Report on SVHN Dataset Classification Using VGG-like Neural Network

Team members: 1. Gu Shunshun, DC228002 2. Yang Zheyu, DC127235

Online code link:
https://colab.research.google.com/drive/1TMlMDdQqoA0nYKPuVDZWTgc-56BCzE2z?usp=sharing

1. Methodology

1.1 Data Processing

- **Dataset**: SVHN (Street View House Numbers) dataset.
- **Data Augmentation**: To improve the model's generalization, we applied several data augmentation techniques using the Albumentations library:
  - **Rotation**: Randomly rotated images within a limit of 30 degrees.
  - **Random Resized Crop**: Resized and cropped images to 32x32 pixels with a scale range of 0.8 to 1.0.
  - **Aspect Ratio Change**: Alter the aspect ratio with a range of 0.75 to 1.33.
  - **Normalization**: Normalized images using the mean and standard deviation of the SVHN dataset.
  - **ToTensorV2**: Converted images to PyTorch tensors.

1.2 Model Architecture

- **Convolutional Layers**: The model consists of three blocks of convolutional layers, each followed by batch normalization and ReLU activations, and max pooling layers. The layers are configured as follows:
  - Block 1: Two convolutional layers with 8 and 16 filters, respectively.
  - Block 2: Two convolutional layers with 32 filters each.
  - Block 3: Two convolutional layers with 32 filters each.
- **Fully Connected Layers**: After flattening the output from the convolutional layers, the model has two fully connected layers:
  - First layer: 256 units with ReLU activation and a dropout layer with a dropout rate of 20%.
  - Second layer: 10 units corresponding to the 10 classes in the SVHN dataset.

1.3 Training Procedures

- **Optimizer**: Both Adam and SGD optimizers.
- **Learning Rate**: Tested learning rates of 0.001 and 0.0001.
- **Batch Size**: Used batch sizes of 32 and 64.
- **Epoch**: Models were trained for 10 and 20 epochs.
- **Loss Function**: Cross-entropy loss was used for training the model.
- **Training Loop**: The training loop included forward pass, loss computation, backpropagation, and optimizer step.

1.4 Evaluation Methods

- **Metrics**: Evaluated the model using test loss, accuracy, and ROC AUC scores (both macro and micro).
- **ROC AUC Calculation**: Used one-vs-rest (OVR) strategy for multi-class ROC AUC calculation.

2. Results

2.1 Quantitative Results

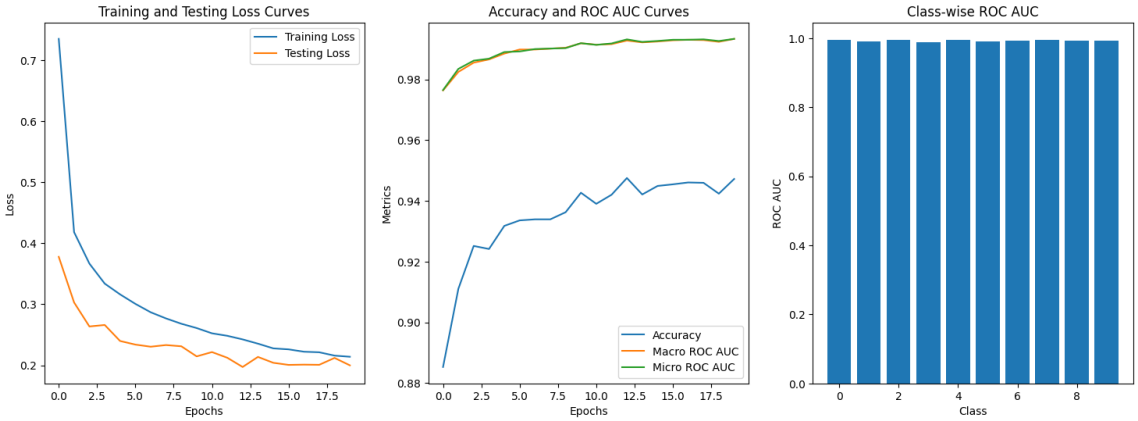
The table below summarizes the results of different training configurations:

Learning Rate	Batch Size	Num Epochs	Optimizer	Rotate	Scale Min	Scale Max	Final Train Loss	Final Test Loss	Final Accuracy	Final Macro ROC AUC	Final Micro ROC AUC
0.001	64	20	adam	30	0.8	1.0	0.2141	0.2000	0.9472	0.9933	0.9933
0.0001	64	20	adam	30	0.8	1.0	0.2621	0.2251	0.9369	0.9890	0.9896
0.001	32	20	adam	30	0.8	1.0	0.2173	0.2063	0.9451	0.9945	0.9946
0.001	64	10	adam	30	0.8	1.0	0.2585	0.2192	0.9380	0.9918	0.9916
0.001	64	20	sgd	30	0.8	1.0	0.2460	0.2112	0.9407	0.9878	0.9884
0.001	64	20	adam	15	0.9	1.0	0.1504	0.2105	0.9456	0.9930	0.9932

