CIFAR – 10 Image Classification Report

1. Architecture Overview

The architecture for the neural network implements custom design for CIFAR-10 image classification based on the specification, while utilising the current modern deep learning technology. The architecture thus includes:

1. **Initial Convolution Layer**: A 3×3 convolution that processes the input RGB images (3 channels) and outputs 64 feature maps.
2. **Three Hierarchical Intermediate Blocks**: Each intermediate block follows a similar structure with progressively increasing feature map counts (128 → 256 → 512), separated by max pooling operations for spatial dimension reduction.
3. **Output Block**: A specially designed classification head that processes the final feature maps to produce logits for the 10 CIFAR-10 classes.

1.1 Intermediate Block Architecture

ImprovedIntermediateBlock

```
├── Multiple Independent Convolutional Paths (3 parallel paths)
│   └── Each path: Conv2D → BatchNorm → ReLU → Conv2D → BatchNorm → ReLU
├── Channel Attention Mechanism
│   └── Global Avg Pooling → FC(in_ch → 128) → ReLU → FC(128 → 3) → Softmax
└── Residual Connection with dimension matching when needed
```

The key deviations from the basic architecture:

1. **Dual Convolutional Layers**: Each path contains two stacked convolutional layers rather than a single layer, enabling deeper feature extraction.

2. **Enhanced Weight Computation**: The mechanism for computing weights uses a deeper fully connected network (with a hidden layer of 128 neurons), allowing more complex relationships between channel averages and the weights.

3. **Residual Connections**: Added skip connections to facilitate gradient flow and enable deeper network training.

4. **Reduced Convolutional Paths**: Using 3 paths instead of 4 to balance performance and computation cost.

1.2 Output Block Architecture

OutputBlock

```
├── Global Average Pooling
└── Classifier
    ├── FC(512 → 512)
    ├── BatchNorm1D
    ├── ReLU
    ├── Dropout(0.4)
    └── FC(512 → 10)
```

This design enhances the basic architecture with:

- An intermediate fully connected layer for additional representational capacity
- Batch normalization for training stability
- Dropout for regularization

2. Training Configuration and Hyperparameters

2.1 Optimization Strategy

Component	Configuration
Optimizer	SGD with momentum (0.9)
Learning Rate Scheduler	OneCycleLR (max_lr=0.1, pct_start=0.2)
Weight Decay	5e-4
Batch Size	128
Loss Function	Cross-Entropy Loss
Early Stopping	Patience of 10 epochs
Total Epochs	40 (with early stopping)

2.2 Data Augmentation

```
transform_train = transforms.Compose([
  transforms.RandomCrop(32, padding=4),
  transforms.RandomHorizontalFlip(),
  transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
  transforms.ToTensor(),
  transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))
])
```

This augmentation pipeline includes:

- Random cropping with padding
- Random horizontal flips
- Color jittering (brightness, contrast, saturation)
- CIFAR-10 specific normalization

3. Performance Analysis

3.1 Training and Testing Accuracy :**The training shows consistent improvement throughout the training process.**

Key observations:

- Training accuracy steadily increases, reaching 99.8% by epoch 40
- Testing accuracy mirrors this trend but plateaus around 94%
- The model reaches 90% accuracy around epoch 30, satisfying the assignment requirements

