NYCU Introduction to Machine Learning, Homework 4

[111550196], [狄豪飛]

Part. 1, Kaggle (70% [50% comes from the competition]):

(10%) Implementation Details

I implemented a facial emotion recognition model based on the architecture described in "Facial Emotion Recognition: State of the Art Performance on FER2013" paper. The implementation focuses on achieving robust emotion classification while maintaining model efficiency.

Model Architecture Details:
- Four convolutional stages, each containing:
  * Two convolutional blocks (Conv2D + BatchNorm + ReLU)
  * Max pooling layer (kernel_size=2, stride=2)
- Three fully connected layers with dimensions: 4096 -> 4096 -> 7
- Adaptive average pooling before the classifier to ensure consistent dimensions
- Input size: 48x48 grayscale images

Training Strategy:
- Optimizer: SGD with Nesterov momentum (0.9) for better convergence
- Weight decay: 0.0001 to prevent overfitting
- Initial learning rate: 0.01, dynamically adjusted during training
- Learning rate scheduler: ReduceLROnPlateau
  * Reduction factor: 0.75
  * Patience: 5 epochs
  * Mode: max (monitoring validation accuracy)
- Training duration: 300 epochs
- Early stopping: Save best model based on validation accuracy

Data Processing and Augmentation:
- Extensive augmentation pipeline to improve model generalization:
  * Random rescaling: ±20% of original size
  * Random shifting: ±20% horizontally and vertically
  * Random rotation: ±10 degrees
  * Ten-crop strategy: 40x40 pixels
  * Random erasing with 50% probability
- Normalization: Pixel values divided by 255
- Batch processing: Images processed in batches of 64

Model backbone (e.g., VGG16, VGG19, Custom, etc)	Custom VGG architecture 4 convolutional stages 3 fully connected layers
Number of model parameters	40,376,519
Other hyperparameters …	Batch size: 64 Dropout rate: 0.1 Loss function: Cross-entropy BatchNorm momentum: 0.1 Input shape: 48x48x1 (grayscale)

(10%) Experimental Results
I analyzed the training progression through multiple metrics, observing distinct phases in the model's learning process:

1. Training Accuracy Progression:
Early Phase (Epochs 1-60):
- Epoch 1: 29.58%
- Epoch 20: 66.70%
- Epoch 40: 77.81%
- Epoch 60: 85.37%

Middle Phase (Epochs 61-160):
- Epoch 80: 89.72%
- Epoch 100: 91.80%
- Epoch 140: 93.90%
- Epoch 160: 94.39%

Final Phase (Epochs 161-300):
- Epoch 190: 95.57%
- Epoch 230: 96.70%
- Epoch 268: 97.21%
- Epoch 291: 97.59%

2. Loss Convergence:
Early Phase:
- Initial loss (Epoch 1): 1.72
- Rapid decrease to 0.881 (Epoch 20)
- Further reduction to 0.398 (Epoch 60)

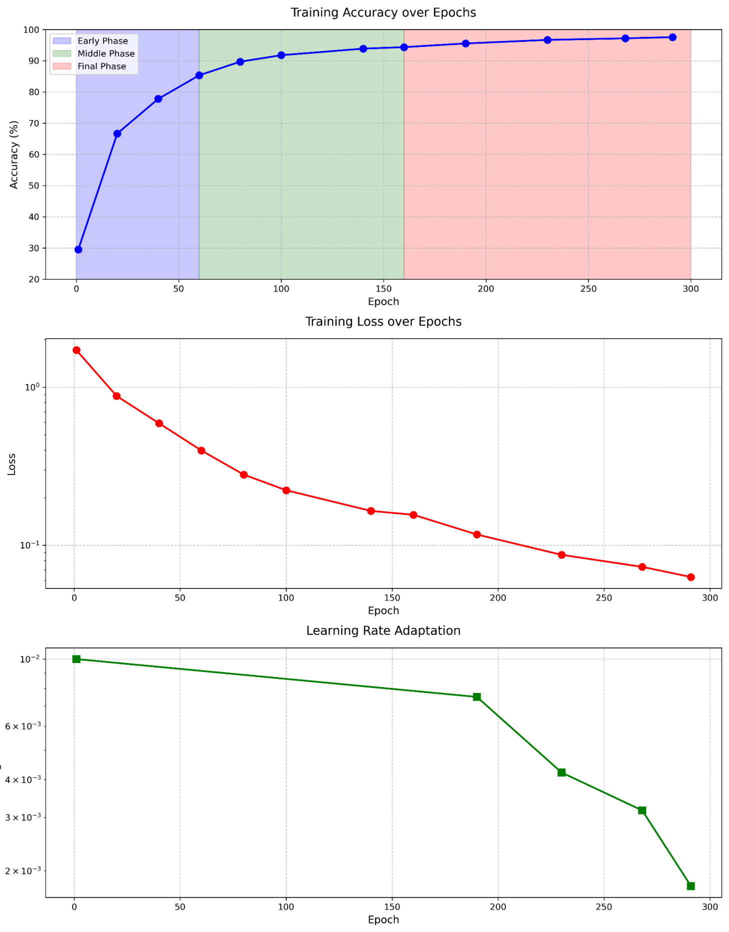
Middle Phase:
- Steady decrease to 0.223 (Epoch 100)
- Continued improvement to 0.156 (Epoch 160)