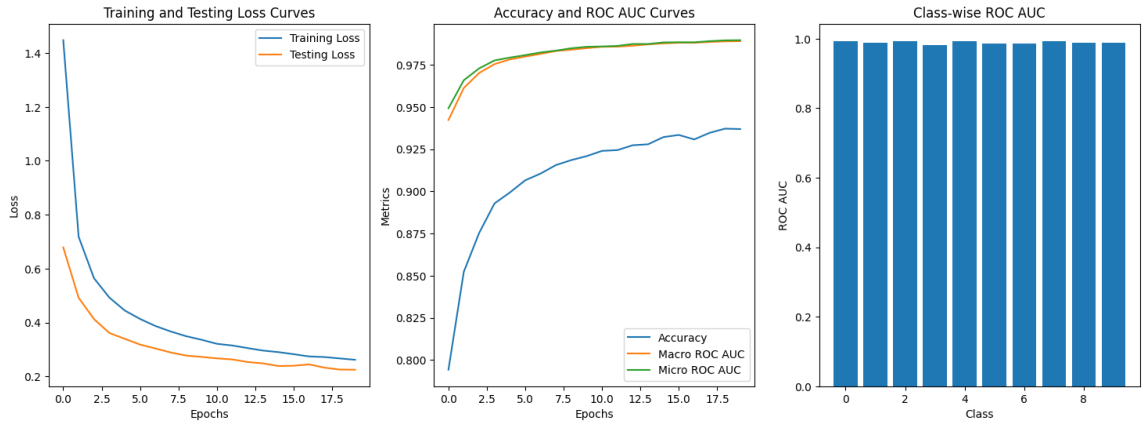
2.2 Visualizations

To better understand the performance of different configurations, here are some visualizations:

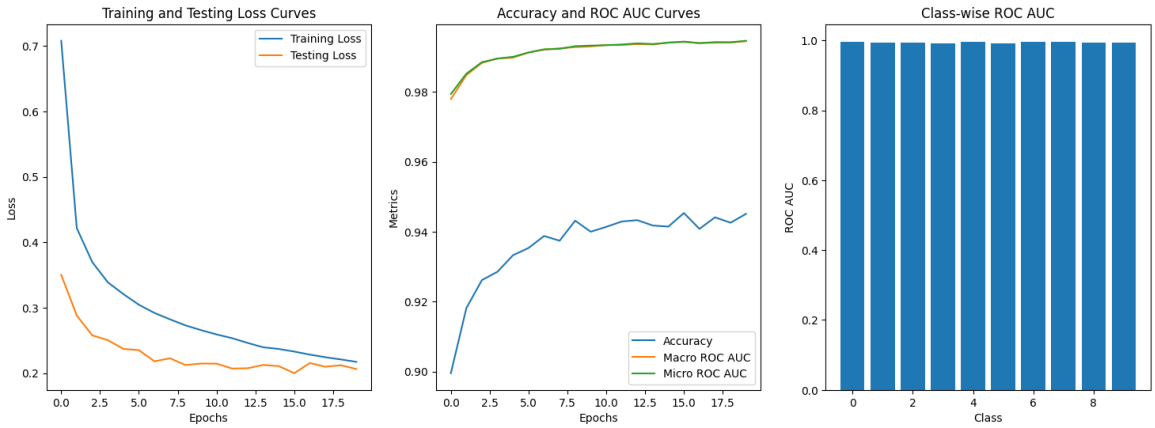
Experiment: {'learning_rate': 0.001, 'batch_size': 64, 'num_epochs': 20, 'optimizer_type': 'adam', 'augmentation_params': {'rotate': 30, 'scale_min': 0.8, 'scale_max': 1.0, 'min_ratio': 0.75, 'max_ratio': 1.33}}



