Final Project Report

Overview

The goal of this project is to develop a **multi-class classification model** capable of identifying whether certain regions in radar images of the Korean contain heavy snow or rain.

To complete this task, I used a residual CNN **ResNet**) as the base model, which architecture includes independent fully connected layers for each label. To address the **data imbalance** issue, I applied data balancing techniques during the preprocessing stage. During the training process, I used curriculum learning to obtain higher performance.

Keywords: imbalance data, ResNet, curriculum learning.

Data Preprocessing

The raw dataset consists of radar observations of cloud intensity across various radar stations in South Korea. The key characteristics of the data are as follows: the images are captured from fixed perspectives, and are generally clean. However, a significant imbalance exists between the number of positive (label 1 and negative (label 0 samples. Based on these characteristics, the following preprocessing steps were implemented:

Addressing Data Imbalance

To handle the imbalance in the dataset, I calculated the total number of samples with non-zero labels and randomly sampled an half number of all-zero-label samples.

Class	Original Positive Ratio	Balanced Positive Ratio
0	0.0318	0.1242
1	0.0208	0.0983
2	0.0430	0.1824
3	0.0358	0.1590
4	0.0225	0.0886
5	0.0203	0.0774
6	0.0121	0.0484
7	0.0420	0.1763
8	0.0627	0.2691

2. **Data Augmentation**

   Given the fixed perspective and low noise in the images, I decided against applying standard augmentation techniques such as random rotations, cropping, or scaling. Additionally, no random noise was added to the images. To accelerate training, all images were resized to 256×256 pixels.

3. **Data Normalization**

   To standardize the input data, I computed the mean and standard deviation of the entire dataset. The computed statistics are as follows:

   - Mean: [0.0234, 0.0241, 0.0240]
   - Standard Deviation: [0.0071, 0.0062, 0.0066]

Model Architecture Selection and Optimization

Iterative Model Development

In this project, I experimented with multiple model architectures to achieve optimal performance. Below is the evolution of the model development:

Model Version	Description	Advantages	Disadvantages	Parameters
Basic CNN	Simple convolutional neural network	Easy to implement	Poor performance	~100K

ResNet (small)	ResNet without specialized FC layers (fewer channels: 16, 32)	Moderate performance	Limited capacity	~800K
ResNet (large)	ResNet without specialized FC layers (more channels: 32, 64)	Achieved the highest score with 10% data	High memory usage	~2.6M
ResNet with FC	Specialized fully connected layers for each label	Optimized accuracy for each label	Prone to overfitting	~2.7M

Key Features of ResNet with Specialized Fully Connected Layers

1. **Residual Blocks** :
   - Incorporate shortcut connections to solve the gradient vanishing problem in deep networks.
   - Preserve information from shallow layers.
2. **Multi-Head Output** :
   - Train independent fully connected (FC) layers for each label.
   - Enhance specificity in multi-label classification tasks.
3. **Batch Normalization and Global Average Pooling** :
   - Batch normalization stabilizes training and accelerates convergence.
   - Global average pooling reduces the number of parameters.
4. **Dropout in Fully Connected Layers** :
   - Dropout is applied with a value of 0.5 to reduce overfitting.

Adjustable Aspects

1. **Number of Residual Blocks** :
   - Experimentation showed that three residual blocks are sufficient to fit the data with 10% of the dataset, while two blocks led to underfitting.
2. **Dropout Value** :
   - Values between 0.2 and 0.5 were tested. A value of 0.5 was chosen because lower values (e.g., 0.2) caused severe overfitting.
3. **Input and Output Channels** :
   - Increasing the number of channels improved model representation and learning ability.
   - The version with channels of 32→64→128→256 achieved the best performance.
   - For the final specialized FC model, channels were reduced to 16→32→64→128 to suppress overfitting.

Model Training

Training Strategy

To train the model, I split the entire dataset into a training set and a validation set with an 8:2 ratio. The primary training strategy was curriculum learning, where training was conducted in two stages:

1. The model was first trained on 50% of the dataset.
2. Subsequently, the model was trained on 100% of the dataset.

This two-stage approach proved more effective than directly training on the full dataset. Direct training on 100% of the data led to severe overfitting, while curriculum learning significantly mitigated overfitting. Additionally, an early stopping strategy was employed to prevent overfitting. The main evaluation metric for model performance was the average F1 score across all classes.

Key Details

1. **Loss Function**:
   - I used the BCEWithLogitsLoss function as the loss function.

- To enhance the model's focus on positive samples, I applied the `pos_weight` parameter.

2. **Optimizer**:
   - The optimizer used was AdamW, which has an adaptive learning rate mechanism.
   - Experiments showed that AdamW performed better than momentum-based SGD.
3. **Learning Rate Scheduler**:
   - An additional learning rate scheduler was applied. If the F1 score did not improve for five consecutive epochs, the learning rate was halved.
4. **Threshold Optimization**:
   - For the validation set, I implemented a function to find the optimal decision threshold for each class.
   - This optimization improved model performance and facilitated better predictions on the test set.
5. **Training Efficiency Considerations**:
   - During training, I observed that the speed of training for models of this scale is primarily determined by the data transfer speed of the CPU, not the GPU computation speed.
   - Since free cloud GPUs and CPUs in kaggle and colab have lower peak performance than my laptop, I conducted training locally. However, my local GPU has limited memory capacity(only 8GB), and I encountered out-of-memory issues.
   - To address this, I carefully adjusted the batch size to 32 and ensured that as much data computation as possible was offloaded to the GPU to optimize efficiency.

Further Thoughts

1. **Ensemble Training**

   My initial plan was to implement ensemble training by training multiple models with different architectures. The idea was to assign different weights to these models based on their performance on the validation set and make ensemble decisions. However, due to limited computational resources (GPU performance) and insufficient training time, I had to set this idea aside.

2. **Per-Label Model Training**

   The most accurate approach would undoubtedly be to create a balanced dataset for each label and train a separate model for each label's decision-making. However, given the concerns about insufficient computational resources and the inability to complete training in time, I did not pursue this option. Nonetheless, I believe this strategy has the potential to deliver the best performance.

3. **Incorporating Temporal Continuity**

   Since the data exhibits clear temporal continuity, another possibility could be to utilize multiple images taken at consecutive times and introduce temporal variables into the decision-making process. However, I did not choose this approach because of my lack of expertise in this area.