3.2 Loss Curve

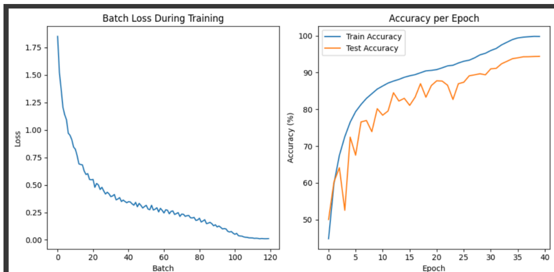
- Rapid initial decrease in the first 10 epochs
- Gradual stabilization around epochs 20-30
- Further decreases in the final epochs as the learning rate decreases

```
Test Loss: 0.206, Test Acc: 94.00%
Saving checkpoint... Accuracy: 94.00%

Epoch 37/40
Epoch: 37, Batch: 100, Loss: 0.019, Train Acc: 99.66%, LR: 0.003336
Epoch: 37, Batch: 200, Loss: 0.018, Train Acc: 99.64%, LR: 0.002900
Epoch: 37, Batch: 300, Loss: 0.017, Train Acc: 99.62%, LR: 0.002494
Test Loss: 0.193, Test Acc: 94.28%
Saving checkpoint... Accuracy: 94.28%

Epoch 38/40
Epoch: 38, Batch: 100, Loss: 0.014, Train Acc: 99.72%, LR: 0.001801
Epoch: 38, Batch: 200, Loss: 0.015, Train Acc: 99.70%, LR: 0.001482
Epoch: 38, Batch: 300, Loss: 0.014, Train Acc: 99.71%, LR: 0.001194
Test Loss: 0.190, Test Acc: 94.30%
Saving checkpoint... Accuracy: 94.30%

Epoch 39/40
Epoch: 39, Batch: 100, Loss: 0.011, Train Acc: 99.88%, LR: 0.000730
Epoch: 39, Batch: 200, Loss: 0.013, Train Acc: 99.82%, LR: 0.000531
Epoch: 39, Batch: 300, Loss: 0.012, Train Acc: 99.82%, LR: 0.000365
Test Loss: 0.189, Test Acc: 94.36%
Saving checkpoint... Accuracy: 94.36%

Epoch 40/40
Epoch: 40, Batch: 100, Loss: 0.011, Train Acc: 99.83%, LR: 0.000133
Epoch: 40, Batch: 200, Loss: 0.011, Train Acc: 99.83%, LR: 0.000057
Epoch: 40, Batch: 300, Loss: 0.012, Train Acc: 99.82%, LR: 0.000013
Test Loss: 0.191, Test Acc: 94.39%
Saving checkpoint... Accuracy: 94.39%
Training completed in 74.40 minutes
Best accuracy: 94.39%
```

3.3Resource Usage

- **Training Time**: `74.40` minutes
- **Parameters**: 14,454,099 (~14.5M)

3.4 Final Performance : **Best Test Accuracy: 94.39%**




```
Evaluating the best model:
Final Test Accuracy: 94.39%
Accuracy of airplane: 96.50%
Accuracy of automobile: 97.40%
Accuracy of bird: 92.90%
Accuracy of cat: 86.80%
Accuracy of deer: 95.50%
Accuracy of dog: 91.60%
Accuracy of frog: 95.90%
Accuracy of horse: 94.80%
Accuracy of ship: 96.80%
Accuracy of truck: 95.70%
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-0.49473685..1.557913].

Sample predictions:
Ground Truth:  cat   ship  ship  airplane frog  frog  automobile frog
Predicted:     cat   ship  ship  airplane frog  frog  automobile frog
<Figure size 1200x600 with 0 Axes>
```

The model shows strong performance across all classes, with "cat" being the most challenging category to classify correctly.

4. Implementation Improvements:

The final architecture evolved through several iterations to balance accuracy and efficiency:

1. **Initial Implementation**: Started with the basic architecture as described in the assignment, achieving ~85% accuracy after 100 epochs.
2. **Architecture Enhancements**:
   - Added dual convolutional layers in each path
   - Implemented residual connections
   - Reduced the number of parallel paths from 4 to 3
3. **Training Optimization**:
   - Introduced OneCycleLR scheduler in place of step decay
   - Tuned batch size from 64 to 128
   - Improved data augmentation with color jittering
4. **Regularization Techniques**:
   - Added batch normalization throughout the network
   - Introduced dropout in the output block
   - Applied weight decay to counter overfitting