**Name: Ziad Mohamed**

**SRN: 21010659**

**Analysis Report**

**Process**

1. **Data Loading and Splitting**

* Cifar-10 Dataset:

The dataset used for this project was the Cifar-10 dataset which includes 60,000 images split into 10 classes: Airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The images are evenly split across these 10 classes with each class including 6,000 images. Each image is colored and has a size of 32x32 pixels. By default, the dataset is split into 50,000 images for training and 10,000 for testing (Krizhevsky, 2009).

* Train-Test-Validation Split:

The dataset was initially split into training and validation data using the default Cifar-10 split which is approximately an 83-17 split. 50,000 images were initially loaded as training data and the remaining 10,000 images were stored for validation. A further 80-20 split was made to the 50,000 images in the training data so that we could have 10,000 images as testing data for our model. This left us with 40,000 images used for the training phase, 10,000 for validation, and another 10,000 for final testing (Krizhevsky, 2009).

Training data was used to train the model. Validation data was used to monitor the performance of the model so that we can tune its hyperparameters, allowing us to enhance the model and improve its overall efficacy in prediction. Once the model was trained and evaluated, the test data was used to evaluate the final performance of the model (Goodfellow et al., 2016).

1. **Data Preprocessing**

* **Grayscale Conversion:**

RGB images were converted to grayscale using OpenCV to simplify the model’s complexity and focus on the important features. The color of the images used in this project was deemed an unnecessary feature for the model to be trained on. This is because many of the images in this dataset, if not all, can be correctly classified regardless of their color (Khan et al., 2020).

* **Histogram Equalization:**

This technique was used to enhance the contrast of the images once they were converted into greyscale by evenly redistributing pixel intensity values, making features more distinguishable. This makes the feature extraction process more efficient during the training of the neural network and enhances the model’s performance (Pratt et al., 2020).

Mathematically, histogram equalization works by calculating the frequency of each pixel value within an image. After that the cumulative frequency is also calculated. Each pixel value is then equalized to a new value using this formula:

**A mathematical equation with numbers and symbols

Description automatically generated**

Where:

* + - cdf(v) is the cumulative frequency of v.
    - cdf min is the minimum cumulative frequency in the table.
    - M \* N is the number of pixels (M -> width, N -> height)
    - L is the number of gray levels, which is 256 in most cases.

Studies confirm that histogram equalization can enhance edge detection and improve overall model accuracy in computer vision tasks (Sharma et al., 2021).

* Min-Max Normalization:

Min-max normalization is a preprocessing technique where the range of pixel intensity values is rescaled to a fixed range (0-1). This is done to standardize the dataset, ensuring that all input features contribute equally to the model’s learning process. The formula for Min-Max Normalization is:

A black and white math equation

Description automatically generated

Where:

* + - 𝑥: The original pixel value.
    - 𝑥 min ​: The minimum pixel value in the dataset (e.g., 0 for 8-bit images).
    - 𝑥 max​: The maximum pixel value in the dataset (e.g., 255 for 8-bit images).
    - 𝑥 normalized​ : The normalized pixel value, which lies within the range [ 0 , 1 ].

For color images (RGB), the normalization is applied channel-wise.

Min-Max normalization improves the performance of gradient-based optimization algorithms used in neural networks, which are sensitive to the scale of the input data. Moreover, Min-Max Normalization prevents the dominance of larger feature values which can lead to suboptimal models (Jain & Patel, 2022).

* One-Hot Encoding:

One-Hot Encoding is a technique which is used to convert categorical data, like class labels, into a binary vector representation. This is very important in machine learning and deep learning where numerical values are required to process inputs effectively.

A benefit to using One-Hot Encoding is that it avoids implicit ordering of classes. This is because when you represent categorical labels with integers (e.g 0, 1, 2, …) the model might incorrectly infer an ordering or a relationship between the classes.

Furthermore, another advantage as to why One-Hot Encoding was used in this project is its compatibility with loss functions. In classification tasks, one-hot encoded vectors are compatible with loss functions like **categorical cross-entropy**.

Compatibility with loss functions like categorical cross-entropy in classification tasks is also another significant benefit provided by the One-Hot Encoding technique (Kumar et al., 2021).

1. **Data Augmentation**

* Image Data Generator:

As an extra step to preprocessing the training data, real-time augmentation was also applied to it to increase the data’s diversity:

* + - Rotation: Randomly rotates the image by up to 15 degrees.
    - Shifting: Randomly shifts images horizontally or vertically by up to 10%.
    - Horizontal Flip: Randomly flips images horizontally.

Data Augmentation helps prevent overfitting and allows for the model to generalize better to unseen data (Liu et al., 2020).

