3_block1_1_conv[0][0] |
| conv3_block1_2_conv (Conv2D) | (None, 8, 8, 128) | 147,456 | conv3_block1_2_pad[0][0] |
| conv3_block1_3_conv (Conv2D) | (None, 8, 8, 512) | 65,536 | conv3_block1_2_conv[0][0] |
| conv3_block1_out (Add) | (None, 8, 8, 512) | 0 | conv2_block2_out[0][0], conv3_block1_3_conv[0][0] |
| avg_pool (GlobalAveragePooling2D) | (None, 512) | 0 | last Add layer |
| predictions (Dense) | (None, 10) | 5,130 | avg_pool |

Table 3. Detailed architecture of the Custom CNN model

| **Layer (type)** | **Output Shape** | **Param #** | **Connected to** |
|---|---|---|---|
| input_layer (InputLayer) | (None, 32, 32, 3) | 0 | - |
| conv2d_1 (Conv2D) | (None, 32, 32, 32) | 896 | input_layer[0][0] |
| batch_normalization_1 (BatchNormalization) | (None, 32, 32, 32) | 128 | conv2d_1[0][0] |
| activation_1 (Activation) | (None, 32, 32, 32) | 0 | batch_normalization_1[0][0] |
| max_pooling2d_1 (MaxPooling2D) | (None, 16, 16, 32) | 0 | activation_1[0][0] |
| conv2d_2 (Conv2D) | (None, 16, 16, 64) | 18,496 | max_pooling2d_1[0][0] |
| batch_normalization_2 (BatchNormalization) | (None, 16, 16, 64) | 256 | conv2d_2[0][0] |
| activation_2 (Activation) | (None, 16, 16, 64) | 0 | batch_normalization_2[0][0] |
| max_pooling2d_2 (MaxPooling2D) | (None, 8, 8, 64) | 0 | activation_2[0][0] |
| conv2d_3 (Conv2D) | (None, 8, 8, 128) | 73,856 | max_pooling2d_2[0][0] |
| batch_normalization_3 (BatchNormalization) | (None, 8, 8, 128) | 512 | conv2d_3[0][0] |
| activation_3 (Activation) | (None, 8, 8, 128) | 0 | batch_normalization_3[0][0] |
| max_pooling2d_3 (MaxPooling2D) | (None, 4, 4, 128) | 0 | activation_3[0][0] |
| flatten_1 (Flatten) | (None, 2048) | 0 | max_pooling2d_3[0][0] |
| dense_1 (Dense) | (None, 256) | 524,544 | flatten_1[0][0] |
| batch_normalization_4 (BatchNormalization) | (None, 256) | 1,024 | dense_1[0][0] |
| activation_4 (Activation) | (None, 256) | 0 | batch_normalization_4[0][0] |
| dropout_1 (Dropout) | (None, 256) | 0 | activation_4[0][0] |
| dense_2 (Dense) | (None, 10) | 2,570 | dropout_1[0][0] |

Table 4. Architecture of the MobileNetV2 model with Inverted Residual Blocks

| **Layer Type** | **Input Size** | **Output Size** | **Kernel Size** | **Stride** | **Expansion Factor** |
|---|---|---|---|---|---|
| Initial Conv | 224x224x3 | 112x112x32 | 3x3 | 2 | - |
| Inverted Residual Block | 112x112x32 | 112x112x16 | 3x3 | 1 | 1 |
| Inverted Residual Block x2 | 112x112x16 | 56x56x24 | 3x3 | 2 | 6 |
| Inverted Residual Block x3 | 56x56x24 | 28x28x32 | 3x3 | 2 | 6 |
| Inverted Residual Block x4 | 28x28x32 | 14x14x64 | 3x3 | 2 | 6 |
| Inverted Residual Block x3 | 14x14x64 | 14x14x96 | 3x3 | 1 | 6 |
| Inverted Residual Block x3 | 14x14x96 | 7x7x160 | 3x3 | 2 | 6 |
| Inverted Residual Block x1 | 7x7x160 | 7x7x320 | 3x3 | 1 | 6 |
| Final Conv | 7x7x320 | 7x7x1280 | 1x1 | 1 | - |
| Global Avg Pooling | 7x7x1280 | 1x1x1280 | - | - | - |
| Fully Connected | 1x1x1280 | 1x1x1000 | - | - | - |