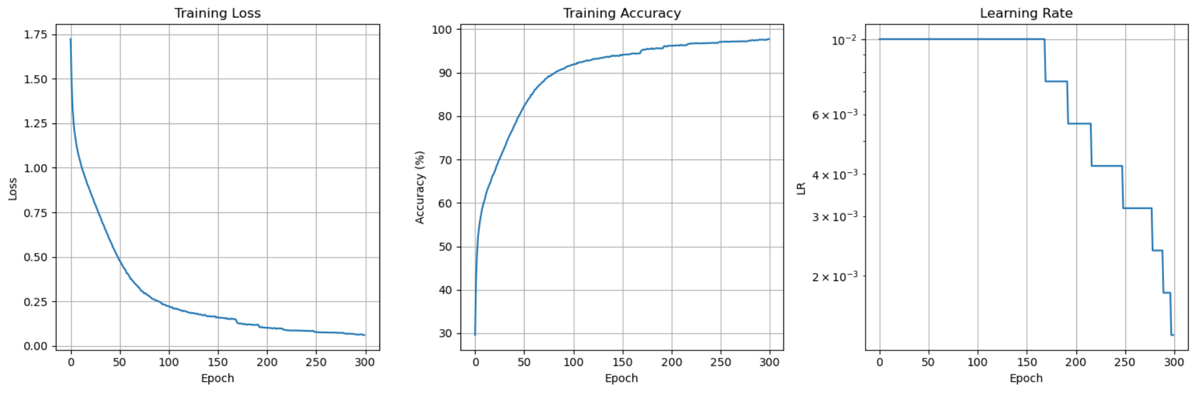
Final Phase:
- Fine-tuning loss to 0.117 (Epoch 190)
- Final convergence at 0.063 (Epoch 291)

3. Learning Rate Adaptation:
The learning rate showed adaptive behavior based on model performance:
- Initial rate: 0.010000 (Epochs 1-180)
- First reduction: 0.007500 (Around Epoch 190)

- Further reduction: 0.004219 (Around Epoch 230)
- Fine-tuning: 0.003164 (Around Epoch 268)
- Final stage: 0.001780 (Around Epoch 291)

Key Observations:
1. The model showed consistent improvement throughout training, with three distinct learning phases
2. The learning rate adaptation helped maintain steady progress in later epochs
3. Loss decrease correlated well with accuracy improvements
4. The final epochs showed continued improvement despite very small learning rates
5. The model achieved stable training with minimal oscillations in the final phase





Additional Kaggle Performance Analysis:
I evaluated multiple model checkpoints on Kaggle's test set, revealing interesting insights about model generalization:

Public Leaderboard Scores:
- Different checkpoints yielded varying results:
  * 0.60045
  * 0.60272
  * 0.60726
  * 0.61861
  * 0.61975
  * 0.62031

Key Observations from Kaggle Submissions:
1. Score Variation: Despite higher training accuracy in later epochs, Kaggle scores show moderate fluctuation (range: 0.60045 - 0.62031)
2. Best Performance: Achieved 0.62031, significantly above the baseline requirement (0.5414)
3. Non-Linear Relationship: Higher training accuracy doesn't necessarily correlate with better Kaggle performance
4. Model Stability: Consistent performance above 0.60, indicating robust learning

To understand the contribution of each component in my model, I analyze the critical elements of the architecture and training process:
1. Data Augmentation Components:
   - Random rescaling (±20%): Essential for scale invariance
   - Random shifting (±20%): Improves position invariance
   - Random rotation (±10°): Adds rotation robustness
   - Ten-crop strategy: Critical for test-time augmentation
   - Random erasing: Helps prevent overfitting to specific facial features

2. Model Architecture Elements:
   - BatchNorm after convolutions: Stabilizes training and speeds up convergence
   - Dropout (0.1): Prevents co-adaptation of features
   - 4-stage convolutional design: Hierarchical feature extraction
   - Dense layers (4096 units): Sufficient capacity for emotion classification

3. Training Strategy Components:
   - Learning rate scheduling: Crucial for model convergence
     * Initial LR: 0.01
     * Adaptive reduction to 0.001780
   - SGD with Nesterov momentum (0.9): Provides better convergence than standard SGD
   - Weight decay (0.0001): Regularization for better generalization

Importance Analysis of Components:

1. Most Critical Components:
   - Ten-crop augmentation: Essential for stable predictions
   - BatchNorm: Enables faster training and better convergence
   - Adaptive learning rate: Crucial for fine-tuning and avoiding local minima

2. Secondary Components:
   - Random erasing: Helps but less critical than geometric augmentations
   - Specific dropout rate: Value of 0.1 was sufficient given other regularization

3. Architecture Decisions:
   - Four convolutional stages: Balance between depth and computational efficiency
   - Dense layer size (4096): Provides sufficient capacity without excessive parameters

This analysis suggests that the combination of extensive data augmentation and careful learning rate management were the most significant contributors to model performance.