1. **Model Architecture**

This section of the report will detail the components and design principles of the **Convolutional Neural Network (CNN)** architecture which was used to classify the images of the Cifar-10 dataset.

**4. 1 Convolutional Neural Network (CNN)**

A **Convolutional Neural Network (CNN)** is a specialized neural network designed for processing structured, grid-like data, such as images. The architecture leverages spatial hierarchies to efficiently learn and extract features from images (LeCun et al., 2015; Simonyan & Zisserman, 2015).

**4.1.1 Key Components**

**1. Con2D Layers (Convolutional Layers):**

* **Purpose:** These layers are located at the forefront of the neural network with their purpose being to apply learnable filters (kernels) to the input image to detect spatial features such as edges, patterns, and textures (Krizhevsky et al., 2012; Tan & Le, 2021).
* **Working:** 
  + Each filter slides over the image and performs an element-wise multiplication followed by summation.
  + The result is a feature map highlighting the presence of specific features (Albawi et al., 2017).
* **Leaky-ReLU Activation:** 
  + The Leaky Rectified Linear Unit (ReLU) activation function introduces non-linearity, enabling the network to learn complex patterns (Maas et al., 2013).
  + Leaky ReLU is an activation function that modifies the standard ReLU by allowing a small, non-zero gradient for negative inputs, preventing the "dying ReLU" problem where neurons output zero for all inputs (Xu et al., 2020).

A black text on a white background

Description automatically generated

**2. Max Pooling**

MaxPooling is a type of pooling operation commonly used in Convolutional Neural Networks (CNNs) to downsample feature maps. It plays a critical role in reducing the spatial dimensions of the data while preserving the most important features (Scherer et al., 2010).

**Operation:**

* + Max pooling operates on small regions (windows) of the feature map, typically of size 2x2 or 3x3.
  + It slides the window across the feature map with a specific stride (step size), which is often 2.
  + For each region, it selects the maximum value and discards the rest (Zhou et al., 2021).

**Formula**:

**A black background with white text

Description automatically generated**

**Where**:

* **P(i,j) is the pooled value at position (i,j).**
* **R(i,j) is the set of values in the pooling region centered at (i,j).**

**Example**: Consider a 4x4 feature map:

**A number on a black background

Description automatically generated**

A number in a rectangle

Description automatically generatedApplying 2x2 Max Pooling with stride 2:

**Benefits of Max Pooling:**

* **Dimensionality Reduction:** It reduces the spatial size of feature maps which decreases the computational load for subsequent layers.
* **Preservation of Important Features:** The maximum value within each region typically represents the strongest activation (e.g edges, textures), helping the network focus on significant features.
* **Overfitting Reduction:** By reducing the number of parameters and computations within the network, MaxPooling may help mitigate overfitting (Springenberg et al., 2015).

**3. Batch Normalization:**

Batch Normalization is a regularization and optimization technique used in neural networks to standardize the outputs (activations) of each layer. Introduced by Sergey Ioffe and Christian Szegedy in their 2015 paper *"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,"* it has become a cornerstone of modern deep learning architectures (Ioffe & Szegedy, 2015).

**How Batch Normalization Works:**

1. **Normalization:**

* For each feature (neuron activation) in the batch, Batch Normalization normalizes the activations by subtracting them by the mean and dividing them by the standard deviation (Santurkar et al., 2018).

**A black background with white text

Description automatically generated**

**Where:**

* + - **x*i*​ :** Activation of the neuron for the 𝑖-th sample in the mini-batch.
    - **𝜇𝐵:** Mean of activations over the mini-batch**. 𝜎^2:** Variance of activations over the mini-batch**.**
    - **𝜖:** A small constant added for numerical stability**.**

1. **Learnable Parameters:**

* After normalization, Batch Normalization introduces two learnable parameters, 𝛾 (scale) and 𝛽 (shift), to allow the network to retain the flexibility to learn the optimal scale and offset for the activations:

**𝑦𝑖 = 𝛾 𝑥 ^ 𝑖 + 𝛽**

**Key Benefits of Batch Normalization:**

* **Accelerated Training:** Standardizing activations reduces the problem of internal covariate shift, where the distribution of activations changes as the network parameters update. This stabilization allows for faster convergence and enables the use of higher learning rates (Ioffe & Szegedy, 2015).
* **Regularization Effect:** By introducing noise into the network, Batch Normalization acts as a regularizer, reducing overfitting (Bjorck et al., 2018).
* **Improved Gradient Flow:** Normalized activations prevent the gradients from becoming too small or too large, addressing the exploding/vanishing gradient problem (Santurkar et al., 2018).