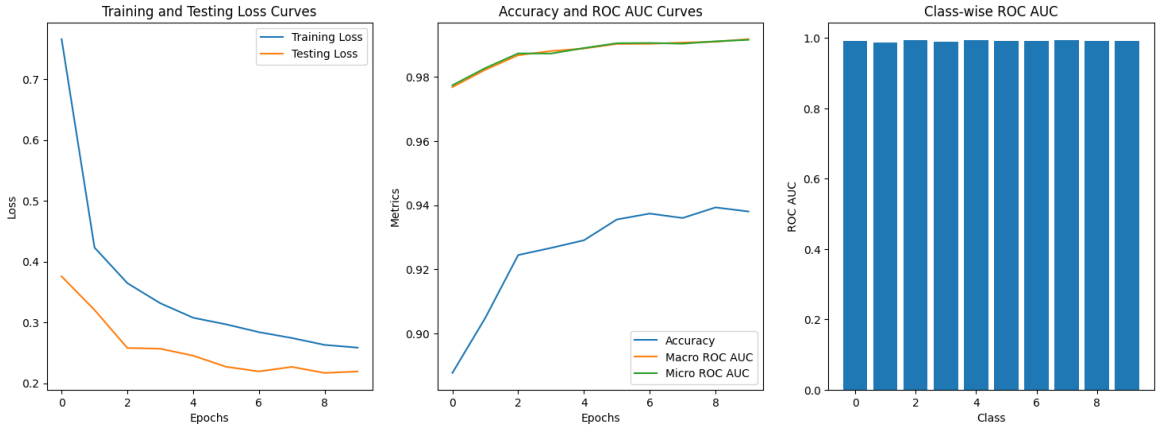
Experiment: {'learning_rate': 0.0001, 'batch_size': 64, 'num_epochs': 20, 'optimizer_type': 'adam', 'augmentation_params': {'rotate': 30, 'scale_min': 0.8, 'scale_max': 1.0, 'min_ratio': 0.75, 'max_ratio': 1.33}}



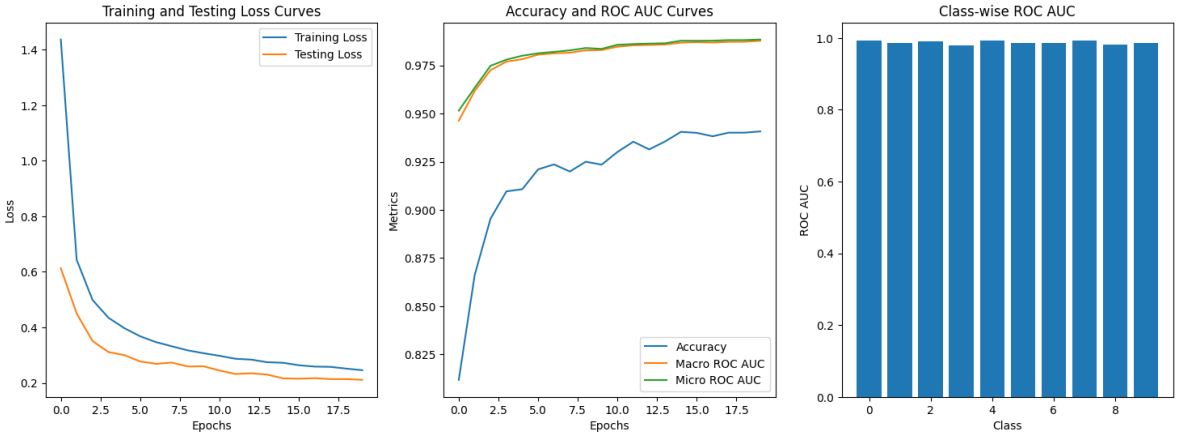
Experiment: {'learning_rate': 0.001, 'batch_size': 32, 'num_epochs': 20, 'optimizer_type': 'adam', 'augmentation_params': {'rotate': 30, 'scale_min': 0.8, 'scale_max': 1.0, 'min_ratio': 0.75, 'max_ratio': 1.33}}



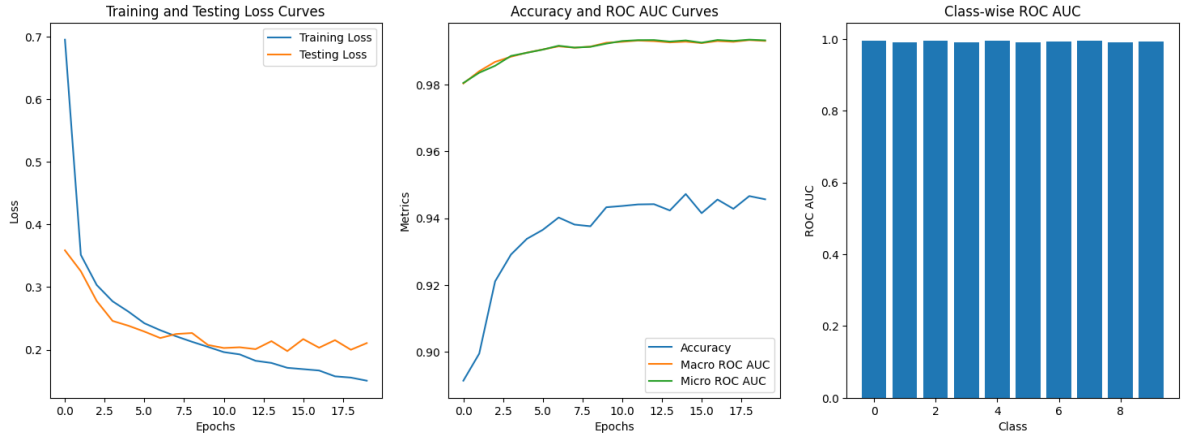
Experiment: {'learning_rate': 0.001, 'batch_size': 64, 'num_epochs': 10, 'optimizer_type': 'adam', 'augmentation_params': {'rotate': 30, 'scale_min': 0.8, 'scale_max': 1.0, 'min_ratio': 0.75, 'max_ratio': 1.33}}