Part. 2, Questions (30%):

1. **(10%) Explain the support vector in SVM and the slack variable in Soft-margin SVM. Please provide a precise and concise answer. (each in two sentences)**

Support Vector in SVM:
Support vectors are the data points closest to the decision boundary that influence its position and orientation. These points are critical because the SVM algorithm optimizes the margin based on them, maximizing the separation between classes.

Slack Variable in Soft-margin SVM:
Slack variables are introduced in the soft-margin SVM to allow some misclassified points while maintaining a balance between maximizing the margin and minimizing the classification error. They quantify the degree of misclassification, enabling the SVM to work with datasets that are not perfectly linearly separable.

2. **(10%) In training an SVM, how do the parameter C and the hyperparameters of the kernel function (e.g., γ for the RBF kernel) affect the model's performance? Please explain their roles and describe how to choose these parameters to achieve good performance.**

The role of C: The parameter CC in SVM controls the trade-off between maximizing the margin and minimizing the classification error. A larger CC gives more weight to minimizing misclassification errors, leading to a narrower margin and higher complexity (more sensitive to noise). In contrast, a smaller CC prioritizes maximizing the margin by allowing more misclassifications, resulting in a simpler model that may generalize better.

The role of Kernel Hyperparameters (e.g., γ): For kernel functions like the Radial Basis Function (RBF) kernel, γγ determines the influence of a single training example. A higher γγ creates a more flexible decision boundary by focusing on points close to the hyperplane, potentially leading to overfitting. A lower γγ results in a smoother, simpler decision boundary by considering a broader influence of training points, which may lead to underfitting.

Choosing C and γ for Good Performance:
1. **Grid Search with Cross-Validation:** Perform a systematic grid search over a range of C and γ values using cross-validation to identify the combination that minimizes validation error.
2. **Trade-off Between Overfitting and Underfitting:**
    - High C and γ: Complex model (risk of overfitting).
    - Low C and γ: Simpler model (risk of underfitting).
3. **Domain Knowledge:** Use insights about the data distribution to narrow the search space (e.g., data dimensionality, noise levels).
4. **Automated Optimization:** Use algorithms like Bayesian optimization for hyperparameter tuning.

3. (10%) SVM is often more accurate than Logistic Regression. Please compare SVM and Logistic Regression in handling outliers.

SVM:

Support Vector Machines (SVM) handle outliers more robustly than Logistic Regression, particularly when using the **soft-margin** approach. In this approach, slack variables (ξ) and a regularization parameter (C) are introduced to allow for some misclassifications while balancing the size of the margin with the classification errors. This allows SVM to ignore certain points that are far from the rest of the data (outliers) and focus on creating a decision boundary that generalizes well to unseen data.

However, the ability of SVM to handle outliers depends critically on the correct tuning of the C parameter. If C is set too high, the model penalizes classification errors heavily and attempts to classify all points correctly, including outliers, leading to overfitting. Additionally, outliers that are close to the decision boundary can act as **support vectors**, disproportionately influencing the placement of the hyperplane and reducing the model's ability to generalize. Therefore, careful selection of C is essential to ensure the model remains robust to outliers while achieving good generalization performance.

Logistic Regression:

Logistic Regression, on the other hand, is generally more sensitive to outliers because its loss function (log-loss) penalizes misclassifications severely, regardless of how far the misclassified points are from the decision boundary. This makes outliers exert a significant influence on the model's coefficients, potentially shifting the decision boundary toward these extreme points. Unlike SVM, Logistic Regression lacks a built-in mechanism to distinguish between typical data points and outliers, making it inherently more vulnerable to their influence.

Regularization techniques such as L1 or L2 can reduce the impact of outliers in Logistic Regression by constraining the size of the coefficients. However, regularization alone may not suffice when the dataset contains many extreme outliers or if they are far removed from the majority of the data. In these cases, preprocessing techniques such as outlier removal or robust scaling are often necessary to mitigate the impact of outliers before training the model.

Therefore,
SVM, particularly with the **soft-margin approach**, **typically handles outliers better than Logistic Regression because it can explicitly balance the trade-off between margin size and classification errors using the C parameter and slack variables.** This allows SVM to effectively ignore extreme outliers without their significant influence on the decision boundary. **Logistic Regression, by contrast, does not have an intrinsic mechanism to handle outliers and is more susceptible to their influence** unless additional steps such as regularization or data preprocessing are taken.