**4. Global Average Pooling 2D:**

GlobalAveragePooling2D (GAP) is a layer used in Convolutional Neural Networks (CNNs) to reduce the spatial dimensions of feature maps into a single vector. It acts as a replacement for fully connected layers, providing a more efficient and less parameter-heavy way to summarize spatial information (Lin et al., 2013).

This technique was used in place of the flattening technique in order to prevent overfitting and reduce the number of parameters passed to the dense layers.

**How Global Average Pooling Works:**

1. **Operation:** For each feature map (channel) in the input, GAP computes the average of all values across the spatial dimensions (width and height). This results in a single value per feature map.
2. **Input and Output Dimensions:**

* Input: H × W × C, where H, W, and C are the height, width, and number of channels in the feature map.
* Output: 1 × 1 × C (a single value for each channel).

**Why Use Global Average Pooling?**

1. **Dimensionality Reduction:** GAP significantly reduces the size of the feature map, converting spatial information into a compact representation.
2. **Prevention of Overfitting:** 
   * Fully connected layers (FC layers) have many parameters, which can lead to overfitting, especially with limited training data.
   * GAP eliminates the need for FC layers, reducing the number of parameters and, consequently, the risk of overfitting (Szegedy et al., 2015).

**5. Dense Layers**

Dense layers, also known as fully connected layers, are critical components in CNNs used for high-level pattern learning and classification (Goodfellow et al., 2016).

**How They Work:**

1. **Connections**:
   * Each neuron in a dense layer is connected to every neuron in the previous layer, enabling the network to aggregate and learn global features.
2. **Transformations**:
   * The layer computes: y=f(Wx+b)

Where:

* + - x is the Input vector.
    - W is the weight matrix.
    - B is the Bias vector.
    - F is the activation function used (e.g., ReLU, softmax).

**Purpose:**

* After convolutional layers extract spatial features, dense layers process these features to learn complex, high-level patterns and perform the final classification task.
* Dense layers are typically used in the last stages of CNNs to bridge the gap between the extracted features and output predictions.

**Benefits of Using Dense Layers:**

* Combines all the learned features for global decision making
* Outputs class probabilities using activation functions like **softmax** in the final layer.

**6. Dropout**

Dropout is a regularization technique used in neural networks to reduce overfitting by randomly deactivating (dropping) a fraction of neurons during training (Srivastava et al., 2014)..

**How It Works:**

* During each training iteration, a specified proportion (p) of neurons in a layer are set to zero (dropped).
* This forces the network to rely on multiple pathways to learn features, improving its generalization ability.

**Purpose:**

* Prevent over-reliance on specific neurons.
* Encourage redundancy and robustness in feature learning.

**Key Benefits:**

1. Reduces overfitting by regularizing the model.
2. Improves performance on unseen data by ensuring the network learns diverse feature representations.
3. **Regularization**

L2 Regularization is a technique used to reduce overfitting by penalizing large weights in the network, encouraging simpler and more generalizable models. A regularization term of 0.001-0.003 was used within some of the layers of the model.

**How It Works:**

Adds a penalty term to the loss function proportional to the square of the weights:

A black background with white text

Description automatically generated

Where:

* + Ltotal​ is the regularized loss.
  + Loriginal is the original loss (e.g., categorical cross-entropy).
  + λ is the regularization strength (controls the penalty's magnitude).
  + wi is the weights of the model.

**Purpose:**

* Prevents the model from assigning overly high importance to specific features by discouraging large weight values.
* Helps avoid overfitting to the training data, improving generalization.

**Key Benefits:**

1. Encourages smoother weight distributions, leading to simpler models.
2. Reduces the risk of overfitting, especially in large networks with many parameters.
3. Works well with dense and convolutional layers.
4. **Training Optimizations**

Training optimizations were critical for improving the efficiency and performance of this model. Several strategies were employed to ensure effective training and prevent overfitting.

**6.1 Early Stopping**

* **Purpose**: Prevents the model from overfitting to the training data by monitoring validation loss during training.
* **How It Works**:
  + Training is halted if no improvement in validation loss is observed for 25 consecutive epochs.
  + This avoids wasting computational resources on further training when the model is no longer improving.
* **Benefit**: Ensures the model retains its best performance on unseen data without unnecessary epochs.

**6.2 Model Checkpointing**

* **Purpose**: Safeguards the best model weights based on validation accuracy during training.
* **How It Works**:
  + At the end of each epoch, the model's validation accuracy is evaluated.
  + If the current epoch achieves the highest validation accuracy so far, the model weights are saved.
* **Benefit**: Guarantees that the final evaluation uses the best-performing model, even if training results in overfitting in later epochs.