Experiment: {'learning_rate': 0.001, 'batch_size': 64, 'num_epochs': 20, 'optimizer_type': 'sgd', 'augmentation_params': {'rotate': 30, 'scale_min': 0.8, 'scale_max': 1.0, 'min_ratio': 0.75, 'max_ratio': 1.33}}



Experiment: {'learning_rate': 0.001, 'batch_size': 64, 'num_epochs': 20, 'optimizer_type': 'adam', 'augmentation_params': {'rotate': 15, 'scale_min': 0.9, 'scale_max': 1.0, 'min_ratio': 0.85, 'max_ratio': 1.23}}



3. Discussion

3.1 Interpretation of Results

The results from the different training configurations reveal several insights into how hyperparameters affect model performance:

- **Learning Rate**:

    - A learning rate of 0.001 generally provided better results compared to 0.0001. This is likely because a higher learning rate allows the model to converge faster and avoid local minima. But a learning rate of 0.0001 can make the training more stable.
- **Batch Size**:
    - A batch size of 64 outperformed a batch size of 32. This is likely because that larger batch sizes provide more stable gradient estimates, which can lead to more efficient training and better generalization.
- **Optimizer**:
    - The Adam optimizer outperformed SGD in terms of both accuracy and loss, likely due to its adaptive learning rate capabilities, which handle sparse gradients and noisy data more effectively.
- **Data Augmentation**:
    - The rotation limit and scale range also impacted performance. A rotation limit of 30 degrees, a scale range of 0.9-1.0 and a ratio range of 0.75-1.33 yielded the better result. This suggests that enough augmentation helps the model generalize better by providing varied training examples without distorting the images too much.

3.2 Challenges Faced During Training

- **Overfitting**:
    - One of the challenges was overfitting, especially with larger batch sizes and longer training epochs. This was addressed by using batch normalization in the convolutional layers and dropout layers in the fully connected layers, which helped to regularize the model and prevent overfitting.
- **Optimizer Selection**:
    - Choosing the right optimizer was crucial. Initial experiments with SGD showed poor performance, which led to the adoption of the Adam optimizer. Adam's adaptive learning rate capabilities helped in improving better convergence and performance.
- **Data Augmentation**:
    - Finding the right balance in data augmentation was another challenge. Too much augmentation introduced noise, while too little did not provide enough variability. This was addressed by experimenting with different augmentation parameters and selecting the ones that provided the best validation performance.
- **Computational Resources**:
    - Initially, using Mac to train the model was extremely slow. However, switching to GPU-based training improved the speed by almost 60 times.

4. Conclusion

4.1 Key Findings

- **Optimal Hyperparameters**: The best performance was achieved with a learning rate of 0.001, batch size of 64, and the Adam optimizer. This configuration yielded the almost highest accuracy ((94.72%) and ROC AUC scores, along with the lowest testing loss.
- **Data Augmentation**: Appropriate data augmentation, specifically a rotation limit of 30 degrees and a scale range of 0.8-1.0, and a ratio range of 0.75-1.33 improved model generalization without introducing excessive noise.
- **Optimizer Choice**: The Adam optimizer outperformed SGD, highlighting the importance of adaptive learning rates in handling sparse gradients and noisy data.

4.2 Possible Improvements

- **Model Architecture**: Experimenting with deeper architectures like ResNet or DenseNet may potentially improve performance by capturing more complex features.
- **Advanced Data Augmentation**: Implementing more sophisticated augmentation techniques such as *CutMix* or *MixUp* could further enhance the model's robustness.
- **Attention Mechanisms**: Incorporate attention mechanisms like *SE (Squeeze-and-Excitation) blocks* or *self-attention layers* to allow the model to focus on important features in the images.

.