These optimizations collectively improve training efficiency, enhance generalization, and ensure the model's best version is preserved for evaluation.

1. **Common Challenges and Solutions**
   * 1. **Overfitting**

* **Issue**: The model performed well on training data but struggled to generalize to unseen data.
* **Solutions**:
  + **Dropout**: Randomly deactivated neurons during training to prevent over-reliance on specific features.
  + **L2 Regularization**: Penalized large weights, encouraging simpler models.
  + **Data Augmentation**: Enhanced training data diversity to improve generalization.
  + **Early Stopping**: Halted training when validation loss stopped improving to avoid overfitting.

**2. Plateaued Validation Accuracy**

* **Issue**: Validation accuracy stagnated after initial improvements.
* **Solutions**:
  + Adjusted regularization techniques and preprocessing methods to optimize feature extraction. This involved place a light regularization term on the initial features so they can capture the basic features faster while having a heavier regularization term placed on the final layers which are responsible for extracting the more complex features within an image.
  + Dropout rates were also tuned so that earlier layers had lower rates compared to the latter layers. This helped reduce regularization in the earlier layers, allowing the model to fit the training data better. Once the parameters increased with the increasing filters and number of layers, the dropout rates were also increased so that the model would not overfit the data.
  + Another solution that was experimented with to increase the plateaued validation accuracy is to reduce the batch size from 64 to 32. This acted as an extra form of regularization by adding some noise to the training process. This can help the model escape local minima and reduce overfitting by preventing the model from converging too tightly to the training data.

1. **Metrics and Evaluation**

The performance of the trained model was evaluated using several metrics and techniques to assess its accuracy, generalization, and classification capabilities:

**1. Overall Accuracy**

* The model achieved a test accuracy of 86.47%, which indicates its ability to correctly classify unseen images in the CIFAR-10 dataset.

**2. Classification Report**

* A detailed classification report was generated, showing metrics such as:
  + **Precision**: The proportion of correctly predicted instances among the total predicted instances for each class.
  + **Recall**: The proportion of correctly predicted instances among the total actual instances for each class.
  + **F1-Score**: The harmonic mean of precision and recall, balancing the two.
  + **Support**: The number of samples for each class.

A screenshot of a computer screen

Description automatically generated

**3. Confusion Matrix**

The confusion matrix provides a visual summary of the model's performance across all classes, showing true positives, false positives, and false negatives.

The heatmap below highlights areas of strong and weak performance for each class:

A graph with blue squares

Description automatically generated

**4. Class-Wise Accuracy**

The accuracy for each class was calculated based on the confusion matrix, revealing the model’s performance across different categories.

A graph of blue bars

Description automatically generated

**References**

* Albawi, S., Mohammed, T.A. and Al-Zawi, S. (2017) ‘Understanding of a convolutional neural network’, *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6.
* Bjorck, N., Gomes, C.P., Selman, B. and Weinberger, K.Q. (2018) ‘Understanding Batch Normalization’, *Advances in Neural Information Processing Systems*, 31.
* Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*. MIT Press.
* Ioffe, S. and Szegedy, C. (2015) ‘Batch normalization: Accelerating deep network training by reducing internal covariate shift’, *arXiv preprint arXiv:1502.03167*.
* Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012) ‘ImageNet classification with deep convolutional neural networks’, *Advances in Neural Information Processing Systems*, 25, pp. 1097–1105.
* Lin, M., Chen, Q. and Yan, S. (2013) ‘Network in network’, *arXiv preprint arXiv:1312.4400*.
* Maas, A.L., Hannun, A.Y. and Ng, A.Y. (2013) ‘Rectifier nonlinearities improve neural network acoustic models’, *Proceedings of the 30th International Conference on Machine Learning (ICML)*.
* Santurkar, S., Tsipras, D., Ilyas, A. and Madry, A. (2018) ‘How does Batch Normalization help optimization?’, *Advances in Neural Information Processing Systems*, 31, pp. 2483–2493.
* Scherer, D., Muller, A. and Behnke, S. (2010) ‘Evaluation of pooling operations in convolutional architectures for object recognition’, *International Conference on Artificial Neural Networks*, pp. 92–101.
* Simonyan, K. and Zisserman, A. (2015) ‘Very deep convolutional networks for large-scale image recognition’, *arXiv preprint arXiv:1409.1556*.
* Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) ‘Dropout: A simple way to prevent neural networks from overfitting’, *The Journal of Machine Learning Research*, 15(1), pp. 1929–1958.
* Szegedy, C., Liu, W., Jia, Y. et al. (2015) ‘Going deeper with convolutions’, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9.